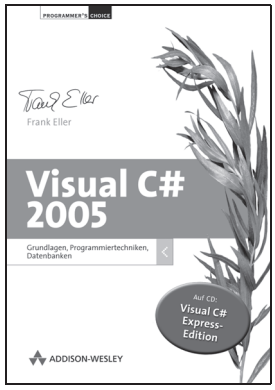
<Professionelle Websites>

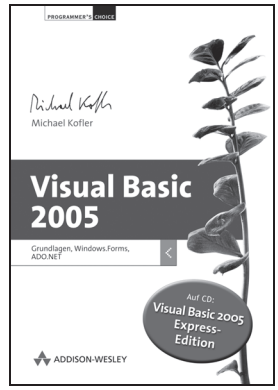
PROGRAMMER'S CHOICE

Die Wahl für professionelle Programmierer und Softwareentwickler. Anerkannte Experten wie z. B. Bjarne Stroustrup, der Erfinder von C++, liefern umfassendes Fachwissen zu allen wichtigen Programmiersprachen und den neuesten Technologien, aber auch Tipps aus der Praxis.
Die Reihe von Profis für Profis!

Hier eine Auswahl:



Fortgeschrittene und Profis erhalten hier umfassendes Know-how zur Windows-Programmierung mit Visual C# in der Version 2. Nach einer Einführung ins .NET-Framework und die Entwicklungsumgebung geht der Autor ausführlich auf die Grundlagen der C#-Programmierung ein. Anhand zahlreicher Beispiele zeigt er die verschiedenen Programmiertechniken wie z.B. Anwendungsdesign, Grafikprogrammierung oder das Erstellen eigener Komponenten. Besondere Schwerpunkte liegen auf der umfangreichen .NET-Klassenbibliothek und Windows Forms sowie auf dem Datenbankzugriff mit ADO.NET.



Sie brauchen ein solides Fundament für Visual Basic 2005? Dieser Titel bietet eine umfassende Spracheinführung, Konzepte zur objektorientierten Programmierung und Lösungen für häufig benötigte Aufgaben (Dateizugriff, Multi-Threading etc.). Der Anwendungsteil konzentriert sich auf die Windows-Programmierung (Windows. Forms, Grafik, Drucken) und Datenbankanwendungen (ADO.NET). Diese Neuauflage wurde vollständig überarbeitet und neu strukturiert. Die Visual-Basic-Bücher von Michael Kofler gelten seit 1991 als Standardwerke.

Stefan Münz

<Professionelle Websites>

Programmierung, Design und Administration von Webseiten


ADDISON-WESLEY

An imprint of Pearson Education

München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

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.

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

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien.
Die gewerbliche Nutzung der in diesem Produkt gezeigten Modelle und Arbeiten ist nicht zulässig.

Fast alle Hardware- und Softwarebezeichnungen und weitere Stichworte und sonstige Angaben, die in diesem Buch verwendet werden, sind als eingetragene Marken geschützt.
Da es nicht möglich ist, in allen Fällen zeitnah zu ermitteln, ob ein Markenschutz besteht, wird das ® Symbol in diesem Buch nicht verwendet.

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

10 9 8 7 6 5 4 3 2 1

08 07 06

ISBN-13: 978-3-8273-2370-5
ISBN-10: 3-8273-2370-3

© 2006 by Addison-Wesley Verlag,
ein Imprint der Pearson Education Deutschland GmbH,
Martin-Kollar-Straße 10–12, D-81829 München/Germany
Alle Rechte vorbehalten
Einbandgestaltung: Marco Lindenbeck, webwo GmbH (mlindenbeck@webwo.de)
Lektorat: Brigitte Bauer-Schiewek, bbauer@pearson.de
Korrektorat: Petra Kienle, München
Herstellung: Elisabeth Prümm, epruemm@pearson.de
Satz: reemers publishing services gmbh, Krefeld, www.reemers.de
Druck und Verarbeitung: Kösel, Krugzell (www.KoeselBuch.de)
Printed in Germany

Inhalt

Vorwort 15

Teil I – Intro 17

1 Begriffsklärungen 19

1.1 Websites, Webseiten und Homepages 19
1.1.1 Websites 19
1.1.2 Webseiten 20
1.1.3 Homepages 21
1.1.4 Webauftritte und Webpräsenzen 21
1.1.5 Portale und Portalseiten 21

1.2 Webdesigner, Webprogrammierer, Webmaster 22
1.2.1 Webdesigner 22
1.2.2 Webprogrammierer und Webentwickler 23
1.2.3 Webmaster 24

1.3 Auszeichnungs- und Programmiersprachen 25

2 Aspekte professioneller Websites 27

2.1 Die Arbeitsumgebung 27
2.1.1 Hardware und Betriebssysteme 27
2.1.2 WYSIWYG- und code-basierte Editoren 29
2.1.3 Webbrowser 31
2.1.4 Grafik- und Multimedia-Software 33
2.1.5 Lokale serverseitige Entwicklungsumgebung 35
2.1.6 Diverse Tools 36

2.2 Usability und Accessibility 37
2.2.1 Wahrnehmung und Aufmerksamkeit 38
2.2.2 Kommunikation mit dem Anwender 39
2.2.3 Usability (Bedienbarkeit) von Websites 40
2.2.4 Accessibility (Zugänglichkeit) von Websites 41

2.3 Planung, Realisierung und Pflege 44
2.3.1 Konzeptphase 44
2.3.2 Realisierungsphase 46
2.3.3 Pflegephase 47

3 Relevante Quellen 49
3.1 Recommendations (Empfehlungen) des W3C 49
3.2 Originaldokumentationen 51
3.3 RFCs und Standarddokumente 51

Teil 2 – HTML und CSS 53

4 Basiswissen HTML und CSS 55
4.1 HTML und XHTML 55
4.1.1 SGML und XML 55
4.1.2 Interpretation von HTML und XHTML 56
4.1.3 Argumente für XHTML 57
4.2 Eine vollständige Webseite mit HTML 58
4.2.1 Quelltext und Verschachtelungsstruktur 58
4.2.2 HTML-Darstellung im Browser 59
4.2.3 Fehlertoleranz bei Browsern 60
4.3 Allgemeine Regeln bei HTML 60
4.3.1 Elemente und Tags 60
4.3.2 Attribute und Attributwerte 62
4.3.3 Notation von Zeichen 63
4.3.4 Dokumenttyp-Deklarationen 69
4.3.5 Editierregeln 72
4.3.6 XHTML-spezifische Syntaxregeln 73
4.3.7 Beispiel einer vollständigen XHTML-Datei 76
4.4 Kopfinformationen einer Webseite 77
4.4.1 Dokumenttitel 77
4.4.2 Meta-Angaben 78
4.4.3 Logische Verlinkung mit anderen Inhalten 84
4.4.4 Adressbasis für referenzierte Dateien 87
4.5 Inhaltliche Strukturierung 88
4.5.1 Block-Elemente (Absatzformate) für den Fließtext 88
4.5.2 Inline-Elemente (Zeichenformate) für den Fließtext 97
4.5.3 Tabellen 100
4.6 Einfache Formatierung mit CSS 109
4.6.1 Das style-Attribut 109
4.6.2 Syntaxregeln für CSS-Formatdefinitionen 111
4.6.3 Farbangaben in CSS 112
4.6.4 Maßangaben in CSS 117
4.6.5 CSS-Eigenschaften für Schriftformatierung 118
4.6.6 Das Boxmodell von CSS 125
4.6.7 CSS-Eigenschaften für Abstände und Ausrichtung 131
4.6.8 CSS-Eigenschaften für Farbe und Form 134
4.6.9 CSS-Eigenschaften für die Positionierung von Elementen 142
4.6.10 CSS-Eigenschaften für Listen und Tabellen 155
4.7 Wiederverwendbare Formate mit CSS 163
4.7.1 Dokumentglobale und dokumentübergreifende Formate 163
4.7.2 Selektoren 167
4.7.3 Formatdefinitionen für Elemente 168

4.7.4 Formatdefinitionen für Klassen und Einzelelemente 171
4.7.5 Attributbedingte Formatdefinitionen 174
4.7.6 Formatdefinitionen für Pseudoelemente 176

4.8 Hyperlinks 178
4.8.1 URIs und Links in HTML 179
4.8.2 Links auf lokale Quellen 184
4.8.3 Anker und Links zu Ankern 185
4.8.4 Links auf Default-Dateinamen 187
4.8.5 Links auf beliebige Inhaltstypen 188
4.8.6 Sonderzeichen in URI-Angaben 189
4.8.7 Features für E-Mail-Verweise 190
4.8.8 Hyperlink-Optimierung 191
4.8.9 Optische Gestaltung von Hyperlinks mit CSS 193

4.9 Grafik, Flash und Multimedia 194
4.9.1 Geeignete Grafikformate 195
4.9.2 Bildbearbeitung in Hinblick auf Webseiten 197
4.9.3 Grafikreferenzen 201
4.9.4 Image-Maps (verweis-sensitive Grafiken) 203
4.9.5 Flash-Einbindung in HTML 205
4.9.6 Java-Applets in HTML 207
4.9.7 SVG-Vektorgrafiken in HTML 208
4.9.8 Eingebettete Objekte aller Art 209

4.10 HTML-Formulare 210
4.10.1 Formularbereiche 212
4.10.2 Formularelemente 213
4.10.3 Formulardesign und Formularoptimierung 220
4.10.4 Optische Verfeinerung von Formularen mit CSS 223

4.11 Allgemeine HTML-Attribute 224

5 Seitenlayouts 227

5.1 Seitenlayouts mit CSS 227
5.1.1 Typische Seitenaufteilungen 228
5.1.2 HTML- und CSS-Basis für ein Portallayout 231
5.1.3 HTML- und CSS-Basis für ein Winkellayout 245
5.1.4 HTML- und CSS-Basis für ein freies Bereichslayout 254

5.2 Navigations- und Menüleisten mit CSS 264
5.2.1 Vertikale Navigationsleiste mit Rollover-Effekt 264
5.2.2 Horizontale Navigationsleiste mit Rollover-Effekt 269
5.2.3 Horizontale Navigationsleiste mit Ausklappmenüs 273

6 Erweiterte Features von HTML und CSS 277

6.1 Mehrfenstertechnik (Frames) 277
6.1.1 HTML mit Framesets und Frames 279
6.1.2 Links zu anderen Frame-Fenstern 283
6.1.3 Rahmen und andere Frame-Eigenschaften 283
6.1.4 Fixe Bereiche ohne Frames 284
6.1.5 Eingebettete Frames 289

6.2 Automatische Überschriftennummerierung 292
6.2.1 Überschriftennummerierung mit CSS 292
6.2.2 Überschriftennummerierung mit JavaScript/DOM 294

6.3 @-Regeln in CSS 297
6.3.1 Die @import-Regel 297
6.3.2 Die @media-Regel 298
6.3.3 Die @page-Regel 300
6.3.4 Die @charset-Regel 302
6.4 CSS und die Browser 302
6.4.1 Netscape 4.x ausschließen 303
6.4.2 Internet Explorer ausschließen und explizit ansprechen 303

Teil 3 – Dynamische Seiten mit JavaScript/DOM 307

7 Basiswissen JavaScript/DOM 309
7.1 Implementierungen von JavaScript und DOM 310
7.1.1 JavaScript 310
7.1.2 DOM (Document Object Model) 311
7.2 JavaScript in HTML 312
7.2.1 Event-Handler und JavaScript 312
7.2.2 JavaScript-Bereiche in HTML 315
7.2.3 JavaScript in separaten Dateien 321
7.3 Sprachkonzepte von JavaScript 323
7.3.1 Allgemeines zur Code-Erstellung 323
7.3.2 Anweisungen, Ausdrücke und Blöcke 326
7.3.3 Variablen und Datentypen in JavaScript 326
7.3.4 Operatoren in JavaScript 333
7.3.5 Kontrollstrukturen in JavaScript 341
7.3.6 Funktionen, Parameter und Return-Werte 347
7.3.7 Die Objektstruktur von JavaScript 1.5 350
7.3.8 Datum und Zeit mit JavaScript 352
7.3.9 Mathematische Operationen mit JavaScript 356
7.3.10 Zeichenketten (Strings) in JavaScript 358
7.3.11 Arrays in JavaScript 362
7.3.12 Weitere Objekte, Eigenschaften und Methoden 367
7.3.13 Existenz von Objekten, Eigenschaften und Methoden 374
7.3.14 Eigene Objekte, Eigenschaften und Methoden 375
7.4 Zugriff auf Seiteninhalte (DOM) 377
7.4.1 Zugriff auf Elemente und Elementinhalte 378
7.4.2 Elemente erzeugen und integrieren 382
7.4.3 Zugriff auf Attribute und Werte 385
7.4.4 Zugriff auf CSS-Eigenschaften 389

8 Praxisfälle für JavaScript/DOM 393
8.1 Clientseitige Formularüberprüfung 393
8.1.1 Das Beispiel 394
8.1.2 Quelltexte und Erläuterungen 394
8.2 Navigation im Explorer-Stil 408
8.2.1 Das Beispiel 408
8.2.2 Quelltexte und Erläuterungen 409

8.3 Interaktive Tabellensortierung 420
8.3.1 Das Beispiel 421
8.3.2 Quelltexte und Erläuterungen 422

Teil 4 – Die Server-Seite 433

9 Hosting und Webserver 435

9.1 Routing, IP-Adressen und Domain-Namen 435
9.1.1 Szenario eines Routings 435
9.1.2 Gateways und Routing-Tabellen 437
9.1.3 TCP/IP, IP-Adressen und Internetanbindung 439
9.1.4 IP-Adressen und Domain-Namen 440

9.2 Web-Hosting 443
9.2.1 Verhältnismäßigkeit von Angeboten und eigenen Zielen 444
9.2.2 Traffic und Transfervolumen 448
9.2.3 Webspace und Speicher 449
9.2.4 Serverzugänge über FTP, SCP/SFTP, Telnet und SSH 450
9.2.5 Domain-Namen und Umziehen von Domains 455
9.2.6 Server-Software und Features 456
9.2.7 Datensicherung, Überwachung und Support 457

9.3 Client, Server und Protokolle 458
9.3.1 TCP/IP und UDP 459
9.3.2 Daemons, Server und Ports 461
9.3.3 Das HTTP-Protokoll 463

9.4 Der Webserver Apache 469
9.4.1 Versionen, Bezug und Installation von Apache 470
9.4.2 Start, Stopp und Neustart von Apache 473
9.4.3 Test von Webseiten unter Apache 473
9.4.4 Die zentrale Konfigurationsdatei httpd.conf 474
9.4.5 Dezentrale Konfigurationsmöglichkeiten über .htaccess 481
9.4.6 Virtuelle Hosts mit Apache 481
9.4.7 Log-Dateien des Apache Webservers 484

10 Basiswissen Linux für Webworker 489

10.1 Linux als Server-Betriebssystem 490
10.1.1 Geschichte und Bedeutung von Linux 492
10.1.2 Aufbau und Komponenten von Linux 493
10.1.3 Standard-Verzeichnisbaum von Linux 496

10.2 Arbeiten auf Shell-Ebene 499
10.2.1 Prompt und Eingabe-Features 499
10.2.2 Kommandos, Optionen und Parameter 501
10.2.3 Umleitungen und Pipes 503
10.2.4 Kommandos für Benutzerverwaltung 504
10.2.5 Kommandos für Dateiverwaltung 510
10.2.6 Kommandos für Zugriffsrechte 521
10.2.7 Kommandos zur Systemüberwachung 525
10.2.8 Kommandos für Softwareverwaltung 530

10.3 Dateibearbeitung mit dem vi-Editor 533
10.3.1 Starten und Beenden von vi 534
10.3.2 Kommandomodus, Kommandozeile und Eingabemodus 535
10.4 Wichtige Konfigurationsdateien 539
10.4.1 Allgemeine Systemkonfigurationsdateien 539
10.4.2 Konfigurationsdateien für Benutzer und Gruppen 541
10.4.3 Zeitgesteuerte Programmausführung (crontab) 543
10.5 Einfache Shellscripts 546
10.5.1 Allgemeines zu Shellscripts 547
10.5.2 Parameter, Variablen und Funktionen 549
10.5.3 Bedingungen, Fallunterscheidungen und Schleifen 553
10.5.4 Komplettbeispiel: ein Errrorlog-Analyse-Script 557

11 Webseiten in HTTP-Umgebung 561
11.1 Webseiten serverseitig 561
11.1.1 URIs und Pfade 561
11.1.2 Default-Dateinamen und Verzeichnis-Browsing 562
11.1.3 GET- und POST-Daten 565
11.2 Server Side Includes (SSI) 566
11.2.1 Voraussetzungen 566
11.2.2 HTML-Templates mit SSI 568
11.2.3 Variablenausgabe mit SSI 569
11.2.4 Script-Einbindung mit SSI 572
11.3 Die CGI-Schnittstelle 574
11.3.1 CGI-Konfiguration 575
11.3.2 CGI-Aufrufmöglichkeiten in HTML 577
11.3.3 Beispiel eines Form-Mailers in Perl 577
11.4 Geschützte Webseiten 583
11.5 Automatische Weiterleitungen 586
11.6 Anpassung von Server-Fehlerseiten 589

Teil 5 – PHP und MySQL 593

12 Basiswissen PHP 595
12.1 PHP als Programmier- und Script-Sprache 596
12.1.1 Geschichte und heutige Bedeutung von PHP 597
12.1.2 Bezug und Installation von PHP 598
12.1.3 Einbindung von PHP in die Apache-Konfiguration 600
12.1.4 Konfiguration von PHP 605
12.2 PHP-Scripting für den Anfang 611
12.2.1 PHP in HTML 612
12.2.2 HTML in PHP 617
12.2.3 PHP mit Einsatz von HTML-Templates 622
12.2.4 Formularverarbeitung und Mail-Funktion mit PHP 629
12.2.5 Code-Erstellung und Code-Verteilung 636

12.3 Sprachkonzepte von PHP 642
12.3.1 Anweisungen, Blöcke, Ausdrücke und Kommentare 642
12.3.2 Variablen und Datentypen 644
12.3.3 Superglobale Variablen 654
12.3.4 Operatoren 659
12.3.5 Kontrollstrukturen 664
12.3.6 Funktionen, Parameter und Return-Werte 671
12.3.7 Klassen und Objekte in PHP 4 675
12.3.8 Objektorientierung in PHP 5 685
12.3.9 Reguläre Ausdrücke 695

12.4 PHP-Funktionsüberblick 703
12.4.1 Daten lesen und auswerten 703
12.4.2 Daten ausgeben und schreiben 714
12.4.3 Zeichenketten und Arrays 723
12.4.4 Datum und Zeit 741
12.4.5 Mathematische Funktionen 747
12.4.6 Datei- und Verzeichnisverwaltung 749
12.4.7 Session-Funktionen 756
12.4.8 Starten externer Programme und Kommandos 764
12.4.9 Diverse Funktionen 767

13 MySQL-Datenbanken und PHP 771

13.1 Basiswissen Datenbanken und MySQL 771
13.1.1 DBM-Systeme und Datenbanken 772
13.1.2 Client-Server-Struktur von Datenbanken 772
13.1.3 Die Transformationssprache SQL 773
13.1.4 Versionen, Bezug und Installation von MySQL 775

13.2 Basiswissen SQL 778
13.2.1 Erste SQL-Schritte mit dem mysql-Client 778
13.2.2 Datenbanken und Tabellen anlegen, ändern und löschen 781
13.2.3 Datensätze einfügen, ändern und löschen 789
13.2.4 Daten abfragen 795
13.2.5 Datenabfragen über mehrere Tabellen 803
13.2.6 Tipps für effiziente Datenbankabfragen 805
13.2.7 InnoDB-Tabellen 809
13.2.8 Benutzer- und Rechteverwaltung 811

13.3 Praxisbeispiel: Entwurf Webshop-Datenbank 816
13.3.1 Schritt 1: Datenaufnahme 816
13.3.2 Schritt 2: Ablaufbeschreibungen 818
13.3.3 Relationen entwerfen 819
13.3.4 Felder und Schlüssel definieren 822

13.4 MySQL-Datenbankzugriffe mit PHP 825
13.4.1 MySQL-Datenbankverbindungen mit PHP 827
13.4.2 Datenbankabfragen in PHP 830

13.5 Praxisbeispiel: Webbasierte Datenverwaltung 838
13.5.1 Das Handling bei der Datenverwaltung 839
13.5.2 Aufgaben des PHP-Scripts 842
13.5.3 Optimierungen für mehr Sicherheit 853

Teil 6 – XML 857

14 XML für Webentwickler 859

14.1 Einsatz- und Verarbeitungsmöglichkeiten 860
14.1.1 XML-Editoren 862
14.1.2 XML-Parser 863

14.2 Aufbau von XML-Dokumenten 864
14.2.1 Elemente, Tags und Attribute 864
14.2.2 XML-Deklarationen und Dokumenttypdeklaration 867
14.2.3 Zeichenvorrat, PCDATA und CDTATA 872
14.2.4 Namensräume in XML 874

14.3 Wichtige öffentlich standardisierte XML-Formate 877

14.4 RSS als Basis für Newsfeeds und Podcasts 878
14.4.1 Aufbau eines RSS-Dokuments 883
14.4.2 RSS-Integration ins eigene Webangebot 888

14.5 Webservices – Anwendungskommunikation 889
14.5.1 SOAP, SOAP-Messages und SOAP-Envelopes 890
14.5.2 Header- und Body-Daten einer SOAP-Message 892
14.5.3 Die SOAP-Attribute mustUnderstand, role und relay 894
14.5.4 SOAP-Fehlerrückmeldungen (Faults) 896
14.5.5 Die Rolle von WSDL 898
14.5.6 UDDI und WSIL 900
14.5.7 Die SOAP-Alternative WDDX 901

14.6 Dokumenttypdefinitionen (DTDs) 903
14.6.1 Aufbau von DTDs 903
14.6.2 Elemente und Elementabhängigkeiten definieren 906
14.6.3 Attribute und Attributzuweisungen definieren 912
14.6.4 Entities definieren und referenzieren 915
14.6.5 Modulare DTDs 917
14.6.6 XML-Schema (XSD) als modernere DTD-Alternative 918

14.7 Übersetzung von XML-Daten in HTML/XHTML 922
14.7.1 Die Transformationssprache XSLT 923
14.7.2 XSLT-Prozessoren 925
14.7.3 Ein einfaches Transformationsbeispiel 926
14.7.4 Wichtige XSLT-Elemente 929
14.7.5 XPath-Funktionen für die Transformation 937
14.7.6 XPath-Adressierung bei der Transformation 942

14.8 PHP und XML 944
14.8.1 XML-Verarbeitung mit der SimpleXML-Klasse 946
14.8.2 XML-Verarbeitung mit der DOM-Klasse (libXML) 950
14.8.3 XML-Verarbeitung mit der XML-Klasse (Expat) 960
14.8.4 XSLT-Funktionen 966
14.8.5 Praxisbeispiel: News-Script für RSS-Feed-Funktion 968
14.8.6 Praxisbeispiel: Google-Suche als SOAP-Webservice 972
14.8.7 Praxisbeispiel: Plausibilitätsprüfung mit WDDX 982

Teil 7 – Betrieb von Websites 989

15 Site-Verwaltung 991

15.1 Rechte und Pflichten eines Site-Anbieters 991
15.1.1 Die Rechtslage 991
15.1.2 Impressumspflicht 994
15.1.3 Vorvertragliche Widerrufsbelehrung 996
15.1.4 Urheberrecht 997
15.1.5 Haftung für Inhalte und Links 999
15.1.6 Datenschutz und Auskunftspflicht 1000
15.1.7 Namens- und Markenschutz 1001
15.1.8 Softwarepatente 1003
15.1.9 Abmahnungen 1004

15.2 Site-Promoting und Aktualität 1007
15.2.1 Launches und Relaunches 1007
15.2.2 Websites und Suchmaschinen 1008
15.2.3 Werbung für Websites 1013
15.2.4 Newsticker, Newsletter und RSS-Newsfeed 1015
15.2.5 Benutzerbasierte Inhalte 1019
15.2.6 Favicons – die Site-Abrundung 1023

Teil 8 – Referenz 1025

16 Referenz zu HTML 1027

16.1 HTML-Elemente 1027
16.1.1 Hinweise zu den Tabellen 1027
16.1.2 HTML-Elemente in alphabetischer Reihenfolge 1027

16.2 HTML-Attribute 1051
16.2.1 Hinweise zu den Tabellen 1051
16.2.2 Attribute in alphabetischer Reihenfolge 1051

16.3 Benannte HTML-Entities 1072
16.3.1 Hinweise zu den Tabellen 1072
16.3.2 Benannte Entities 1072

17 CSS-Referenz 1081

17.1 CSS-Eigenschaften 1081
17.1.1 Hinweise zu den Tabellen 1081
17.1.2 CSS-Eigenschaften in alphabetischer Reihenfolge 1081

Index 1117w

Vorwort

Das Erstellen größerer Websites ist längst eine so vielschichtige Materie, dass ihr kein einzelnes Fachbuch mehr vollständig gerecht werden könnte. Jedes Buch, das sich mit dieser Materie befasst, muss entweder Spezialisierung oder Gesamtschau betreiben. Das vorliegende Buch versucht einen Spagat. Einerseits beschränkt es sich thematisch auf bestimmte Technologien, spezialisiert sich also, um nicht oberflächlich zu werden. Doch sein eigentlicher Anspruch ist letztlich ein Gesamtüberblick auf die wichtigsten Aspekte moderner Webentwicklung, angefangen von den ersten Gehversuchen in HTML über client- und serverseitige Programmierung bis hin zur Konfiguration eines Webservers, ergänzt um fachfremde, aber relevante Aspekte wie etwa der juristischen Seite beim Betrieb einer Website.

Das Buch setzt seine fachlichen Schwerpunkte so, dass Sie als Leser und Anwender die beschriebenen Technologien möglichst effizient und routiniert einsetzen können. Es geht an entscheidenden Stellen in die Tiefe, erhebt aber keinen Anspruch auf vollständige und gleichmäßig ausführliche Behandlung aller verfügbaren Features und Möglichkeiten.

Mancher Leser mag aus diesem Grund einiges vermissen, beispielsweise die Beschreibung zahlreicher klassischer HTML-Attribute. Deren Verwendung ist jedoch – entgegen der immer noch landläufigen Praxis – in den meisten Fällen nicht mehr professionell und daher auch kein sinnvoller Vermittlungsgegenstand mehr. Stattdessen lernen Sie in diesem Buch, wie Sie CSS richtig einsetzen.

Das Buch wiederholt auch nicht – wie vielfach immer noch üblich – abermals die Paradigmen, Vorgehensweisen und Tricks aus dem vorigen Jahrhundert, also aus der proprietärseligen Zeit von Netscape 4 und Internet Explorer 4. Es ist endgültig Zeit, über solch völlig veraltete Browser kein Wort mehr zu verlieren und Webdesign mit HTML, CSS und Scriptsprachen so zu vermitteln, dass die längst vorhandenen Standards, die von den heute verbreiteten Browsern auch leidlich umgesetzt werden, im Vordergrund stehen. Deshalb werden Sie in diesem Buch nichts mehr über font- oder layer-Elemente finden und nichts mehr über ***document.all***. Sie ***dürfen*** sich mittlerweile guten Gewissens auf standardkonformes Webdesign beschränken – und Sie ***sollten*** es!

Die Zielgruppe dieses Buchs sind alle, die mit der Entwicklung anspruchsvoller Websites zu tun haben. Am meisten werden vermutlich jene Leser profitieren, die sich bei einem der zahlreichen Hosting-Provider einen eigenen Root-Server gemietet haben und darauf nun ihr eigener Herr sind. Gerade diese Zielgruppe benötigt umfangreiches Know-how auf verschiedensten Gebieten.

Der Autor dieses Buchs ist vor allem bekannt durch seine Dokumentation SELFHTML, die ebenfalls auf zahlreiche Aspekte des Erstellens von Webseiten eingeht. Während SELFHTML jedoch systematischer vorgeht und eher eine Referenz mit vielen Beispielen ist, verfolgt das vorliegende Buch einen anderen Ansatz. Es hat die Aufgabe, einen Weg durch den Dschungel der Möglichkeiten zu zeigen, einen Weg, der sich an aktuellen Standards und gesammelten Einsichten orientiert.

Teil I – Intro

Einige wichtige Begriffe, Aspekte und Originalinformationsquellen sollten Sie kennen, bevor Sie sich an die Entwicklung eines Webprojekts wagen, das die Bezeichnung »professionell« verdient. Dieser erste Buchteil vermittelt das nötige Basiswissen.

1 Begriffsklärungen

Das Web ist, verglichen mit anderen Medien, noch recht jung. In den Anfangsjahren herrschte eine teilweise babylonische Sprachverwirrung rund um die wichtigen Termini technici dieses neuen Mediums. Fast täglich tauchten neue Schlagwörter auf und die meisten davon wurden häufiger falsch benutzt als richtig. Mittlerweile ist allerdings ein Stadium eingetreten, in dem sich einigermaßen klar unterscheidbare Begriffe herauskristallisiert haben. Mit diesen Begriffen sollten Sie souverän umgehen können.

1.1 Websites, Webseiten und Homepages

Den drei Substantiven der Überschrift lassen sich noch diverse weitere Wörter wie Webauftritt, Webpräsenz, Webangebot, Internetseiten, Portal oder anglophile Varianten wie Webpages hinzufügen. All diese Wörter werden regelmäßig benutzt.

1.1.1 Websites

Eine Website ist ein als Ganzes erkennbares Projekt, das in der Regel aus mehreren bis vielen einzelnen, miteinander verlinkten Webseiten besteht. Der erkennbare Zusammenhang ergibt sich aus verschiedenen Faktoren:

- Eine Website hat ein erkennbar identifizierendes und einheitliches Seitenlayout.
- Es gibt eine feste, immer wieder an der gleichen Stelle erscheinende Navigationsmöglichkeit. Über diese kann der Anwender jederzeit den Rückbezug zu zentraleren Seiten des Projekts herstellen, sie verschafft ihm ein räumliche Grobvorstellung vom Gesamtprojekt und vermittelt ihm, wo er sich innerhalb des Angebots gerade befindet.
- Die meisten Websites sind heute über einen eigens dafür reservierten Domain-Namen aufrufbar.

Ein Firmenauftritt im Web, ein Dokumentationsprojekt, ein elektronischer Anzeigenmarkt, eine Partnerbörse, ein Fan-Projekt, ein Wiki-System, eine komplette Community-Infrastruktur, ein Informations-Service, eine Suchmaschine, ein Linkverzeichnis, ein Magazin – all das sind jeweils Websites. Die Webpräsenz der Frankfurter Allgemeine Zeitung (*http://www.faz.net/*) ist eine Website, die offiziellen Seiten der Stadt München (*http://www.muenchen.de/*) stellen eine Website dar, ein Blog wie der Schockwellenreiter (*http://www.schockwellenreiter.de/*) ist eine Website usw.

Der Begriff Website wird häufig mit dem einer einzelnen Webseite verwechselt. Der Grund ist, dass das englische ***site*** ein »falscher Freund« ist, der die Übersetzung »Seite« suggeriert, was aber falsch ist. Korrekt ist jedoch die Übersetzung »Sitz« im Sinne eines Unternehmenssitzes.

1.1.2 Webseiten

Eine einzelne Webseite (oft auch als Internetseite bezeichnet) ist ein HTML-Dokument, das von einem Webbrowser angezeigt werden kann. Die genaue Abgrenzung lässt sich eigentlich nur technisch erklären, nämlich dadurch, dass die Webseite aus allem besteht, was zwischen den beiden HTML-Tags `<html>` und `</html>` notiert ist. Das können sein:

- normale, strukturierte Textinhalte, also Überschriften, Textabsätze, Aufzählungen, Tabellen usw.,
- Formulare, in denen der Anwender etwas eingeben, auswählen oder anklicken kann,
- referenzierte Inhalte wie Grafiken, bewegte Animationen, Sound oder Anwendungen,
- dynamische Bestandteile, die durch clientseitiges Scripting (JavaScript) zustande kommen, inklusive zusätzlich geöffneter Popup-Fenster und dergleichen,
- Definitionen, die das typische Seitenlayout bestimmen und sich sowohl innerhalb des HTML-Codes befinden als auch aus referenzierten Dateien stammen können.

Ein wichtiges Kriterium einer einzelnen Webseite ist: Sie hat eine Internetadresse (URI) wie *http://www.claudia-klinger.de/digidiary/04_03_05.shtml*. Umgekehrt ist allerdings nicht jede solche Adresse auch eine Webseite. Denn auch in Webseiten referenzierte Grafiken, Flashmovies, Java-Applets usw. haben eigene URIs, ebenso wie Download-Dateien in Archivformaten, PDF-, Postscript-, Word- oder sonstige Dokumente, die häufig im Web zu finden sind. Sie sind deshalb im Web, weil sie einen URI haben, der mit ***http://*** oder ***https://*** beginnt, doch es sind keine Webseiten.

Wenn die so genannte Frame-Technik zum Einsatz kommt, wird es problematisch. Genaugenommen weicht die Frame-Technik den Begriff der Webseite auf, was auch der Grund dafür ist, dass diese Technik mittlerweile von Fachleuten heftig kritisiert wird. Bei Verwendung von Frames wechseln nur Teile einer im Browser-Fenster angezeigten Seite beim Wechsel zu einer anderen Seite ihren Inhalt. In der Adresszeile des Browsers bleibt immer die gleiche Adresse stehen.

Statische und dynamisch generierte Webseiten

Eine Webseite muss keine Datei mit der Endung ***.htm*** oder ***.html*** sein. Ein vollständiges HTML-Dokument, reichend von `<html>` bis `</html>`, kann ebenso gut durch ein serverseitiges Script generiert werden. Der Browser (und der Anwender) merken davon nichts. Der Inhalt der Seite wird auf dem Server-Rechner dynamisch generiert und dann vom Webserver an den Webbrowser genauso wie eine normale HTML-Datei ausgeliefert.

Dynamisch generierte Seiten sind beispielsweise Ergebnisseiten einer Suche oder Bestätigungsseiten, die zuvor vom Anwender eingegebene Daten wiederholen, etwa für einen Kaufvorgang. Für dynamisch generierte Seiten kommen typischerweise serverseitige Programmiersprachen wie PHP, JSP oder Perl zum Einsatz. Nicht selten werden auszugebende Inhalte dabei aus einer ebenfalls auf dem Server-Rechner laufenden Datenbank geholt. Deshalb können am Zustandekommen einer Webseite auch Datenbank-Management-Systeme wie MySQL beteiligt sein.

1.1.3 Homepages

Der Begriff der Homepage ist bis heute mehrdeutig. Er kann folgende Bedeutungen annehmen:

- Die im Browser eingestellte Startseite, die nach jedem Start des Browsers und bei jedem Anklicken des Home-Symbols wieder angezeigt wird.
- Die Einstiegsseite einer Website, also die Webseite, die automatisch angezeigt wird, wenn im Browser z. B. nur *www.stempelgeheimnis.de* eingegeben wird.
- Synonym für eine kleinere, meist persönliche Website, eine elektronische Visitenkarte im Web.

In der letztgenannten Bedeutung hat »Homepage« allerdings manchmal auch einen leicht negativen Unterton. Der Unterton zielt auf Seiten, die typischerweise bei kostenlosen, werbefinanzierten Hostern unter ellenlangen Internetadressen erreichbar sind, auf denen meist pubertierende Jünglinge ihre ersten HTML-Gehversuche starten und dabei inmitten eines bunten, zappelnden Layout-Chaos über ihren Hamster oder das Innenleben ihres PCs berichten.

1.1.4 Webauftritte und Webpräsenzen

Diese Begriffe stammen aus der kommerziellen Welt und werden immer dann gebraucht, wenn es um Seiten mit Präsentationscharakter geht, also um Unternehmensporträts, Informationsseiten zu Messen oder Großveranstaltungen usw.

Gemeint sind in der Regel Websites. Allerdings können die Begriffe auch einschränkend sein. Ist vom Webauftritt oder der Webpräsenz einer Firma die Rede, so versteht sich darunter vor allem der öffentlich zugängliche, repräsentative Teil der Website, nicht aber unbedingt der Business-to-Business-Bereich für Geschäftspartner, die sich einloggen können, um personalisierte, nicht öffentliche Informationen abzurufen oder Geschäfte abzuschließen.

1.1.5 Portale und Portalseiten

Die etwas klassizistisch anmutende Bezeichnung »Portal« ist einerseits einer der zahlreichen modernen Ausdrücke, mit denen manche Anbieter versuchen, die Bedeutung ihrer Website anzuheben. Andererseits ist es jedoch durchaus auch ein nützlicher Begriff, um eine Sorte von Websites zu kennzeichnen, deren Aufgabe darin besteht, Besuchern the-

meneinführende Informationen zu bieten und ansonsten noch zahlreiche Links auf speziellere Seiten sowie z.B. themenbezogene Diskussionsforen oder Ähnliches anzubieten. Meist ist mit einer solchen Site auch ein attraktiver, allgemeiner Domainname verbunden. Gute Beispiele dafür sind *http://www.recht.de/* oder *http://www.medizin.de/*.

Innerhalb von Websites gibt es aber auch häufig einzelne Seiten, die Portalcharakter haben. Deshalb werden sie als Portalseiten bezeichnet. Diese Bezeichnung trifft vor allem auf die Start- bzw. Einstiegsseite einer Website zu.

1.2 Webdesigner, Webprogrammierer, Webmaster

Auch in diesem Fall lässt sich die Liste der Ausdrücke beliebig weiterführen. So ist auch von Webautoren, Webredakteuren, Online-Redakteuren, Webentwicklern oder Internetprogrammierern die Rede.

Eines ist allerdings all diesen Bezeichnungen gemeinsam: Bei keiner von ihnen handelt es sich um eine offizielle Berufsbezeichnung. Jeder, der möchte, kann sich Webdesigner oder Webprogrammierer nennen, ohne damit juristische Probleme zu bekommen. Es gibt zwar zahlreiche Berufsschullehrgänge, Weiterbildungsmaßnahmen usw. mit dem beruflichen Ziel z.B. eines Webdesigners oder Webmasters. Doch es existiert keine kontrollierende Kammer oder Institution, die solche Berufsbezeichnungen schützt oder zentrale Zulassungsprüfungen durchführt. Wer sich mit Tätigkeiten dieser Art selbstständig macht und ohne eigene Angestellte arbeitet, ist ein typischer Freelancer (Freiberufler).

Die Folge ist, dass all diese Bezeichnungen eine gewisse Unschärfe besitzen und sich in ihrer Bedeutung teilweise überlappen. Letzteres liegt allerdings auch daran, dass nicht wenige Websites immer noch in Personalunion von einem Alleskönner erstellt und betrieben werden oder dass zumindest wenige Bearbeiter ganz verschiedene Aufgaben wahrnehmen, die alle Bereiche des Betriebs einer Website berühren.

Eine begriffliche Abgrenzung ist also nicht ganz einfach, jedoch sinnvoll.

1.2.1 Webdesigner

Ein Webdesigner entwirft und erstellt Webseiten. Dabei umfasst sein Aufgabengebiet jedoch weniger das Erstellen von Inhalten als vielmehr das Umsetzen von Layoutvorstellungen. Von Fall zu Fall gehört auch das Verteilen von Information auf verschiedene Seiten sowie das Verlinken von Seiten zu seinen Aufgaben.

Webdesigner kommen schnell mit Programmierung in Berührung. Der Grund ist, dass ein Webdesigner meist nicht nur das Layout einer Website entwirft, sondern es auch realisieren soll. Dabei muss er möglicherweise nicht nur HTML-Code erstellen, sondern beispielsweise auch ein paar Dinge in PHP programmieren.

Es gibt jedoch auch Webdesigner, die sich so nennen, aber eigentlich fast ausschließlich schicke Animationen in Flash produzieren. Sofern die Website gar nicht mehr aus HTML besteht, sondern nur noch aus Flash, ist diese Bezeichnung nicht einmal falsch. Handelt es

sich jedoch nur um Flashmovies, die später in Webseiten eingebettet werden, dann ist eine Bezeichnung wie Multimedia-Designer vermutlich richtiger.

Zu den fachlichen Qualitäten eines Webdesigners gehören auf jeden Fall:

- grafisches Vorstellungs- und Gestaltungsvermögen,
- souveräner Umgang mit Grafiksoftware, sowohl für Pixelgrafik als auch für Vektorgrafik,
- solide Kenntnisse in HTML, CSS und JavaScript, um Layoutvorstellungen umsetzen zu können,
- Kenntnisse in Macromedia Flash, da dies derzeit der De-facto-Standard für animierte oder multimediale Inhalte im Web ist,
- Grundkenntnisse in serverseitiger Programmierung sowie beim Arbeiten auf Unix/Linux-basierten Server-Rechnern,
- je nach Auftraggeber auch Kenntnisse in bestimmten Content-Management-Systemen.

Der Gesamtbereich der nötigen Kenntnisse ist also sehr breit gefächert, doch der Schwerpunkt liegt in der Fähigkeit, webgerechte, aber dennoch individuelle Layouts zu entwerfen und mit den Basistechnologien der Webseitenerstellung umzugehen.

1.2.2 Webprogrammierer und Webentwickler

Die Abgrenzung der Begriffe »Programmierer« und »Entwickler« ist nicht ganz einfach, betrifft aber nicht nur das Web, sondern die gesamte Softwarebranche. Ein Entwickler ist tendenziell jemand, der Software sowohl konzipieren als auch programmieren kann, während ein Programmierer eben nur nach Konzepten programmiert, die er nicht selbst erstellt hat.

Bei den Basissprachen HTML und CSS scheiden sich die Geister: Wer nur damit arbeitet, ist kein Entwickler oder Programmierer. Andererseits kommt schnell auch JavaScript ins Spiel und schon sind die Grenzen zur Programmierung überschritten. Im engeren Sinne versteht man unter Webprogrammierern oder Webentwicklern jedoch serverseitige Programmierung.

Wenn eine Website sich auch als Webanwendung bezeichnen lässt, wie etwa ein Auktionshaus, ein Kleinanzeigenmarkt, eine Tauschbörse oder ein Shop, dann kommen Entwickler und Programmierer ins Spiel, welche die gewünschte Funktionalität konzipieren und realisieren müssen. Die Arbeiten können manchmal vor allem darin bestehen, am Markt vorhandene Webanwendungslösungen für eine konkrete Anwendung anzupassen. In wieder anderen Fällen sind völlig neue und eigene Lösungen zu entwickeln.

Zu den fachlichen Qualitäten gehören auf jeden Fall:

- Fähigkeit zu systemanalytischem und lösungsstrategischem Denken,
- solide Beherrschung möglichst mehrerer verschiedener Programmiersprachen, darunter für Webprogrammierung typische Sprachen wie PHP oder Perl,

- solide Kenntnisse in den Basissprachen HTML, CSS und JavaScript,
- je nach Projekt auch Kenntnisse in Datenbank-Management-Systemen und in SQL,
- solide Kenntnisse beim Arbeiten auf Unix/Linux-basierten Server-Rechnern,
- je nach Auftrag auch Kenntnisse in speziellen, vorhandenen Softwarelösungen, etwa für Diskussionsplattformen, Shops, Content-Management usw., die an projekteigene Anforderungen anzupassen sind.

1.2.3 Webmaster

Mit der Bezeichnung »Webmaster« wurde in früheren Jahren viel Unsinn getrieben. Jeder kleine Homepage-Bastler schimpfte sich am Ende jeder stolz erstellten HTML-Seite Webmaster, weil er das auf anderen Seiten so gesehen hatte und toll fand.

Ein Webmaster hat jedoch mit dem Erstellen von Webseiten nicht viel zu tun. Der Tätigkeitsbereich umfasst eher die Installation und Administration des Webservers sowie der Serversoftware, die für den Betrieb einer Website benötigt wird. Auf der anderen Seite muss das Tätigkeitsfeld jedoch auch von dem eines Systemadministrators abgegrenzt werden. Ein Webmaster ist normalerweise nicht dafür verantwortlich, welcher Linuxkernel installiert wird, welche Datenträger zu einem Dateisystem gemountet werden oder wie die automatische Datensicherung organisiert wird.

Der Webmaster ist der administrative Ansprechpartner (auch als ***admin-c*** bezeichnet) für alles, was mit dem Betrieb einer Website, jedoch nichts mit deren Inhalten oder der Programmierung zu tun hat. Wenn Domains reserviert werden, muss unter anderem ein »Administrativer Ansprechpartner« bestimmt werden. Das deutsche Network Information Center Denic (*http://www.denic.de/*), die Vergabestelle für .de-Domains, definiert die Aufgaben dieser Person so:

Der administrative Ansprechpartner (admin-c) ist die vom Domaininhaber benannte natürliche Person, die als sein Bevollmächtigter berechtigt und gegenüber DENIC auch verpflichtet ist, sämtliche die Domain [name].de betreffenden Angelegenheiten verbindlich zu entscheiden.

Dieser Aufgabenbereich wird wiederum von dem des technischen Ansprechpartners und Zonenverwalters abgegrenzt, der in der Regel der Hosting-Provider oder ein Rechenzentrum ist, in dem der oder die Server-Rechner einer Site gehostet werden.

Auch wenn die Funktion gegenüber der Domain-Vergabestelle eher eine administrative ist, so muss ein Webmaster doch auch über technisches Detailwissen verfügen. Auf jeden Fall sind erforderlich:

- solide Kenntnisse in Thematiken wie TCP/IP, HTTP-Protokoll, Domain Name Service oder Routing,
- mindestens Grundkenntnisse in den Basissprachen HTML, CSS und JavaScript,
- Grundkenntnisse bis solide Kenntnisse in serverseitigen Scriptsprachen,

- solide Kenntnisse in der Installation und Konfiguration von Webservern, vor allem des Apache Webservers, sowie von Interpretern wie PHP, Perl oder Python, von Datenbank-Management-Systemen wie MySQL und anderen, für Webanwendungen benötigten Softwareprodukten,
- solide Kenntnisse beim Arbeiten auf Unix/Linux-basierten Server-Rechnern.

1.3 Auszeichnungs- und Programmiersprachen

Es gibt Zeitgenossen, die geradezu militant eine strikte Unterscheidung von Auszeichnungs- und Programmiersprachen propagieren (anstelle von Auszeichnungssprachen wird häufig auch von *Markup-Sprachen* gesprochen). Nicht selten steckt dahinter der verletzte Stolz eines Programmiererherzens, das mit ansehen muss, wie sich plötzlich junge Schüler, die ein paar HTML-Elemente kennen, als Programmierer bezeichnen. Obwohl die Trennung zwischen Markup- und Programmiersprachen nicht immer so klar ist wie propagiert (z. B. ist XSLT eine XML-basierte Markup-Sprache, die aber wie eine Programmiersprache prozedural interpretiert wird), so ist es doch sinnvoll, auf die Gemeinsamkeiten und Unterschiede zumindest von HTML und Programmiersprachen hinzuweisen.

Gemeinsamkeiten sind:

- Sowohl Markup- als auch Programmiersprachen sind künstliche Sprachen, welche die Aufgabe haben, einem Computer oder Computerprogramm etwas mitzuteilen oder anzuweisen.
- Sowohl Markup- als auch Programmiersprachen werden als Quelltext notiert. Bestimmte Zeichen, Zeichenfolgen und Schlüsselwörter im Quelltext werden beim Verarbeiten des Quelltextes als Bestandteile der künstlichen Sprache erkannt und entsprechend interpretiert.

Unterschiede sind dagegen:

- Mit Markup-Sprachen werden Dokumente erzeugt, mit Programmiersprachen Scripts oder Programme.
- Dokumente von Markup-Sprachen werden von einem Parser gelesen, der die Dokumentstruktur analysiert, während Scripts oder Programme als eine Folge von Anweisungen abgearbeitet werden, sei es von einem Interpreter oder vom Betriebssystem.
- Eine Programmiersprache gilt nach landläufiger Ansicht dann als Programmiersprache, wenn sich mit ihrer Hilfe eine so genannte *Turingmaschine* nachbauen lässt. Eine Turingmaschine besteht aus einem Speicher mit Feldern. In jedes Feld kann ein Zeichen geschrieben werden. Die Turingmaschine kann den Speicher beschreiben bzw. überschreiben, und zwar auf Grund eines Programms. Die Turingmaschine verarbeitet und berechnet also einen Input und erzeugt daraus einen Output. Auszeichnungssprachen sind dazu nicht in der Lage.

Aus Anwendersicht bedeutet der Unterschied erst einmal, dass Markup-Sprachen leichter zu erlernen sind als Programmiersprachen, da sie lediglich Daten mit Bedeutung auszeichnen, während Programmiersprachen das Konzipieren und Realisieren eines Programmablaufs erfordern.

2 Aspekte professioneller Websites

So viel vorweg: Eine professionelle Site zeichnet sich nicht durch den Umfang oder den betriebenen Entwicklungsaufwand aus. Auch eine einfache elektronische Visitenkarte im Web, bestehend aus einer einzelnen oder wenigen Seiten, kann das Attribut »professionell« verdienen. Professionalität ist daran erkennbar, was alles beachtet wurde, ob technische Standards und wichtige Gestaltungsregeln eingehalten wurden und ob die Site insgesamt einen durchdachten und bis ins Detail ausgearbeiteten Eindruck vermittelt.

2.1 Die Arbeitsumgebung

Wer Websites entwickelt, benötigt viele verschiedene Arten von Software. In diesem Abschnitt gehen wir darauf ein, welche Anforderungen ein Entwickler-PC erfüllen muss und welche Arbeitsmittel benötigt werden.

2.1.1 Hardware und Betriebssysteme

Für das Erstellen von Webseiten ist weder eine bestimmte Hardware noch ein bestimmtes Betriebssystem erforderlich. Ein gewöhnlicher PC genügt, wobei jedoch folgende Voraussetzungen an die Hardware und Umgebung erfüllt sein sollten:

- True-Color-Monitor und entsprechende Grafikkarte, um bei grafischen oder Layoutarbeiten präziser arbeiten zu können.
- Ausreichend Arbeitsspeicher für viele gleichzeitig geöffnete Anwendungen. Die Ausstattung sollte sich im oberen Bereich der jeweils üblichen Ausstattung bewegen. Zum Redaktionszeitpunkt dieses Buchs wären das 512 bis 1024 Mbyte.
- Internetanbindung, am besten mit Standleitung oder DSL-Flatrate über LAN-Anschluss. In diesen Fällen muss der PC auch über eine Netzwerkkarte verfügen. Herkömmliche Modem- oder ISDN-Anschlüsse mit Zeittakt-Einwahl sind zwar möglich, verzögern die Arbeit in der Praxis jedoch immer wieder.

Betriebssysteme: MS Windows

Bei den Betriebssystemen hat MS Windows einen überragenden Marktanteil. Für die meisten Zwecke beim Erstellen und Entwickeln von Websites ist diese Betriebssystemfamilie auch geeignet. Allerdings sollte auf jeden Fall eines der auf NT basierenden Familienmitglieder zum Einsatz kommen, also z.B. Windows 2000, 2003 oder XP. Ältere

Varianten wie Windows 95 oder 98 sind zu vermeiden, vor allem in Hinblick auf den Betrieb lokaler Server und generell auch bezüglich der Sicherheit beim Internetzugang.

Vorteile von MS Windows beim Entwickeln von Webseiten sind:

- Größtes Auswahlsortiment an Software.
- Testen in der Umgebung, die auch von der Mehrheit der Seitenbesucher verwendet wird.

Nachteile sind dagegen:

- Windows unterstützt nicht alle Möglichkeiten (oder zumindest nicht in der Form, in der sie in Programmiersprachen wie PHP implementiert sind), die beim Entwickeln komplexer Websites möglicherweise benötigt werden, z.B. Mikrosekunden, Datei-Locking oder Prozess-Forking.
- Da die meisten Server-Rechner, auf denen Websites am Ende in Betrieb gehen, Unix/Linux-basierte Systeme sind, ist keine optimale Anpassung der lokalen Entwicklungsumgebung an die Produktivumgebung möglich.

Betriebssysteme: Apple Macintosh

Die alte Weisheit, dass jemand, der ernsthaft mit Grafik zu tun hat, unbedingt einen Macintosh-Rechner braucht, gilt nur für den Print-Bereich, da das Macintosh-System intern Farben nach einem print-orientierten Farbmodell auflöst. Für Webgrafik zählen jedoch am Ende nur RGB-Farben. Dennoch lassen sich selbstverständlich auch mit Macs sehr gut Webseiten erstellen.

Vorteil eines Apple Macintosh ist:

- Besonders gute Abstimmung von Hard- und Software, was zu hoher Performance und Stabilität beim Arbeiten führt.

Nachteile sind:

- Nicht so viele kostenlose, aber qualitativ hochwertige Softwareprodukte.
- Wie bei MS Windows lässt sich die lokale Entwicklungsumgebung nicht optimal an eine Unix/Linux-basierte Produktivumgebung anpassen.

Betriebssysteme: Linux

Endanwenderorientierte Linux-Distributionen wie die von SUSE sowie erfolgreiche grafische Oberflächen wie KDE haben dazu beigetragen, dass Linux auch im Desktop-PC-Bereich zu einer ernstzunehmenden Alternative zu MS Windows oder Apple Macintosh geworden ist.

Vorteile von Linux sind:

- Optimale Unterstützung von Internetstandards sowie von C-basierten Programmiersprachen wie etwa PHP.

- Die Entwicklungsumgebung lässt sich sehr genau auf die Produktivumgebung eines Unix/Linux-basierten Server-Rechners abstimmen.

Nachteil ist:

- Nicht so große Auswahl an Software für grafische Oberflächen.

Andere Betriebssysteme

Für andere Unix-Derivate gilt im Prinzip das Gleiche wie für Linux. Betriebssysteme wie OS/2 oder BE, deren Zukunft entweder ungewiss ist oder die nur noch für speziellere Einsatzgebiete weiterentwickelt werden, mögen zwar prinzipiell geeignet sein, doch dürfte der Mangel an Software und an Möglichkeiten zum ordentlichen Austesten von Webseiten ein gewisses Hindernis darstellen.

2.1.2 WYSIWYG- und code-basierte Editoren

Webdesigner teilen sich in solche, denen die Kontrolle über alle Quelltexte wichtiger ist, und solche, die Webseiten genauso erstellen möchten wie Präsentationen oder Prospekte: nämlich nach dem Prinzip, beim Entwickeln genau das am Bildschirm zu sehen, was am Ende beim Anwender zu sehen sein wird (WYSIWYG – what you see is what you get).

Aus verschiedenen Gründen raten wir davon ab, auf WYSIWYG-Editoren zu setzen:

- WYSIWYG-Editoren sind komplexe Werkzeuge, die selbst eine hohe Einarbeitungszeit erfordern. In mindestens der gleichen Zeit ist es auch möglich, sich mit dem Bearbeiten auf Code-Ebene vertraut zu machen.
- WYSIWYG ist in Zusammenhang mit Webseiten eine glatte Lüge. Denn was der Designer sieht, ist nicht zwangsläufig das, was jeder Seitenbesucher sieht. Mit unterschiedlichen optischen Hardwarevoraussetzungen wie Bildschirmauflösungen, Farbtiefen, Größe eines Pixels fängt es an und mit unterschiedlichen Farbberechnungsmethoden, unterschiedlichen Browsern und unterschiedlicher Performance geht es weiter.
- WYSIWYG-Editoren erzeugen bis auf wenige Ausnahmen mangelhaften bis grottenschlechten HTML-Code, der zahlreiche veraltete Auszeichnungen und eine Menge sinnloser Aufblähungen enthält. Der so erzeugte Code ist von Hand später kaum noch wartbar, was sich in teuren oder zeitaufwendigen Neuentwicklungen rächen kann.
- Ab dem Augenblick, wo Sie in Zusammenhang mit Webseiten programmieren wollen oder müssen, sei es client-seitig mit JavaScript oder server-seitig mit Sprachen wie PHP, müssen Sie ohnehin auf Code-Ebene arbeiten und benötigen leistungsfähige Texteditoren. Wenn Sie auch HTML und CSS mit solchen Editoren bearbeiten, können Sie problemloser alle Ressourcen mit der gleichen Software bearbeiten.
- Moderne, code-basierte Texteditoren sind keine archaischen Werkzeuge mehr, sondern sie bieten eine Menge Funktionen an, die komfortables und effizientes Arbeiten ermöglichen.

Das code-basierte Arbeiten, immer wieder tot gesagt, hat sich also nicht nur gehalten, sondern ist im professionellen Bereich eindeutig präziser und wirtschaftlicher, weshalb wir jedem Webentwickler, aber auch jedem Webdesigner, dazu raten, sich code-basiertes Arbeiten anzugewöhnen.

Auch das vorliegende Buch werden Sie nur sinnvoll als Arbeits- und Übungsbuch einsetzen können, wenn Sie bereit sind, auf Code-Ebene zu arbeiten. Bevor Sie dabei allerdings zu archaischen Werkzeugen wie dem Windows Notepad Editor greifen, probieren Sie besser ein paar der nachfolgend aufgelisteten Editor-Produkte aus. Lassen Sie sich ruhig etwas Zeit beim Ausprobieren und Vergleichen, denn mit dem Editor, für den Sie sich entscheiden, werden Sie eine Menge Zeit verbringen.

Nachfolgende Editoren eignen sich für alle in diesem Buch behandelten Sprachen, also HTML, CSS, JavaScript, PHP und SQL sowie zum Bearbeiten von XML- oder sonstigen Textdateien oder Konfigurationsdateien. Sie sind also als Universaleditoren einsetzbar.

Links zu code-basierten Editoren für MS Windows

EditPlus (en, Shareware): *http://www.editplus.com/*
HomeSite (en, Shareware): *http://www.macromedia.com/software/homesite/*
SuperHTML (de, Shareware): *http://www.superhtml.de/*
TextPad (en/de, Shareware): *http://www.textpad.com/*
UltraEdit (en/de, Shareware): *http://www.ultraedit.com/*
Weaverslave (de, Freeware): *http://www.weaverslave.ws/*

Links zu code-basierten Editoren für Macintosh

BBEdit (en, kostenpflichtig): *http://www.barebones.com/products/bbedit/*
PageSpinner (en, Shareware): *http://www.optima-system.com/pagespinner/*

Links zu code-basierten Editoren für Linux

Bluefish (en, kostenlos, GTK2): *http://bluefish.openoffice.nl/*
Kate (en, kostenlos, KDE): *http://kate.kde.org/*
Quanta (en, kostenlos, KDE): *http://sourceforge.net/projects/quanta/*

Gute code-basierte Editoren verfügen über folgende typischen Features:

- sprachenabhängiges Syntax-Highlighting,
- integrierbarer Webbrowser oder schnelles Anzeigen aktuell bearbeiteter Seiten im Browser,
- als explorer-artige Leiste zuschaltbarer und dauerhaft anzeigbarer Dateimanager, der zum schnellen Öffnen von Dateien dient oder zum Verlinken oder Referenzieren von anderen Dateien via Drag and Drop,
- HTML- und CSS-gerechte Farbauswahl,
- integrierter Grafikviewer,

- automatische Code-Vervollständigung,
- speicherbare Bausteine für häufig benötigte Code-Konstrukte,
- dateiübergreifendes Suchen und Ersetzen,
- Speicherfähigkeit für mehrere Kodierungsvarianten (Ansi, Unicode, UTF8 usw.),
- Möglichkeit, Projekte zu definieren mit Projektdateien usw.,
- viele Dateien gleichzeitig öffnen und bearbeiten (Multiple Document bzw. tabbed Interface),
- automatische Rechtschreibprüfung,
- Ansichten-Splitting, um Dateien an mehreren Stellen gleichzeitig zu bearbeiten,
- flexibel einstellbarer Zeilenumbruch,
- Möglichkeit zum Aufnehmen und Wiedergeben von Tastatur- und Mausmakros.

2.1.3 Webbrowser

Die meisten Anwender bevorzugen einen bestimmten Webbrowser und benutzen fast nur diesen, um Webseiten aufzurufen. Wer jedoch Webseiten erstellt, muss diverse andere als den persönlich bevorzugten Browser zumindest installieren, mitunter auch in mehreren Versionen. Der Grund ist, dass das Austesten mit verschiedenen Browsern eine wichtige qualitätssichernde Maßnahme beim Erstellen von Webseiten ist.

Es ist jedoch nicht möglich und nicht zumutbar, mit allen nur erdenklichen Browsern auf allen nur erdenklichen Systemen zu testen. Zumindest sollten folgende Browser auf einem Entwicklersystem installiert sein:

- ***MS Internet Explorer*** (*http://www.microsoft.com/windows/ie_intl/de/*, für Macintosh: *http://www.microsoft.com/mac/ie/*), zum Redaktionszeitpunkt dieses Buches in Version 6.0, da diese am verbreitetsten ist.
- ***Firefox*** (*http://www.firefox-browser.de/*), der beliebteste Browser, der auf der Gecko-Engine basiert. Verschiedene andere Browser wie Netscape seit Version 6.0, Mozilla oder K-Meleon basieren ebenfalls auf der Gecko-Engine und sind damit, was die technischen Standards betrifft, sehr ähnlich. Zum Redaktionszeitpunkt dieses Buches war Firefox in Version 1.0 der verbreitetste Standard.
- ***Opera*** (*http://www.opera.com/*), leistungsfähiger Browser, der technisch sehr ausgereift ist und eine überzeugte Fangemeinde hat. Zum Redaktionszeitpunkt dieses Buches war Opera in Version 7.5x aktuell und Version 8.0 in Vorbereitung. Da Opera recht häufig in neuen Versionen erscheint, ist es sinnvoll, sich zumindest mehrere Vollversionen nebeneinander zu installieren.
- ***Lynx*** (*http://lynx.browser.org/*), Textmodus-Browser, der technisch jedoch sehr aktuell und präzise ist. Ideal zum Austesten von Seiten in Hinblick auf die Zugänglichkeit für nicht grafische Ausgabesysteme.

Der Internet Explorer ist für MS Windows und Apple Macintosh verfügbar, die übrigen Browser zusätzlich auch für verschiedene Linux-Oberflächen und teilweise auch für andere Plattformen, jedoch nicht immer in der gleichen Version für alle Plattformen und auch nicht immer mit identischem technischen Funktionsumfang. Vor allem beim MS Internet Explorer weichen die Entwicklungslinien für Macintosh und Windows trotz gleicher Versionsnummern voneinander ab.

Browser-Markt und Standardkonformität

Es ist also beim Erstellen von professionellen Websites unbedingt erforderlich, diese mit verschiedenen Browsern auszutesten. Der Grund ist, dass die Basistechnologien des Web produktunabhängige Standards sind, die von den einzelnen Browsern in ihren verschiedenen Produktversionen unterschiedlich vollständig und korrekt umgesetzt sind.

Daran sind allerdings nicht die Browser-Hersteller alleine schuld. Die Standardisierung der Basissprachen HTML und CSS ist selbst noch nicht endgültig abgeschlossen, sondern ein weiter voranschreitender Prozess. Damit kein völliges Chaos entsteht, gibt es Sprachversionen. Immerhin liegen die für gegenwärtiges Webdesign maßgeblichen Sprachversionen HTML 4.01, XHTML 1.0, CSS 1.0 und 2.0 mittlerweile seit Jahren vor. Gleichzeitig ist das allgemeine Bewusstsein um die Bedeutung dieser Standards geschärft worden. So konnte sich der Softwaremarkt auf die Unterstützung dieser Sprachversionen allmählich einstellen.

Das größte Problem ist derzeit der MS Internet Explorer, der sich dank der Integration ins Windows-Betriebssystem einen fast monopolistischen Marktanteil von über 90% gesichert hat, aber seit Jahren in der Version 6.0 vorliegt und die gesamte Weiterentwicklung verschlafen hat. Verglichen mit Browsern wie Firefox 1.0 oder neueren Opera-Versionen ist der MS Internet Explorer 6.0 nicht nur funktionsarm, umständlich und wegen der Systemintegration stets ein potenzielles Sicherheitsrisiko für den PC, sondern er ist auch technisch leicht veraltet.

Dass trotzdem einige Sites mit diesem Browser am besten (oder schlimmer noch: überhaupt nur mit diesem Browser) funktionieren, liegt daran, dass die Entwickler nie über den Tellerrand des Internet Explorers geschaut haben, die Webstandards nicht berücksichtigen und einfach so lange herumtüfteln, bis ihre Seiten im Windows-Default-Browser so aussehen wie gewünscht. Wer so denkt, untergräbt jedoch alle Standardisierungsbemühungen und huldigt letztlich nur einem Monopolisten. Zwar hat sich dieser seine Stellung selbst erarbeitet, was aber nichts daran ändert, dass jegliche Monokultur schädlich ist.

Immerhin ist der Marktanteil des Internet Explorers seit der Zeit leicht gesunken, als die *SpreadFirefox-Initiative* gegründet wurde (*http://spreadfirefox.com/*). Diese Initiative hat sich zum Ziel gesetzt, auf breiter Anwenderfront darüber aufzuklären, warum es besser ist, auf moderne Browser wie Firefox umzusteigen. In Deutschland sammelten Anwender sogar Gelder, um eine ganzseitige Anzeige für den Firefox-Browser in der Frankfurter Allgemeinen Zeitung FAZ zu schalten (Einzelheiten auf *http://www.firefox-kommt.de/*).

Solche Bemühungen können jedoch auch nichts entscheidend daran ändern, dass die meisten Anwender nur den auf ihrem PC vorinstallierten Browser benutzen. Eine Statistik des niederländischen Webstatistikanbieters *OneStat.com* weist für Februar 2005 folgende Marktanteile aus:

Browser	Markanteil (%)
MS Internet Explorer	87,28 %
Mozilla Firefox	8,45 %
Apple Safari	1,21 %
Netscape	1,11 %
Opera	1,09 %

Tabelle 2.1: Browser-Marktanteile laut OneStat.com im Februar 2005

Kein Thema mehr ist dagegen *Netscape* in der Version 4.x. Noch lange Jahre nach seinem Erscheinen 1997 gab es eine hartnäckige Benutzergemeinde, die den Netscape-Browser in einer 4er-Version einsetzte, zum Entsetzen vieler Webdesigner, da dieser Browser kaum noch mit modernen Standards kompatibel war und jede Menge Extraarbeit erforderte. Diese Verrenkungen nimmt heute glücklicherweise fast niemand mehr auf sich. Die in der obigen Tabelle ausgewiesenen 1,11% für Netscape beziehen sich vor allem auf die neueren Versionen 6.x und 7.x, die wie Firefox und Mozilla auf der Gecko-Engine basieren, also die modernen Standards weitestgehend unterstützen.

2.1.4 Grafik- und Multimedia-Software

Gewöhnliche Webgrafiken sind leider immer noch ausschließlich Pixelgrafiken in den drei gängigen Formaten JPEG, PNG und GIF. Wer jedoch Grafiken selbst entwirft und erstellt, arbeitet zunächst häufig mit vektororientierten Programmen, wie CorelDRAW oder Adobe Illustrator. Im Grunde benötigt ein Grafikprofi also Programme beider Grafiktypen. Für Pixelgrafiken sind sogar meist mehrere Programme im Einsatz, die unterschiedliche Schwerpunkte setzen. Eine professionelle Ausstattung sieht in etwa so aus:

- Ein führendes Vektorgrafikprogramm wie *CorelDRAW* (*http://www.corel.de/*) oder *Adobe Illustrator* (*http://www.adobe.de/products/illustrator/*),
- ein führendes Programm zur Bearbeitung von Pixelgrafiken wie *Adobe Photoshop* (*http://www.adobe.de/products/photoshop/*) oder Paintshop Pro, das mittlerweile von Corel vertrieben wird (*http://www.corel.com/*),
- ein Grafikverwaltungsprogramm wie *ThumbsPlus* (*http://www.thumbsplus.de/*),
- ein schneller Grafikviewer mit einer Menge Konvertierungs- und sonstigen Funktionen wie *Irfan View* (*http://www.irfanview.com/*),
- gegebenenfalls Spezialsoftware für Rendering, spezielle Photoshop-Filter usw.

Eine solche Ausstattung benötigen allerdings nur Designer, die tatsächlich mit dem Entwerfen und Erstellen von Grafiken betreut sind. Für Webdesigner, deren Aufgaben-

schwerpunkte eher das Strukturieren und saubere Realisieren von Sites sind und die Grafiken meist zugeliefert bekommen und allenfalls noch ein wenig anpassen müssen, genügt eine einfachere Ausstattung. Diese kann in erster Linie aus einer älteren Version von Paintshop Pro oder einem anderen ordentlichen Pixelgrafikbearbeitungsprogramm bestehen. Verwaltungsprogramme wie ThumbsPlus oder Viewer und Tools wie IrfanView (Letzteres ist übrigens Freeware) sind allerdings für jeden, der mit Webgrafik zu tun hat, empfehlenswert.

Professionelle Grafiksoftware ist im Gegensatz zu den meisten anderen Softwaretypen, die Sie für eine Entwicklungsumbebung benötigen, richtig teuer. Wenn das Budget jedoch nicht für die neuesten Versionen der Corel Suite oder von Photoshop reicht, dann lohnt es sich, nach etwas älteren Versionen dieser Programme Ausschau zu halten. Oftmals werden diese für einen Bruchteil des Preises der neuesten Version vertrieben.

Flash-Autorensoftware

Der De-facto-Standard für Multimedia im heutigen Web ist Flash. Es handelt sich um ein Dateiformat, das sowohl animierte grafische Abläufe mit oder ohne Ton als auch die Realisierung schicker grafischer Navigationsmenüs sowie interaktiver Spiele oder Lernübungen ermöglicht. Flash verfügt über eine eigene Scriptsprache und ist sehr flexibel einsetzbar. Die fertig kompilierten Flashmovies sind vergleichsweise klein und der Flash-Player, der als Plug-In bei allen modernen Browsern mit dabei ist, benötigt wenig Overhead und ist schnell geladen. Kein Wunder, dass vor allem grafisch und multimedial orientierte Webdesigner so von Flash schwärmen, dass sie am liebsten den ganzen anderen Krempel von HTML bis PHP über Bord schmeißen würden.

Doch es hat gute Gründe, warum Flash nicht zum Standardformat für Webseiten wurde. Die rein grafische und animierte Präsentation ist für die meisten Inhalte schlichtweg ungeeignet. Flash kann auch als Objekt in den HTML-Code von Webseiten kontrolliert integriert werden und in dieser Anwendungsform wird es auch meistens verwendet.

Anders als die übrigen verbreiteten Webtechnologien ist Flash kein offener Standard, sondern ein firmeneigenes Format von Macromedia. Immerhin ist Flash offen dokumentiert und wer Flash-Movies erstellen möchte, ist nicht mehr zwangsläufig auf die zwar führende, doch auch teuerste Autorensoftware angewiesen, die von Macromedia selbst angeboten wird.

Flash-Autorensoftware

Flash MX (Windows, Mac, en, kostenpflichtig): *http://www.macromedia.com/*
Swishmax (Windows, en/de, Shareware): *http://www.swishzone.com/*
Swift 3D (Windows, Mac, en/de, Shareware): *http://www.erain.com/*
FlaX (Windows, en, Shareware): *http://www.flaxfx.com/*
KoolMoves (Windows, en, Shareware): *http://www.koolmoves.com/*

2.1.5 Lokale serverseitige Entwicklungsumgebung

Im Verlauf dieses Buches werden wir genauer beschreiben, wie Sie sich Schritt für Schritt eine lokale serverseitige Entwicklungsumgebung aufbauen können. An dieser Stelle soll nur erläutert werden, welchen Zweck und Vorteil eine solche Entwicklungsumgebung hat.

Das Web basiert auf einer so genannten *Client-Server-Struktur*. Webbrowser sind Clients. Wenn Sie damit eine Website im Internet aufrufen, sendet der Browser eine Datenanforderung für die eingegebene, ausgewählte oder angeklickte Internetadresse aus. Über den Domain Name Server des Internetzugangs und das übrige Domain Name System (DNS) wird der gewünschte Server-Rechner im Netz ausfindig gemacht und die Anfrage an ihn weitergeleitet. Auf Grund des Portsystems weiß der Server-Rechner, dass die ankommende Anfrage für den Webserver gedacht ist. Der Webserver ist einfach ein Programm, das dauerhaft im Speicher des Server-Rechners geladen ist und auf Anfragen wartet. Er kann die vom Browser angeforderten Daten ermitteln und zusammen mit einer Antwort an den wartenden Browser ausliefern. Die Kommunikationsdaten bei Browser-Anfrage und Server-Antwort folgen bestimmten Regeln. Was ein Browser fragen kann und welche Antworten ein Webserver geben kann, regelt das HTTP-Protokoll.

Auf einem Entwickler-PC können Sie diesen gesamten Vorgang als eine Art Mikrokosmos nachbauen. Sie können einen Webserver lokal installieren und anschließend Webseiten auf dem eigenen Rechner über HTTP-Adressen, also beginnend mit *http://...*, aufrufen.

Mit der Installation eines Webservers können Sie aber auch jede Art von serverseitiger Datenverarbeitung am eigenen PC realisieren. So können Sie PHP und MySQL selber installieren und komplette Webanwendungen inklusive Datenbankanbindung lokal am eigenen Rechner realisieren. Damit sparen Sie sich das mühsame Arbeiten auf einem entfernten Server-Rechner. Vor allem, wenn Sie keinen anderen entfernten Server-Rechner als den Produktivrechner haben, also den Rechner, auf dem das aktuell im Web zugängliche Angebot abgelegt ist, sollten Sie die Entwicklungs- und Testumgebung unbedingt lokal am eigenen PC einrichten. Alles andere wäre fahrlässig. Denn beim Entwickeln passieren immer wieder mal Pannen wie Fehler in Scripts, die zum versehentlichen Löschen wichtiger Daten führen, das System lahmlegen oder andere unerfreuliche Auswirkungen haben.

Ein weiterer Vorteil besteht darin, dass Sie, wenn Sie sich eine serverseitige Entwicklungsumgebung lokal einrichten, viel besser verstehen, wie die serverseitigen Komponenten zusammenspielen und was letztlich genau passiert, wenn ein Browser eine Anfrage startet.

Eine serverseitige Entwicklungsumgebung, wie wir sie für die in diesem Buch behandelten Technologien benötigen, besteht aus:

- einem *Apache Webserver*,
- einer Installation der *Scriptsprache PHP*,
- und einer Installation des *Datenbank Management Systems MySQL*.

Alle drei Produkte sind für alle nur erdenklichen Plattformen erhältlich und außerdem OpenSource und kostenlos, sofern keine speziellen Supportverträge abgeschlossen wer-

den. Es ist also kein Problem, sich diese Produkte am eigenen PC für eine Entwicklungsumgebung zu installieren.

Je nach Anwendungsfall müssen gegebenenfalls noch weitere serverseitige Softwareprodukte installiert werden, z. B.:

- ein Mailserver, der vor allem Mails über SMTP versenden kann,
- ein CVS-Repository zur Versionenverwaltung von Software oder Dokumenten bei Bearbeitung durch mehrere Mitarbeiter,
- ein XML-Parser, falls z. B. in einem XML-Format gespeicherte Daten mit XSLT in eine normale, HTML-basierte Webseite transferiert werden sollen.

Doch um solche Erweiterungen brauchen Sie sich nicht kümmern, solange kein Bedarf dafür besteht.

2.1.6 Diverse Tools

Zur perfekten Ausstattung eines Webdesigner- oder Webentwickler-PCs gehören noch ein paar weitere Softwaretypen, die das Arbeiten erleichtern:

- ***Leistungsfähige Dateiverwaltung:*** Dazu gehören Programme, die in verschiedenen Formaten packen und entpacken (z. B. ZIP, TAR.GZ, RAR), Dateien inhaltlich vergleichen oder Dateien kodieren und dekodieren können (z. B. Mime-Format, UUE-Format). Ideal geeignet sind leistungsfähige »Commander« wie für MS Windows der ***Total Commander*** (*http://www.ghisler.com/* – de/en, Shareware) oder für Linux der ***Midnight Commander*** (*http://www.ibiblio.org/mc/* –en, OpenSource). Falls Sie nicht auf eine solche Art der Dateiverwaltung umsteigen möchten, sollten Sie entsprechende Einzelprogramme wie ***WinZip*** (*http://www.winzip.com/*, – de/en, Shareware) installieren.
- ***Linkchecker:*** Vor allem bei umfangreicheren Websites mit zahlreichen externen Links und vielen internen Querverweisen ist es wichtig, regelmäßig zu prüfen, ob alle Links einwandfrei funktionieren. Denn tote Links sind aus Anwendersicht ein Ärgernis. Linkchecker helfen beim Aufspüren solcher Links. Produkte dieser Art sind ***Xenu's Link Sleuth*** (*http://home.snafu.de/tilman/xenulink.html*, en, Windows, Freeware), Linklint (*http://www.mindspring.com/~bowlin/linklint/*, en, plattformunabhängig, Perl-Interpreter erforderlich, OpenSource) oder gleich die aus mehreren Tools bestehenden ***HTML PowerTools*** (*http://www.tali.com/*, en, Windows, Shareware).
- ***Markup-Korrektur und Validierung:*** Wer fehlerfreie, standardkonforme Webseiten erstellt, kommt nicht umhin, die Seiten mit entsprechenden Tools auf Korrektheit hin zu testen. Das geht zwar auch als Webservice mit dem Validator des W3-Konsortiums (*http://validator.w3.org/*), doch für die lokale Ausstattung sind auch Programme wie der ***CSE HTML-Validator*** (*http://www.htmlvalidator.com/*, en, Windows, Shareware) oder der ***Real Validator*** (*http://arealvalidator.com/*, en, Windows, Shareware) sinnvoll. Auch das von HTML-Chefentwickler Dave Raggett stammende Korrektur-Tool ***HTML Tidy*** (*http://tidy.sourceforge.net/*, en, Freeware, diverse Plattformen) ist sehr zu empfehlen.

Welche und wie viele Tools Sie tatsächlich benötigen, ist zweifellos auch eine Frage des individuellen Arbeitsstils. Entscheiden können Sie dies jedoch letztlich nur, indem Sie das eine oder andere Tool einfach mal ausprobieren.

2.2 Usability und Accessibility

Viele Anfänger im Bereich Webdesign begehen den Fehler, ihre persönlichen Vorlieben oder Vorstellungen zu verallgemeinern. Richtig ist jedoch der umgekehrte Weg: Zunächst muss sich, wer erfolgreiche Webseiten erstellen will, in die Lage seiner Anwender versetzen und sich fragen, was diese Anwender praktisch und unpraktisch finden, was sie stört und was nicht und was sie bereit sind zu akzeptieren und was nicht. Am einfachsten ist es dabei, sich das eigene Verhalten als Anwender vor Augen zu führen, wenn man auf einer Website landet und dort z. B. nach bestimmten Informationen sucht, etwas einkaufen will oder nach Gleichgesinnten sucht.

Auch in Anwender mit Handicaps wie einer Sehschwäche oder Körperbehinderungen sollte man sich versuchen hineinzuversetzen. Websites sind öffentliche Orte, ähnlich wie Busse. Und so, wie es Richtlinien für behindertengerechte Busse gibt, existieren mittlerweile auch Richtlinien für »zugängliche« Webseiten. Deutsche Kommunen werden beispielsweise bereits gezwungen, diese Richtlinien bei ihrem Webauftritt zu beachten. Es ist zu erwarten, dass sich die Verpflichtung auch auf andere Anbieter ausdehnt.

Für die beiden Aspekte haben sich die Schlagwörter Usability und Accessability etabliert. Dieser Abschnitt geht auf die genauere Bedeutung dieser beiden Schlagwörter ein.

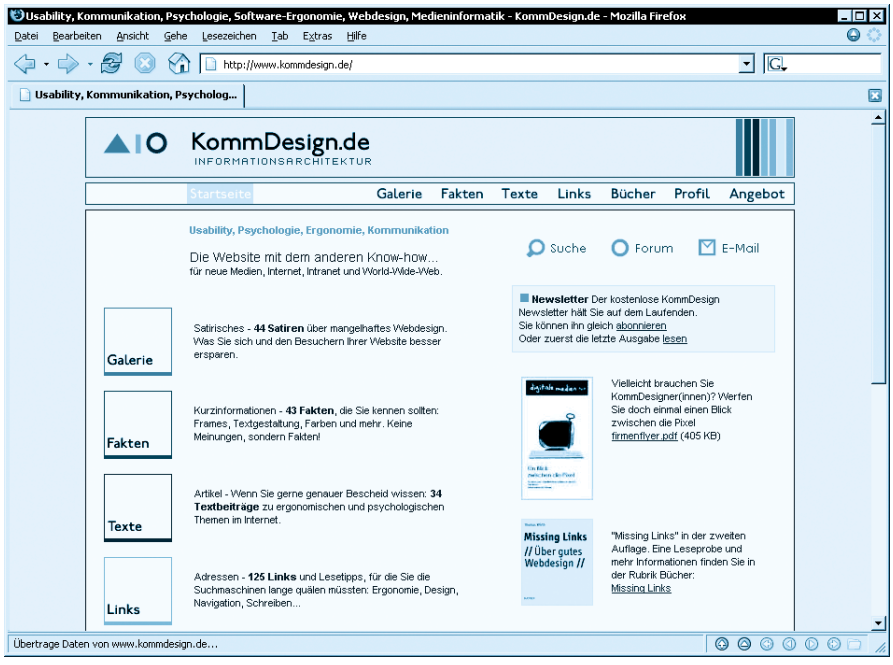


Abbildung 2.1: www.kommdesign.de – alles über Kommunikationsdesign für Webdesigner

2.2.1 Wahrnehmung und Aufmerksamkeit

Jede Webseite wird von Anwendern, die sie aufrufen, wahrgenommen. Das bedeutet, es laufen vorwiegend unterbewusste Prozesse ab, die z. B. bestimmen, was von der Seite überhaupt beachtet wird und was nicht, was zuerst beachtet wird und was erst später oder was spontan als störend oder ablenkend empfunden wird.

Von der Mitte nach links oben

Webseiten erscheinen im Anzeigefenster eines Browsers. Der Blick des Anwenders ist beim Aufbau einer neuen Seite in der Regel zunächst auf die Fenstermitte gerichtet, wandert dann aber, wenn der Inhalt erscheint, nach oben links. Was aus der Fenstermitte her gesehen nordwestlich liegt, wird demnach tendenziell am ehesten wahrgenommen und alles, was südöstlich liegt, am ehesten überhaupt nicht.

Allerdings lässt sich diese Aussage nicht verallgemeinern. So genannte Eye-Catcher wie bewegte Elemente oder alleinstehende Grafiken können dazu führen, dass die Wahrnehmung schnell einer anderen Richtung folgt.

Eye-Catcher-Overkill und Banner-Blindheit

Eye-Catcher sind gut, solange sie einzeln auftreten. Ein wohl platzierter Eye-Catcher lenkt die Aufmerksamkeit in eine vom Anbieter gewünschte Richtung und kann sie von ihrer Default-Nordwest-Route abbringen.

Jede Webseite jedoch, auf der mehrere Eye-Catcher um die sofortige Aufmerksamkeit buhlen, sorgt beim Anwender für eine unterbewusste Panik, die umso stärker ist, je weiter die Eye-Catcher auseinander liegen. Dieser Effekt entsteht vor allem auf Seiten, die voll mit teilweise animierter Werbung sind und wo die Werbeflächen oben, links und rechts zu finden sind. Der eigentliche Inhalt verkommt dabei zur Marginalie.

Bei geübten Anwendern hat man aber noch eine andere Reaktion festgestellt: Der typische, 468x60 Pixel große Werbebanner, der von vielen werbefinanzierten Sites oben mittig platziert ist, wird oftmals gar nicht mehr wahrgenommen, genauer gesagt, unterbewusst automatisch ausgeblendet. Der Effekt wird auch als Banner-Blindheit bezeichnet. Das Phänomen dürfte sich auf alle Werbeflächen ausdehnen, die der Anwender bereits vor Aufruf einer Seite erwartet.

Nähe und Ähnlichkeit

Aus der Gestaltpsychologie stammen zwei wichtige Gesetze: das Gesetz der Nähe und das Gesetz der Ähnlichkeit.

- Das Gesetz der Nähe besagt, dass nahe beieinander liegende Elemente als zusammengehörig empfunden werden, während Elemente, die weiter auseinander liegen, als nicht zusammengehörig empfunden werden.

- Das Gesetz der Ähnlichkeit besagt, dass ähnlich aussehende Elemente als zusammengehörig empfunden werden, während Elemente mit unterschiedlichem Aussehen als nicht zusammengehörig empfunden werden.

Es gibt also zwei Mittel, um zusammengehörige Elemente rein optisch in Zusammenhang zu bringen: Nähe oder gleiches Aussehen.



Abbildung 2.2: Nähe und Ähnlichkeit in einem Webformular

Im Formular der Abbildung wird sowohl durch Nähe als auch durch Ähnlichkeit für Ordnung gesorgt:

- Der Abstand einer Feldbeschriftung zum zugehörigen Eingabefeld ist kleiner als der Abstand zwischen Eingabefeld und Beschriftung des nächsten Feldes (durch das Gesetz der Nähe wird zusammengerückt, was zusammengehört).
- Gleichartige Felder sind gleich lang – im Beispiel die Felder für Vorname und Zuname (durch das Gesetz der Ähnlichkeit erscheinen diese beiden Felder optisch zusammengehöriger als das Feld zur Altersangabe).

Solche Zusammenhänge mögen trivial erscheinen, aber dennoch wird in der Praxis immer wieder dagegen verstoßen.

2.2.2 Kommunikation mit dem Anwender

Ob ein Anwender eine Website »ansprechend« findet, hängt häufig davon ab, wie er auf der Seite angesprochen wird. Stimmen Anrede, Tonfall, benutzte Sprache und erkennbare Absicht nicht, baut der Anwender sehr schnell und oft unterbewusst eine Blockade gegen die Site auf, Diese beherrscht auch den Gesamteindruck und bestimmt, ob der Anwender die Site jemals wieder besucht und was er anderen über die Site erzählt.

Tatsache ist indes, dass man es als Anbieter niemals allen nur erdenklichen Benutzern recht machen kann. Wer hanseatisch seriös daher kommt, erreicht die meisten Jugendlichen nicht. Umgekehrt fliehen viele Erwachsene bei penetrantem Duzen. Der saloppe Schreibstil, der sich an der gesprochenen Sprache orientiert, verschreckt zwar Leser von überregionalen Zeitungen, zieht dagegen ein anderes, mit solchen Medien unvertrautes Publikum gerade an. Das ist nicht anders als in Büchern, Zeitschriften oder bei Rundfunk und Fernsehen.

Kommunikation hat also viel mit der Zielgruppe zu tun. Deshalb ist auch Vorsicht geboten bei der Formulierung allgemeingültiger Regeln. Akzeptiert man die Zielgruppenorientierung, lassen sich jedoch zumindest einige allgemeine Aussagen treffen:

- Wenn eine begrenzbare Zielgruppe existiert und Anrede, Tonfall, benutzte Sprache, Symbolik, Layout oder Art der Präsentation nicht dazu passen, entsteht ein Kommunikationsproblem, das zu Lasten des Anbieters geht.
- Wenn keine begrenzbare Zielgruppe existiert, ist »freundliche Neutralität« die einzig richtige Strategie, um die unterschiedlichen Zielgruppen zumindest nicht zu verschrecken. Freundliche Neutralität wird erreicht durch eine einfache, unauffällige, fehlerfreie Ansprache und durch Verzicht auf eindeutig belegte Symbole. Ebenso wichtig ist jedoch die Freundlichkeit. Sie allein muss in diesem Fall die emotionale Beziehung zum Anwender herstellen.
- Wenn eine begrenzbare Zielgruppe existiert, sollten sich Anrede, Tonfall, benutzte Sprache, Symbolik usw. an der konservativen Grenze bewegen, um mögliche Besucher aus benachbarten Zielgruppen oder neutrale Besucher nicht gleich ganz zu verschrecken.

Kommunikation jenseits der Website

Die meisten Websites bieten ein Formular für Feedback oder allgemeiner für Kontaktaufnahme an. Zumindest aber muss es – das schreibt sogar die Impressumspflicht vor – eine Mailadresse angegeben werden, unter der Besucher den Site-Anbieter erreichen können.

Viele Anbieter nehmen diese Kommunikation nicht ernst, antworten gar nicht oder gereizt auf E-Mails, blenden Kritik innerlich aus und verpassen so die Chance, einen vielleicht unbequemen, aber trotzdem geneigten Anwender zu gewinnen. Denn Anwender, die überhaupt Feedback geben, zeigen bereits dadurch, dass sie die Site ernst nehmen.

Wenn Sie eine Community-Site betreiben oder erleben, dass rund um Ihr Projekt eine Community oder Fangemeinde entsteht, sollte diese gepflegt werden. Dazu gehören die Organisation von Real-Life-Treffen, aber auch Gewinnspiele oder andere Aktionen speziell für die Community.

Verstecken Sie sich nicht hinter Ihrem Webangebot. Die Site ist eine Art öffentlicher Auftritt. Es gibt zwar kein Rampenlicht, doch die Mehrheit der Anwender würde selber nie eine Website unterhalten und betrachtet Sie deshalb wie einen Veranstalter, einen Künstler oder einen Protagonisten. Trauen Sie sich, persönlich zu sein, ohne gleich penetrant privat zu werden.

2.2.3 Usability (Bedienbarkeit) von Websites

Wenn ein Anwender und eine Webseite aufeinander treffen, kommt es meistens zur Kollision zweier Intentionen: Der Anwender sucht etwas oder möchte etwas tun und die Webseite möchte etwas mitteilen oder anbieten. Der Anwender möchte sein Ziel möglichst schnell und bequem erreichen und die Website möchte den Anwender einerseits zufrieden stellen, andererseits möglichst lange halten.

Erfolgreiche Websites geben in diesem Konflikt einfach nach und versuchen nur eines: es dem Anwender so bequem wie nur möglich zu machen. Er soll sein Ziel so schnell wie möglich erreichen, selbst wenn er dadurch doppelt so schnell wieder verschwindet. Er wird dafür aber wiederkommen, denn er weiß es zu schätzen, wenn er wirklich einmal genau das bekommt, was er sich gewünscht hat.

Usability ist Barrierefreiheit

Eigentlich gehört das Schlagwort »Barrierefreiheit« eher in den Zusammenhang, den wir im nächsten Abschnitt in Bezug auf Accessibility ansprechen werden. Doch das Ziel von Usability ist letztlich auch nichts anderes als das Ausräumen möglichst vieler Hindernisse. Usability ist aus Anwendersicht auch das, was durch den Begriff »Flow« ausgedrückt wird: Alles funktioniert wie geschmiert und alles reagiert so wie erwartet. Wer beispielsweise in einem webbasierten Shop einen Artikel kaufen will, erwartet Folgendes:

- Übersichtliche Navigation durch die Produkte
- Suchmöglichkeit
- Aussagekräftige Produktbeschreibungen
- Warenkorbfunktion
- Verschiedene, gängige Bezahlmöglichkeiten wie Kreditkarte, Lastschrifteneinzug, Nachnahme und Rechnung
- Sichere Transaktionen (HTTPS)
- Eindeutige Aussagen zu allgemeinen Geschäftsbedingungen, Lieferzeit, Lieferbedingungen, Garantie, Rückgaberecht usw. vor der Bestellung
- Bestätigung über einen abgeschlossenen Einkauf in Form einer Bestätigungsseite sowie per E-Mail

Werden solche (technisch durchaus anspruchsvollen) Erwartungen nicht erfüllt, entsteht beim Kunden schnell Misstrauen und Käufe werden nicht getätigt.

Für die meisten Anwendungsarten gibt es bereits erfolgreich existierende Vorbilder im Web. Studieren Sie solche Vorbild-Sites genau und orientieren Sie sich daran.

2.2.4 Accessibility (Zugänglichkeit) von Websites

Das W3-Konsortium sorgt nicht nur für die Standardisierung von Sprachen wie HTML und CSS, sondern hat auch eine Initiative für die Zugänglichkeit im Web gegründet: die *Web Accessibility Initiative (WAI)* – *http://www.w3.org/WAI/*. Ziel dieser Initiative ist es, unter Webdesignern das allgemeine Bewusstsein dafür zu schaffen, dass Webseiten auch für Menschen mit eingeschränkten Zugangsfähigkeiten erreichbar sein sollten.

Über das Ausmaß der eingeschränkten Zugangsfähigkeit lässt sich natürlich trefflich streiten. Sehbehinderte oder völlig erblindete Menschen erfahren im Web ganz offensichtlich Einschränkungen. Doch eine eingeschränkte Zugangsfähigkeit kann in einem weiter

gefassten Sinne auch schon Menschen betreffen, die keine DSL-Flatrate und keinen Pentium-4-PC ihr eigen nennen, sondern aus finanziellen Gründen oder wohnortbedingt mit einem alten PC über eine Modemverbindung mit teurem Zeittakt ins Internet gehen.

Menschen mit Sehschwächen erleiden auch andere Einschränkungen als Menschen mit Hörschwächen und wieder andere als Menschen mit motorischen Störungen. Dennoch wird mit gutem Grund versucht, für Websites ähnliche Richtlinien für die Behindertengleichstellung zu schaffen, wie es sie für öffentlichen Orte ebenfalls gibt.

Die Richtlinien der WAI-Initiative sind in dem Dokument »Web Content Accessibility Guidelines« zusammengefasst, das in Fassung 1.0 unter *http://www.w3.org/TR/WCAG10/* zu finden ist. Unter *http://www.w3c.de/Trans/WAI/webinhalt.html* wird eine deutsche Übersetzung angeboten. Zusammengefasst werden in diesem maßgeblichen Dokument folgende Richtlinien aufgestellt:

1. ***Stellen Sie äquivalente Alternativen für Audio- und visuellen Inhalt bereit.*** Der Aufwand, etwa eine audiovisuelle, in Flash realisierte Sprachlernübung alternativ in reiner Textform bereitzustellen, ist sicherlich nicht gering, und Abstriche bei der Wirkung werden sich kaum vermeiden lassen. Für Betroffene erzielt Ihre Site dadurch jedoch einen entscheidenden Mehrwert.

2. ***Verlassen Sie sich nicht auf Farbe allein.*** Das bedeutet vor allem, dass stets auf ausreichenden Kontrastreichtum von Hinter- und Vordergrundfarben geachtet werden sollte. Schwache Kontraste gelten unter manchen Designern als edel, doch wenn sie nur von Menschen mit hundertprozentiger Sehfähigkeit wahrnehmbar sind, wirken sie überheblich und arrogant.

3. ***Verwenden Sie Markup und Stylesheets und tun Sie dies auf korrekte Weise.*** Dies richtet sich vor allem an Designer, die wild verschachtelte Tabellen benutzen, um ihr Seitenlayout zu realisieren, oder die eine Auszeichnung wie die für Blockzitate benutzen, um Absätze einzurücken, bei denen es sich gar nicht um ein Zitat handelt. HTML-Code sollte syntaktisch fehlerfrei und strukturell sinnvoll sein. Für die Optik sollte allein CSS zum Einsatz kommen. Nicht skalierbare Größen sollten vermieden werden.

4. ***Verdeutlichen Sie die Verwendung natürlicher Sprache.*** HTML sieht Möglichkeiten vor, anzugeben, in welcher natürlichen Sprache welcher Text geschrieben ist. Dies erleichtert es akustischen Ausgabesystemen, die Texte korrekt vorzulesen. Auch Abkürzungen sollten als solche mit den dafür vorgesehenen HTML-Elementen ausgezeichnet werden, um akustischen Ausgabesystemen die Arbeit zu erleichtern.

5. ***Erstellen Sie Tabellen, die problemlos transformieren.*** Tabellen, die für nicht-tabellarische Zwecke verwendet werden (z. B. für Seitenlayouts), sollten in linearisierter Form eine sinnvolle inhaltliche Reihenfolge ergeben. »Linearisiert« bedeutet »Zelle für Zelle hintereinander als Absätze gedacht, in der Reihenfolge, wie die Zellen im Code notiert sind«. Ferner gehört die korrekte Auszeichnung von Kopf- und Datenzellen dazu sowie die Verwendung weiterer logischer Tabellenelemente wie Spaltengruppen, Kopf-, Körper- und Fußbereich usw.

6. ***Sorgen Sie dafür, dass Seiten, die neue Technologien verwenden, problemlos transformieren.*** Damit ist gemeint, dass ein HTML-Dokument auch dann gut lesbar ist, wenn das schicke Stylesheet nicht angezeigt werden kann oder wenn JavaScript deaktiviert ist. Der Inhalt sollte auf keinen Fall von zusätzlichen Technologien für Optik oder Dynamik abhängig sein. Falls doch, sollte eine Alternativversion bereitgestellt werden, die auf zusätzliche Technologien verzichtet.

7. ***Sorgen Sie für eine Kontrolle des Benutzers über zeitgesteuerte Änderungen des Inhalts.*** Bewegliche Abläufe, Filme oder Sounds sollten nicht ohne Eingriffsmöglichkeit des Anwenders durchlaufen oder gar in Endlosschleife wiederholt werden. Alles, was in der Zeit abgespielt wird, sollte über Pausen- und/oder Stopp-Funktion verfügen.

8. ***Sorgen Sie für direkte Zugänglichkeit eingebetteter Benutzerschnittstellen.*** Eingebettete Multimedia-Objekte oder Java-Applets sollen entweder selbst den Richtlinien für Zugänglichkeit folgen oder die einbettende Webseite muss dafür sorgen, dass eine solche Zugänglichkeit gegeben ist.

9. ***Wählen Sie ein geräteunabhängiges Design.*** Die Website sollte sowohl über die Tastatur als auch per Maus oder über andere Eingabeformen wie Sprache oder spezielle Zeigegeräte bedienbar sein. Vieles davon nehmen Ihnen bereits Betriebssystemoberfläche und Browser ab. Doch sollten Hyperlinks auf die dafür vorgesehene Weise ausgezeichnet werden. Von Möglichkeiten, wichtige Links oder Schaltflächen mit einem Tastatur-Hotkey zu belegen, sollte Gebrauch gemacht werden.

10. ***Verwenden Sie Interim-Lösungen.*** Damit ist die Rückwärtskompatibilität zu älteren Browsern und anderer Zugangs-Software gemeint. So wird etwa empfohlen, Inhalte nicht ohne Not auf mehrere Fenster zu verteilen.

11. ***Verwenden Sie W3C-Technologien und -Richtlinien.*** Diese Richtlinie besagt, dass die Technologien des W3-Konsortiums beim Erstellen von Webseiten den Vorzug vor anderen Technologien erhalten sollten. Damit wird der Unsitte mancher Webdesigner, eine reine Flash-Lösung zu bevorzugen und eine HTML-Lösung nur noch als belächelte Alternative zur Verfügung zu stellen, eine klare Absage erteilt. Auch PDF-Dokumente sollten nicht anstelle von HTML, CSS usw. angeboten werden, sondern nur als ergänzendes Material oder als Komfortfunktion, etwa um eine optimierte Druckversion eines Artikels anzubieten.

12. ***Stellen Sie Informationen zum Kontext und zur Orientierung bereit.*** Dazu gehört beispielsweise, komplexe Formulare in »Fieldsets« aufzuteilen, also in logische Gruppen. Wichtig ist aber auch, Textwüsten mit ausreichend vielen Zwischenüberschriften zu versehen, um »Haltestellen« einzurichten. Frame-Fenster, Grafiken usw. sollten mit sprechenden Titeln und Alternativtexten versehen werden.

13. ***Stellen Sie klare Navigationsmechanismen bereit.*** HTML stellt, abgesehen vom link-Element, keine richtige Funktionalität zur Realisierung einer Site-Navigation bereit. Deshalb sind Webdesigner angehalten, ein Set von Links durch Position, Beschriftung und andere Merkmale als erkennbare Site-Navigation zu definieren. Die Navigation

sollte site-weit konsistent sein. Linkziele sollten mithilfe der verschiedenen Möglichkeiten, die HTML dazu bereitstellt, möglichst genau beschrieben werden.

14. *Sorgen Sie dafür, dass Dokumente klar und einfach gehalten sind.* Wenn möglich, sollte eine einfache, klare Sprache gewählt werden. Nicht nur Muttersprachler lesen eine Webseite und auch Menschen mit Leseschwierigkeiten gehören zur Benutzergruppe.

2.3 Planung, Realisierung und Pflege

Größere Websites müssen ebenso wie umfangreichere Software- oder Produktentwicklungen in Angriff genommen werden. »Einfach drauf los legen« rächt sich früher oder später, und je später, desto schlimmer.

Hinzu kommt, dass Websites häufig in Auftrag gegeben werden. Sowohl Auftraggeber als auch Auftragnehmer haben ein Interesse daran, dass genau festgehalten wird, was entwickelt werden soll, welcher Zeitraum dafür zur Verfügung steht, welche definierbaren Zwischenstände (Meilensteine) es geben soll, wann diese fertig sein sollten (Roadmap), welche qualitätssichernden Maßnahmen durchgeführt werden (Tests) und wie nach der Erstrealisierung die weitere Betreuung geregelt wird.

2.3.1 Konzeptphase

Diese Phase lässt sich in folgende Abschnitte unterteilen:

1. Kreative Ideensammlung
2. Erstellung eines Lastenhefts (Fachgrobkonzept)
3. Erstellung eines Pflichtenhefts (Fachfeinkonzept)
4. Errichtung und Organisation des Entwicklungsumfelds

Kreative Ideensammlung

In der kreativen Ideenphase sollten Techniken wie Brainstorming oder vergleichbare Kreativitätstechniken zum Einsatz kommen. Dabei dürfen alle Wünsche geäußert werden, die eine zu entwickelnde Website erfüllen sollte, egal wie aufwändig diese zu realisieren wären und aus welcher Ecke sie kommen. Alle Wünsche sind erlaubt, egal in welcher Reihenfolge. Kritik der Sorte »das kostet doch viel zu viel« oder »wer soll das denn machen?« ist tabu und muss notfalls bewusst unterdrückt werden. Es geht zunächst einmal darum, alles zuzulassen, was an Wünschen an eine solche Website herangetragen werden könnte. Das Ergebnis wird auch einen guten Eindruck davon vermitteln, mit welchen Vorstellungen oder Erwartungshaltungen manche Benutzer möglicherweise die Site aufrufen werden.

Alle Ideen sollten notiert werden. In einer Auswertungsphase können die Ideen dann der besseren Übersichtlichkeit halber logisch gruppiert werden.

Lastenheft

Der zweite Schritt besteht darin, aus dem gewonnenen Ideenpool ein realistisches Projekt zu entwickeln. »Realistisch« bedeutet: vereinbar mit dem verfügbaren Budget, mit der vorgegebenen Zeit, mit der vorhandenen Manpower und mit dem verfügbaren Know-how. Ein Lastenheft sollte schriftlich als Dokument fixiert werden und auf jeden Fall folgende Abschnitte enthalten:

- Definition von Zielen und Zielgruppen der geplanten Website
- Definition von erforderlicher Umgebung (Hard- und Softwarevoraussetzungen beim Server-Rechner, Domainnamen usw.)
- Übersicht der Bereiche, welche die Site anbieten soll (z.B. News, Artikelsammlung, Linkverzeichnis, Diskussionsforum)
- Übersicht der geplanten Funktionen (z.B. Volltextsuche, webbasierte Redaktionsschnittstelle, Anwenderregistrierung)
- Definition der Qualitätsanforderungen (z.B. Validität von HTML- und CSS-Code, Browser-Kompatibilitäten, Richtlinien für die Programmierung wie etwa Code-Wiederverwendung, Objektorientierung oder XML-Einsatz sowie Aussagen zum Datensicherungskonzept, zu Server-Ausfällen usw.)

Pflichtenheft

Das Pflichtenheft enthält die konkrete Ausarbeitung der im Lastenheft formulierten Aussagen. Oftmals ist es so, dass das Lastenheft vom Auftraggeber erstellt wird, während die Erstellung des Pflichtenhefts die erste Aufgabe des Auftragnehmers ist.

Bei den allgemeineren Punkten wie Zielen und Zielgruppen sollte im Pflichtenheft unterschieden werden zwischen dem, was auf jeden Fall erreicht werden muss, dem, was »nice to have« ist, und dem, was keinesfalls oder bewusst nicht erreicht werden soll.

Bei der erforderlichen Umgebung kommt es darauf an, wie präzise die Vorgaben im Lastenheft sind. Gegebenenfalls ist im Pflichtenheft festzulegen, welche genaue Hardwareausstattung benötigt wird, welche Softwareprodukte in welchen Versionen einzusetzen sind, bis wann der Produktivserver fertig eingerichtet verfügbar sein muss, ob ein paralleler Testserver benötigt wird usw.

Die Funktionen, über die eine Website verfügen sollte, sollten im Pflichtenheft so genau präzisiert werden, dass die Programmierung darauf aufsetzen kann. Falls mehrere Entwickler an der Website arbeiten, ist es sinnvoll, ergänzend zum Pflichtenheft Funktionsbeschreibungen und Programmablaufdiagramme (Flussdiagramme) nach DIN 66001 zu erstellen. Je umfangreicher das Projekt, desto mehr Aufwand sollte in möglichst präzise Entwicklervorgaben gesteckt werden. Es kann durchaus sein, dass die Entwicklungsbeschreibungen mehr Zeit in Anspruch nehmen als die reine Realisierung.

Vorgaben zur Qualitätssicherung sollten wenn nötig präzisiert werden. Aus einer Forderung wie der nach validem HTML sollte eine genaue Vorgabe werden, mit welcher

HTML-Version in welcher Variante gearbeitet werden sollte, welcher CSS-Standard eingehalten werden sollte usw.

Einrichtung und Organisation des Entwicklungsumfelds

Dazu gehört das Installieren und Konfigurieren benötigter Software am Entwickler-PC und gegebenenfalls am Server-Rechner. Verzeichnisstrukturen müssen angelegt werden, Ressourcen wie vorhandene Grafiken müssen gegebenenfalls webgerecht angepasst und bereitgestellt werden, Script-Dateien können schon mal angelegt und beispielsweise mit einheitlichen Kommentarinformationen am Dateianfang versehen werden. Wenn mehrere oder viele Entwickler an einem Projekt mitarbeiten, muss gegebenenfalls ein Versionenkontrollsystem (Repository) eingerichtet werden, es müssen Verzeichnis- und Dateirechte definiert werden und die beteiligten Entwickler müssen falls notwendig eine Einführung in die einzuhaltenden Konventionen erhalten.

2.3.2 Realisierungsphase

Das Erstellen von Seitenlayouts sollte zunächst von jeglicher Programmierung abgekoppelt werden. Hier genügt es, mit statischen HTML-Dateien zu arbeiten, die später gar nicht mehr in dieser Form benötigt werden. Für jeden geplanten Seitentyp sollte eine HTML-Datei mit realistischen Inhalten (oder notfalls mit realistisch umfangreichem Blindtext) erstellt werden. Zentrale CSS-Definitionen können ruhig schon in einem separaten Stylesheet notiert werden.

Später können die testweise erstellten HTML-Dateien in Templates oder Dokumentvorlagen umfunktioniert werden.

Die Programmierung sollte ihrerseits zunächst von der späteren Website-Optik abgekoppelt in Angriff genommen werden. Professionelle Programmierung entsteht modular, d. h. ein ganzes Ensemble an Scripts, von denen jedes spezielle Aufgaben erfüllt, wird am Ende zusammenarbeiten. Bevor es so weit ist, sollte jedoch jede einzelne Funktionalität wenn irgend möglich separat ausgetestet werden, bevor sie ins Ensemble aufgenommen wird. Dazu sind oft kleine, schnell gebastelte Testseiten erforderlich und Testdaten, die von »üblich« bis »möglichst fies« reichen sollten.

Je zuverlässiger alle Einzelteile funktionieren, desto einfacher ist es, am Ende alles zusammenzufügen. Chaos, Panik und Entwicklerwut entstehen am leichtesten dann, wenn versucht wird, viele Details gleichzeitig zu realisieren, um dann bei auftretenden Fehlern oder Problemen gar nicht zu wissen, wo man mit der Fehlersuche beginnen soll.

Testphase

Zwischen Fertigstellung und offiziellem »Launch« einer Website sollte auf jeden Fall eine ausreichende Testphase eingeplant werden. In dieser Phase sollte sich die Site bereits auf einem Server-Rechner befinden, z. B. unter anderem Domainnamen oder ohne Domainnamen über IP-Adresse zugänglich, oder so geschützt, dass nur autorisierte Benutzer Zugang haben.

Für den Test sollten auch andere Personen als nur die Entwickler gewonnen werden. Wenn möglich, sollten ausgewählte »Echtbenutzer« in die Tests integriert werden. Für Entwickler ist es oft erstaunlich bis erschreckend, wie viele Unstimmigkeiten, Unschönheiten, aber auch kleine und große Programmierfehler bei solchen Tests noch gefunden werden.

2.3.3 Pflegephase

Folgende Aktivitäten sind für die Wartung einer typischen größeren Website erforderlich:

- Inhalte müssen aktuell gehalten werden. Neue Inhalte müssen regelmäßig und rasch eingepflegt werden.
- Von Benutzern beigetragene Inhalte, etwa aus Foren oder Board-Systemen, aus Wikis oder Gästebüchern müssen überwacht und gegebenenfalls gelöscht oder deaktiviert werden.
- Newsticker oder Newsletter müssen regelmäßig erstellt und gegebenenfalls versendet werden.
- Das Benutzerverhalten muss durch Auswertung von auf dem Server installierter Webstatistik-Software analysiert werden. Bei Auffälligkeiten wie z. B. einseitigem Besucherinteresse muss geklärt werden, ob die Website in der realisierten Form ihr Ziel verfehlt.
- Die Site muss wenn nötig promoted und beworben werden.
- Der Server-Rechner muss regelmäßig kontrolliert werden, und zwar in Hinblick auf RAM- und CPU-Auslastung ebenso wie auf freien Festplattenspeicher.
- Anwender-Feedback muss beantwortet werden. Konstruktive Kritik oder Fehlermeldungen sollten archiviert und so schnell wie möglich oder beim nächsten technischen Update berücksichtigt werden.

3 Relevante Quellen

Niemand muss bekanntlich alles wissen, man muss nur wissen, wo es steht. Auch das vorliegende Buch kann nicht alles vermitteln, aber es möchte nicht versäumen, Ihnen zu verraten, wo Sie zur Not alles nachlesen können. Zu allen in diesem Buch behandelten Sprachen und Technologien gibt es Originalliteratur in Form von Spezifikationen, Referenzen oder Dokumentationen, die im Web abrufbar sind und im Zweifelsfall die maßgebliche Quelle darstellen.

Die Lektüre der vorwiegend englischen Fachtexte ist nicht immer einfach, aber die Texte sind mittlerweile besser als ihr Ruf. So bemüht sich das W3-Konsortium, in seinen Spezifikationen zu Sprachen wie HTML, CSS oder zur Programmierschnittstelle DOM eine klare, einfache Sprache zu benutzen, auch mal Beispiele einzufügen und wenn nötig auch etwas ausführlicher zu werden.

Dokumentationen wie die zu PHP profitieren unter anderem davon, dass Anwender eine vorhandene Dokumentationsseite um Kommentare erweitern können. So sind zu vielen PHP-Funktionsbeschreibungen nützliche und erhellende Praxisbeispiele hinzugekommen, was das Verständnis oft erheblich erleichtert.

Einiges an Literatur ist auch schon ins Deutsche übersetzt worden, etwa die wichtigsten Dokumente des W3-Konsortiums oder große Teile der Dokumentation zum Apache Webserver, zu PHP und zu MySQL.

3.1 Recommendations (Empfehlungen) des W3C

Das ***W3-Konsortium (W3C)*** ist eine mitgliederstarke Organisation, die sich in zahlreiche Arbeitsgruppen aufteilt und dabei jede Menge Literatur produziert. Die wichtigsten Dokumente werden in Form so genannter Recommendations (Empfehlungen) herausgegeben. In einer Recommendation wird eine jeweils behandelte Technologie wie etwa HTML oder CSS vollständig und möglichst unmissverständlich beschrieben.

Bis ein Dokument den Status einer Recommendation erlangt, durchläuft es einen mehrstufigen Review-Prozess, der sicherstellen soll, dass das Dokument am Ende möglichst fehlerfrei, eindeutig und vollständig ist. Zwischenstände des Review werden ebenfalls schon veröffentlicht, sind jedoch noch nicht so verbindlich wie eine endgültige Recommendation. Zwischenstufen des Review-Prozesses sind ***Working Drafts*** (Entwürfe), ***Candidate Recommendations*** (Vorschlagskandidaten für eine Recommendation) und ***Proposed Recommendations*** (endgültig als Recommendation vorgeschlagene Fassungen).

Sämtliche Recommendations des W3-Konsortiums sind unter *http://www.w3.org/TR/* zu finden.

Ein anderer wichtiger Einstieg, um sich über die Arbeit des W3-Konsortiums zu informieren, ist der über *http://www.w3.org/Consortium/activities*. Die Aktivitäten-Übersichtsseite beschreibt kurz die einzelnen, vom W3-Konsortium betreuten Technologien und leitet von dort aus über Links weiter zu den jeweiligen Aktivitäten-Bereichen.

Auch eine deutschsprachige Webpräsenz des W3-Konsortiums gibt es mittlerweile, zu finden unter *http://www.w3c.de/*.

Links zu W3C-Recommendations von in diesem Buch behandelten Technologien:

HTML 4.01: *http://www.w3.org/TR/html4/*
XHTML 1.0: *http://www.w3.org/TR/xhtml1/*
CSS 1.0: *http://www.w3.org/TR/CSS1*
CSS 2.1: *http://www.w3.org/TR/CSS21/*
DOM 1.0: *http://www.w3.org/TR/REC-DOM-Level-1/*
DOM 2.0 Core: *http://www.w3.org/TR/DOM-Level-2-Core/*
DOM 2.0 Events: *http://www.w3.org/TR/DOM-Level-2-Events/*
DOM 2.0 Style: *http://www.w3.org/TR/DOM-Level-2-Style/*
DOM 2.0 HTML: *http://www.w3.org/TR/DOM-Level-2-HTML/*
DOM 2.0 Views: *http://www.w3.org/TR/DOM-Level-2-Views/*
DOM 2.0 Traversal/Range: *http://www.w3.org/TR/DOM-Level-2-Traversal-Range*

Deutsche Übersetzungen (nicht normativ, jedoch eine sinnvolle Ergänzung):

HTML 4.01: *http://www.edition-w3c.de/TR/html4*
XHTML 1.0: *http://www.edition-w3c.de/TR/xhtml1*
CSS 2.0: *http://www.edition-w3c.de/TR/REC-CSS2*

Weitere wichtige Recommendations des W3-Konsortiums:

XML 1.0: *http://www.w3.org/TR/xml-c14n*
XSL 1.0: *http://www.w3.org/TR/xsl/*
XSLT 1.0: *http://www.w3.org/TR/xslt*
XML und CSS: *http://www.w3.org/TR/xml-stylesheet*
SMIL 2.0: *http://www.w3.org/TR/SMIL2/*
PNG-Grafikformat: *http://www.w3.org/TR/PNG*
SVG-Grafikformat: *http://www.w3.org/TR/SVG/*
Web Content Accessibility Guidelines: *http://www.w3.org/TR/WAI-WEBCONTENT/*

Weitere wichtige Services des W3-Konsortiums:

HTML-Validator: *http://validator.w3.org/*
CSS-Validator: *http://jigsaw.w3.org/css-validator/*
Amaya Browser und Editor: *http://www.w3.org/Amaya/*

3.2 Originaldokumentationen

Während HTML, CSS und DOM vom W3-Konsortium spezifiziert werden, ist eine weitere wichtige Technologie, JavaScript, eine von Netscape entwickelte und lizenzierte Programmiersprache. Die ursprüngliche Originaldokumentation von Netscape ist, so unglaublich das klingt, einfach aus dem Web entfernt worden. Stattdessen müssen Programmierer momentan auf die als PDF-Dokument vorliegende Spezifikation ECMA-262 zurückgreifen.

Eindeutig besser bestellt ist es um die Dokumentationen zu den hier maßgeblichen Softwareprodukten Apache, PHP und MySQL. Alle drei Anbieter stellen umfangreiches Dokumentationsmaterial bereit.

Zu Linux gibt es eine offizielle Dokumentationsseite von ***linux.org***, über die alle Manpages, wichtige FAQ-Sammlungen (***frequently asked questions*** – häufig gestellte Fragen) und andere Dokumente abrufbar sind. Weiterhin maßgeblich sind bei Linux jedoch auch die distributionsspezifischen Handbücher und Dokumentationen.

Links zu Originaldokumentationen von in diesem Buch behandelten Technologien:

JavaScript: *http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf*

Apache Webserver 1.3: *http://httpd.apache.org/docs/*
Apache Webserver 2.0: *http://httpd.apache.org/docs-2.0/*
Apache Webserver 2.0 deutsch: *http://httpd.apache.org/docs-2.0/de/*

PHP-Dokumentationsübersicht: *http://www.php.net/docs.php*
PHP-Handbuch deutsch: *http://www.php.net/manual/de/*

MySQL-Dokumentationsübersicht: *http://dev.mysql.com/doc/*
Reference Manual: *http://dev.mysql.com/doc/mysql/en/*
Referenzhandbuch deutsch: *http://dev.mysql.com/doc/mysql/de/*

Linux-Dokumentation: *http://www.linux.org/docs/*

3.3 RFCs und Standarddokumente

Weitere in unserem Buch angesprochene Technologien, etwa das HTTP-Protokoll, liegen nicht als W3C-Recommendation vor, sondern als so genannte ***RFCs*** (***Requests for Comments***). RFCs sind die Bibliothek für Internetstandards aller Art, also eine Art Gehirn des Internets und maßgeblich für Internetsoftware ebenso wie für Webentwickler.

RFCs erscheinen mit fortlaufend vergebenen Nummern. Mittlerweile existieren über 4000 RFCs. Ihr Gesamtumfang beläuft sich auf zigtausend Seiten. Es gibt verschiedene Server, die es sich zur Aufgabe gemacht haben, alle RFCs bereitzustellen. Eine gut organisierte und aktuelle Bereitstellung finden Sie unter *http://www.faqs.org/rfcs/*.

Jeder Internetbenutzer kann neue RFCs beitragen. Allerdings findet auch hierbei ein qualitätssicherndes Review-Prozedere statt. Wer einen neuen Standard vorschlagen möchte, etwa für eine neue XML-Sprache oder ein neues Client-Server-Protokoll, der kann eine so genannte ***Internet Draft*** verfassen. In der RFC mit der Nummer 1543 sind die formalen Regeln beschrieben, an die sich eine solche Internet Draft halten muss. Die fertige Internet Draft reicht der Verfasser bei der Organisation ***Internet Engineering Task Force*** (IETF – *http://www.ietf.org/*) ein. In einer dafür vorgesehenen Mailingliste wird die neue Internet Draft verbreitet und diskutiert. Wird der Vorschlag angenommen, so wird das Dokument nach einer Korrekturphase zur RFC erklärt und erhält eine eigene Laufnummer.

Links zu einigen RFCs für in diesem Buch angesprochene Technologien:

HTTP 1.1: *http://www.faqs.org/rfcs/rfc2616.html*
HTTP 1.0: *http://www.faqs.org/rfcs/rfc1945.html*
HTTPS: *http://www.faqs.org/rfcs/rfc2660.html*
TCP-Protokoll: *http://www.faqs.org/rfcs/rfc761.html*
IP-Protokoll: *http://www.faqs.org/rfcs/rfc791.html*
SMTP-Protokoll: *http://www.faqs.org/rfcs/rfc2821.html*
POP3-Protokoll: *http://www.faqs.org/rfcs/rfc1225.html*
URI-Adressierung: *http://www.faqs.org/rfcs/rfc2396.html*
MD5-Algorithmus: *http://www.faqs.org/rfcs/rfc1321.html*

Teil 2 – HTML und CSS

HTML (HyperText Markup Language, zu Deutsch: Hypertext-Auszeichnungssprache) ist die Basissprache für das Strukturieren von Webseiten. CSS (Cascading Style Sheets, zu Deutsch: stufenartige Formatvorlagen) ist die in HTML integrierbare Sprache für das Layouten und Formatieren von Webseiteninhalten.

4 Basiswissen HTML und CSS

Dieses Kapitel soll die beiden grundsätzlichen Sprachen vermitteln, mit denen moderne Webseiten erstellt werden: HTML und CSS. In HTML strukturieren Sie Inhalte durch Elemente für Überschriften, Absätze, Listen, Tabellen und andere. Mithilfe von CSS bestimmen Sie, wie diese Elemente z. B. im Browser-Fenster angeordnet und formatiert werden.

Beide Sprachen sind computerlesbar, aber auch für Menschen genießbar. Sie bestehen aus englischsprachigen Begriffen oder Abkürzungen davon sowie aus einfachen Steuerzeichen, die auf jeder Tastatur zu finden sind.

4.1 HTML und XHTML

Vielleicht haben Sie bereits von XHTML gehört und kennen Aussage wie die, dass man kein HTML mehr verwenden solle, sondern nur noch XHTML. In diesem Zusammenhang sollten Sie ein paar grundsätzliche Hintergründe kennen.

4.1.1 SGML und XML

HTML ist ein wohldefinierter Standard. Die Regeln für korrektes HTML werden jedoch nicht mithilfe natürlicher Sprache, sondern in einer ebenfalls computerlesbaren Meta-Sprache namens SGML (*Structured Generalized Markup Language*) formuliert. Die Computerlesbarkeit ist wichtig, damit z. B. ein Programm eine in HTML geschriebene Webseite gegen die in SGML formulierten HTML-Regeln auf syntaktische Korrektheit testen kann. Diesen Vorgang nennt man Validierung (Gültigkeitsüberprüfung).

SGML ist sehr leistungsfähig, aber auch recht komplex und für viele praktische Anwendungsfälle überdimensioniert. Als praxisnähere Alternative zu SGML hat sich mittlerweile XML (*Extensible Markup Language*) etabliert. XML ist wie SGML eine computerlesbare Meta-Sprache zum Formulieren der Regeln von Markup-Sprachen wie HTML.

XHTML (Version 1.0) ist zunächst einmal nichts anderes als eine Neuformulierung der Sprachregeln von HTML (Version 4.01) in XML. Mit XHTML 1.0 können Sie also nicht mehr und nicht weniger und nichts anderes tun als mit der aktuellen HTML-Version 4.01: nämlich die Inhalte Ihrer Webseiten strukturieren. Es gibt in diesem Stadium von XHTML auch keine wesentlich anderen Sprachbestandteile als in HTML. Da allerdings die Markup-Regeln von XML etwas anders aussehen als die von SGML, gibt es einige syntaktische Abweichungen und Besonderheiten. In ***Abschnitt 4.3.6*** werden wir auf diese Abweichungen und Besonderheiten gegenüber HTML eingehen.

4.1.2 Interpretation von HTML und XHTML

Ein *Parser* leistet die Verarbeitung von Markup-Sprachen entsprechend der Regeln der Markup-Sprache. Jeder Browser, der eine Webseite am Bildschirm anzeigt, muss deren HTML-Quelltext parsen. Aus dem Markup geht hervor, aus welchen Elementen die Webseite besteht und welche Elemente innerhalb welcher anderer Elemente vorkommen. So kann beispielsweise ein einzelnes Wort wie »Mensch« Inhalt einer Kopfzelle einer Tabelle sein, die sich in einem bestimmten Bereich innerhalb des Dateikörpers befindet.

Zum Verarbeiten von HTML verfügt jeder Browser über einen HTML-Parser. Bei XHTML ist die Angelegenheit etwas komplizierter: Wenn das HTML-Dokument vom Browser als HTML akzeptiert wird, verwendet der Browser seinen HTML-Parser. Wird es jedoch als XML-Dokument akzeptiert, so wird der XML-Parser verwendet, sofern der Browser über einen entsprechenden Parser verfügt. In der Darstellung kann dies erhebliche Auswirkungen haben. So stellen einige Browser XML-geparste Dokumente nur als Markup-Strukturbaum dar, also letztlich als eine sauber formatierte Quelltextansicht. Andere bieten XML-geparste Dokumente möglicherweise nur zum Download an und stellen gar nichts dar.

Ob ein Browser bei einem XHTML-Dokument den HTML-Parser oder den XML-Parser verwendet, hängt in erster Linie vom zugewiesenen *Mime-Type* ab. Der Mime-Type legt den Datentyp fest. Jeder Browser hat lokal eine eigene Liste von Mime-Typen gespeichert, mit deren Hilfe er z. B. Dateiendungen und Datentypen zuordnen kann. Beim MS Internet Explorer ist diese Liste eng verzahnt mit den Dateiendungsverknüpfungen des Betriebssystems. Wird eine beliebige Datei im Browser lokal geöffnet, kann dieser an Hand seiner Mime-Type-Liste entscheiden, wie er mit der Datei verfahren soll. Anders sieht es aus, wenn der Browser Daten von einem Webserver anfordert und empfängt. In diesem Fall übermittelt der Webserver Kopfdaten an den Browser, in denen unter anderem eine Angabe zum Mime-Type der übertragenen Daten stehen kann, die dem Browser zur Orientierung dient.

Für HTML-Dokumente lautet der Mime-Type `text/html`. Für XHTML sollte jedoch laut Spezifikation nicht dieser Mime-Type verwendet werden, sondern `application/xhtml+xml`. Alternativ dazu sind auch die Angaben `text/xml` und `application/xml` möglich. Dies führt jedoch je nach Server- und Browser-Einstellungen zu Problemen. Der immer noch am weitesten verbreitete MS Internet Explorer kann mit dem Mime-Type `application/xhtml+xml` sogar bei angepasster Server-Konfiguration nichts anfangen und bietet ein Dokument mit diesem Mime-Type zum Download an. Daher wird in der Praxis doch noch der Mime-Type `text/html` verwendet.

Wenn Sie sichergehen wollen, dass Ihre XHTML-Seiten als Webseiten im Browser erscheinen, dann weisen Sie statischen XHTML-Dateien am besten ebenso wie HTML-Dateien die Dateiendung *.htm* oder *.html* zu.

Links zu weiteren Informationen:

W3C-Übersicht SGML-Ressourcen:
http://www.w3.org/MarkUp/SGML/
W3C-Übersicht zu XML:
http://www.w3.org/XML/
Artikel »SGML, die leise Revolution«:
http://mintert.com/xml/leise-revolution/leise_revolution.html
Wikipedia-Artikel über Parser:
http://de.wikipedia.org/wiki/Parser
RFC 2045 (Mime):
http://www.faqs.org/rfcs/rfc2045.html
RFC 2046 (Mime-Medientypen):
http://www.faqs.org/rfcs/rfc2046.html
Liste der Mime-Typen:
http://www.iana.org/assignments/media-types/

4.1.3 Argumente für XHTML

Bei neu zu erstellenden Webseiten sprechen durchaus einige Gründe dafür, von vorneherein ein XHTML statt »herkömmlichem« HTML zu verwenden. Diese Gründe hängen letztlich alle mit der Integration von XHTML in der XML-Welt zusammen:

- ***Kombination mit anderen XML-Sprachen:*** In einem XHTML-Dokument können Sie beispielsweise direkt mathematisch/naturwissenschaftliche Formeln in MathML oder Vektorgrafiken in SVG notieren. Der Grund ist, dass in XML-basierende Dokumente, zu denen XHTML-Dokumente ja gehören, Daten aus beliebigen anderen XML-basierenden Sprachen eingebettet werden können.
- ***Konvertierung in andere XML-Sprachen oder Ausgabeformate:*** Mithilfe von XSLT, einer standardisierten, XML-basierten Markup-Sprache zum Übertragen von Inhalten aus XML-basierten Sprachen in andere XML-basierten Sprachen oder beliebige andere Ausgabeformate ist XHTML optimal für die Konvertierung in andere Ausgabeformate gerüstet. Dies ist beispielsweise für Inhalte sinnvoll, die nicht nur als Webseite, sondern z. B. auch als Print-Dokument veröffentlicht werden sollen.
- ***Zukunftssicherheit:*** Die Spezifikation zu XHTML wird in jedem Fall weiterentwickelt. So ist XHTML 2.0 bereits in Entwicklung. Eine parallele Weiterentwicklung von herkömmlichem HTML ist dagegen nicht vorgesehen. Zwar ist XHTML 2.0 für die aktuelle Praxis noch irrelevant, doch kann sich dies in wenigen Jahren ändern. Eine Konvertierung von XHTML-1.0-Dokumenten nach XHTML 2.0 ist vermutlich unproblematischer zu bewerkstelligen als eine Konvertierung von herkömmlichem HTML 4.01 auf XHTML 2.0.

Andererseits ist sauberes HTML auch nicht zwangsläufig schlechter konvertierbar als sauberes XHTML. Die Einbettung anderer XML-Formate ist ebenfalls nur schöne Theorie, die in den heutigen Browsern höchstens ansatzweise funktioniert. So betrachtet spricht aus

praktischer Sicht nichts gegen die Verwendung von HTML. Die Quelltexte in diesem Buch verwenden ebenfalls »gewöhnliches« HTML.

4.2 Eine vollständige Webseite mit HTML

Betrachten wir nun den Quelltext einer vollständigen HTML-Datei, um die Funktionsweise von HTML zu verstehen. Der Quelltext lässt sich so wie hier abgedruckt in einem Texteditor eingeben und als Datei mit der Endung .html oder .htm abspeichern. Aus Gründen der Übersichtlichkeit ist der abgedruckte Quelltext mit Einrückungen versehen. Die Einrückungen verdeutlichen die Struktur der HTML-Datei.

4.2.1 Quelltext und Verschachtelungsstruktur

Listing 4.1: Eine vollständige HTML-Datei

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;
            charset=ISO-8859-1">
      <title>Gemeinden mit A...</title>
   </head>
   <body>
      <div id="Kopf">
         <h1>Gemeinden mit A...</h1>
      </div>
      <div id="Inhalt">
         <p>Die Links führen zu den offiziellen Internet-
         Präsenzen der jeweiligen Gemeinden!</p>
         <ul>
            <li><a href="http://www.aachen.de/">Aachen</a> -
            Regierungsbezirk Köln</li>
            <li><a href="http://www.aalen.de/">Aalen</a> -
            Regierungsbezirk Stuttgart</li>
            <li><a href="http://www.achern.de/">Achern</a> -
            Regierungsbezirk Freiburg</li>
            <li><a href="http://www.ahaus.de/">Ahaus</a> -
            Regierungsbezirk Münster</li>
            <li><a href="http://www.ahlen.de/">Ahlen</a> -
            Regierungsbezirk Münster</li>
         </ul>
      </div>
   </body>
</html>
```

Markup-Sprachen wie HTML sind für die Abbildung von verschachtelten Datenstrukturen ausgelegt. Der Quelltext ist am leichtesten nachvollziehbar, wenn er als Schachtelstruktur verstanden wird. Die äußerste Schachtelebene stellt das html-Element dar,

markiert durch den Anfang mit der Zeichenfolge <html> und das Ende mit </html>. Die Dokumenttypangabe vor <html> lassen wir an dieser Stelle außen vor. Das html-Element schließt zwei bedeutende Elemente ein. Das erste dieser Elemente ist das head-Element, markiert durch den Anfang <head> und das Ende </head>. Das head-Element ist für HTML-Kopfdaten gedacht, wie zum Beispiel den Titel des Dokuments, markiert durch den Anfang <title> und das Ende </title>. Das zweite Element innerhalb des html-Elements ist das body-Element für die im Browser sichtbaren Inhalte, markiert durch <body> bzw. </body>.

Innerhalb des body-Elements enthält das Beispiel-Listing weitere Elemente, auf die wir an dieser Stelle nicht weiter eingehen. Zu erkennen ist jedoch, dass die Verschachtelung der Elemente den gleichen Gesetzen folgt: Elemente beginnen und enden innerhalb anderer Elemente.

4.2.2 HTML-Darstellung im Browser

Wie eine HTML-Datei im Browser dargestellt wird, das hängt von den Einstellungen des Browsers und von Umgebungseinstellungen ab.



Abbildung 4.1: HTML-Datei in einem modernen grafischen Webbrowser

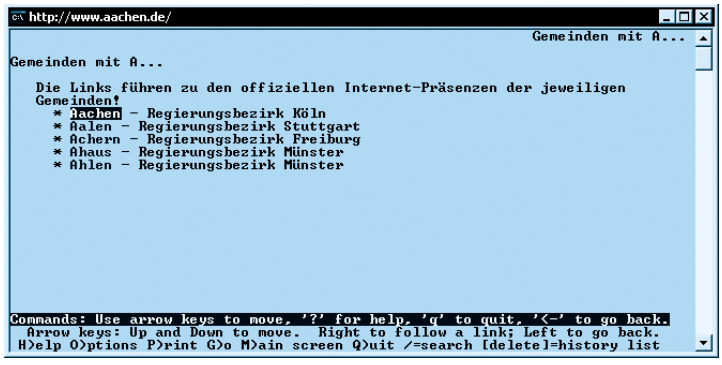


Abbildung 4.2: HTML-Datei in einem textbasierten Browser (Lynx)

Das Listing enthält keinerlei Anweisungen, in welcher Weise die Daten darzustellen sind. Webbrowser verwenden umgebungsabhängig realisierbare Defaultwerte für Textattribute, Abstände usw., um die HTML-Elemente im Browser-Fenster anzuzeigen. Je nach Browser kann das, was dabei herauskommt, sehr unterschiedlich sein, wie die beiden Abbildungen zeigen.

Der Grund ist, dass in HTML nur die Bedeutung von Daten markiert wird, nicht deren Darstellung – man spricht auch von »semantischem Markup«. HTML-Authoring ist also keine gestalterische, sondern eine architektonische Arbeit. Die Sprache ist bewusst so konzipiert, dass sie »geräteunabhängig« ist. Ausgabegeräte können nicht nur grafische Browser sein, sondern z. B. auch rein akustische Reader, Mini-Displays von PDAs oder Handys oder auch eine Braille-Leiste.

Befreien Sie sich also von vorneherein von der Vorstellung, dass Sie mit HTML das Aussehen von Webseiten optisch gestalten können. Gewöhnen Sie sich gleich an, diese Aufgabe allein der Ergänzungssprache CSS zu überlassen.

4.2.3 Fehlertoleranz bei Browsern

Während Compiler oder Interpreter ein Programm oder Script sofort abbrechen oder gar nicht erst ausführen, sobald darin ein Syntaxfehler vorkommt, gehen Browser mit Syntaxfehlern in HTML-Dokumenten meist sehr gnädig und tolerant um. Dies hat bedauerlicherweise dazu geführt, dass zahllose Webseiten, nicht selten auch diejenigen bekannter Sites und großer Firmen, syntaktisch voller Fehler sind. Dazu kommt, dass HTML bereits eine wechselvolle Geschichte hinter sich hat. Die Ausrichtung auf »semantisches Markup« ist noch nicht so alt wie die Sprache selbst. Viele Webdesigner der ersten Generation, die heute oft als besonders erfahren gelten, produzieren ein völlig veraltetes und nicht mehr standardkonformes HTML.

An dieser Stelle sei daher gewarnt vor der Laxheit der Browser im Umgang mit HTML. Die Tendenz der Weiterentwicklung geht auf jeden Fall in die Richtung, Syntaxfehlern gegenüber intoleranter zu sein. Wenn ein Browser beispielsweise eine XHTML-Datei mit seinem XML-Parser verarbeitet und dabei auf Syntaxfehler stößt, wird die Verarbeitung abgebrochen. Es lohnt sich also, sorgfältig auf syntaktisch korrektes HTML zu achten!

4.3 Allgemeine Regeln bei HTML

Bevor Sie die HTML-Elemente im Einzelnen kennen lernen, sollten Sie mit den allgemeinen Regeln von HTML vertraut sein.

4.3.1 Elemente und Tags

Ein HTML-Element besteht in den meisten Fällen aus einer Anfangsmarkierung und einer Endmarkierung, den so genannten *Tags*. So wird beispielsweise der Dokumenttitel durch das Anfangs-Tag `<title>` und das End-Tag `</title>` markiert. Anfangs- und End-Tags

werden in spitze Klammern eingeschlossen. End-Tags sind jederzeit daran zu erkennen, dass unmittelbar hinter der öffnenden spitzen Klammer ein Schrägstrich folgt.

Es gibt auch Elemente, die nur aus einem Anfangs-Tag bestehen. Dies sind Elemente, die keinen weiteren Inhalt haben können, beispielsweise `<br>`, was einen Zeilenumbruch an der notierten Stelle im Text erzwingt.

HTML unterscheidet nicht zwischen Groß- und Kleinschreibung, weshalb Schreibweisen wie `<title>`, `<Title>` oder `<TITLE>` gleichermaßen korrekt sind. Aus heutiger Sicht empfiehlt es sich jedoch, alles klein zu schreiben, und zwar in Hinblick auf eine mögliche spätere Umstellung auf XHTML. In XHTML müssen nämlich alle HTML-Auszeichnungen klein geschrieben werden. Da die heute üblichen Quelltext-Editoren meist Syntax-Highlighting für HTML unterstützen, entfällt auch das Argument, die Großschreibung von Element- oder Attributnamen erhöhe die Quelltextlesbarkeit.

Elementverschachtelung

Wie bereits ersichtlich wurde, entsteht die Strukturierung von Inhalten in HTML durch Verschachtelung von Elementen. Die meisten Elemente mit Anfangs- und End-Tag können also andere Elemente enthalten, was allerdings nicht automatisch bedeutet, dass beliebige Elemente beliebige andere Elemente enthalten können. Die HTML-Spezifikation legt genau fest, welche Elemente innerhalb welcher anderer Elemente vorkommen dürfen und welche nicht. So kann beispielsweise eine Tabellenzelle so ziemlich alles enthalten, von Überschriften über Grafikreferenzen bis hin zu kompletten inneren Tabellen. Dagegen kann eine Überschrift keine Textabsätze oder Tabellen enthalten.

Die Grundregel beim Verschachteln lautet, dass innere Elemente auch tatsächlich innerhalb des Elternelements geschlossen werden müssen.

Falsch ist also ein derartiges Konstrukt:

```
<p>Er war in sie <strong>verliebt.</p>
<p>Sie</strong> war ebenfalls verliebt, aber nicht in ihn.</p>
```

Richtig ist dagegen dieses:

```
<p>Er war in sie <strong>verliebt</strong>.</p>
<p><strong>Sie</strong> war ebenfalls verliebt, aber nicht in ihn.</p>
```

Viele Fehler beim Notieren von HTML entstehen dadurch, dass die Regeln verletzt werden in Bezug darauf, welche Elemente innerhalb welcher anderer Elemente vorkommen dürfen. Im Zweifelsfall benötigen Sie eine Referenz, der eindeutig zu entnehmen ist, ob ein Element innerhalb eines anderen Elements erlaubt ist oder nicht. Im Anhang dieses Buches finden Sie eine entsprechende Element-Referenz zu HTML. Zu empfehlen ist hierfür jedoch auch der Blick in die Original-HTML-Spezifikation des W3-Konsortiums.

Block- und Inline-Elemente

Bei den Elementen, die im sichtbaren Dateikörper, also zwischen `<body>` und `</body>` notiert werden können, wird zwischen Block-Elementen und Inline-Elementen unterschieden. Block-Elemente erzeugen per definitionem eine neue Zeile im Textfluss. Inline-Elemente tun dies nicht. So gehören beispielsweise `<h1>` (Überschrift 1. Ordnung) oder `<div>` (Bereich) zu den Block-Elementen, während `<strong>` (betont) oder `<cite>` (Zitat) zu den Inline-Elementen gehören.

Wie groß bei Block-Elementen die Abstände zum vorhergehenden und zum nachfolgenden Element sind, ist in HTML nicht festgelegt. Auch hierfür verwenden die Browser Default-Werte. Kontrollierbar werden solche Abstände erst durch entsprechende Formatierung mit CSS.

Die Unterscheidung nach Block- und Inline-Elementen ist für die Formulierung allgemeiner Verschachtelungsregelungen recht praktisch. So gibt es Block-Elemente, die keine anderen Block-Elemente, sondern nur Inline-Elemente enthalten dürfen (z.B. `<h1>`), aber auch Block-Elemente, die sehr wohl andere Block-Elemente enthalten dürfen (z.B. `<div>`). Doch auch im Hinblick auf CSS ist die Unterscheidung von Block- und Inline-Elementen von Bedeutung. Näheres dazu erläutern wir später in Zusammenhang mit dem Boxmodell von CSS.

4.3.2 Attribute und Attributwerte

HTML-Elemente können Attribute mit variablen Wertzuweisungen enthalten. Solche Attribute werden im Anfangs-Tag eines Elements notiert.

Ein Beispiel:

```
<p title="Ich bin ein p-Element" lang="de">Fahr mit der Maus über mich, und ich
erkläre dir, was ich bin</p>
```

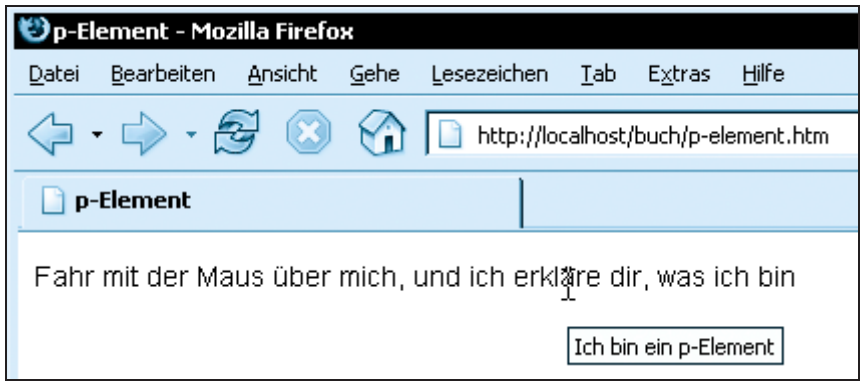


Abbildung 4.3: Tooltipp-Fenster durch title-Attribut

Die meisten Tags können auch mehrere Attribute enthalten, wie das Beispiel zeigt. Alle Attribute werden im Anfangs-Tag notiert, durch Leerraum vom Elementnamen getrennt und vor der schließenden spitzen Klammer. Attributname und Wertzuweisung werden durch ein =-Zeichen getrennt. Wertzuweisungen stehen immer in Anführungszeichen. Erlaubt sind einfache und doppelte Anführungszeichen, doppelte sind jedoch üblicher. Typografische Anführungszeichen sind nicht erlaubt, sondern nur diejenigen, die auf deutsch belegten Tastaturen beispielsweise über die Tastenkombination ⇧+2 erreichbar sind.

Welche Attribute in welchen Elementen möglich sind, legt die HTML-Spezifikation fest. Neben diversen elementspezifischen Attributen gibt es in HTML auch eine Reihe von universell verwendbaren Attributen. Die beiden Attribute im obigen Beispiel, `title=` (Tooltipp-Inhalt) und `lang=` (Textsprache), gehören zu diesen universellen Attributen.

In der HTML-Spezifikation ist nicht nur festgelegt, welche Elemente welche Attribute enthalten dürfen, sondern auch, wie die Wertzuweisung an ein Attribut beschaffen sein muss. So gibt es Attribute, die nur bestimmte Werte zulassen, beispielsweise ein Attribut namens dir=, welches die Schriftrichtung anzeigt. Erlaubt sind hierbei nur die Wertzuweisungen `dir="rtl"` (von rechts nach links) oder `dir="ltr"` (von links nach rechts). Andere Attribute, wie das `title`-Attribut, lassen dagegen relativ freie Angaben zu. Wieder andere Attribute erwarten eine frei wählbare Angabe, die jedoch einem bestimmten Wertebereich oder einem bestimmten Schema entsprechen muss. So erwartet beispielsweise das `lang`-Attribut, welches die Sprache des Elementinhalts angibt, die Angabe eines Sprachenkürzels nach den Listen in RFC 1766 oder RFC 3066. Erlaubt sind also Angaben wie `lang="de"` (Deutsch), `lang="de-at"` (österreichisches Deutsch) oder `lang="en-us"` (amerikanisches Englisch), nicht aber Angaben wie `lang="german"`.

4.3.3 Notation von Zeichen

Webseiten werden in allen Sprachen dieser Welt erstellt und gelesen. Die Element- und Attributnamen von HTML sind dabei immer die gleichen. Doch Elementinhalte oder Wertzuweisungen an Attribute können auch kyrillische, arabische oder chinesische Schriftzeichen erfordern. Der Zeichenvorrat von HTML basiert daher auf dem des ***Unicode-Systems***, das alle bekannten Schriftzeichen, Ziffern, Symbole usw. verzeichnet. Das Unicode-System ist in der internationalen Norm ISO 10646 festgeschrieben.

Wir werden an dieser Stelle nicht auf die ganze theoretische Komplexität von Zeichenvorräten, Zeichenkodierungen, Zeichenklassen, Zeichensätzen usw. eingehen. Stattdessen werden wir an der Praxis orientierte Regeln und Vorgehensweisen vorstellen. Wichtig ist jedoch, dass Sie diese beherzigen. Denn andernfalls können Sie böse Überraschungen erleben, weil beispielsweise die Umlaute und diverse andere Zeichen in Ihren Texten auf Nachbars Macintosh-Rechner oder im Internetcafé von Trinidad völlig verhunzt dargestellt werden.

HTML-eigene Zeichen

Wenn in Ihren Texten Zeichen vorkommen sollen, die von HTML- oder XML-Parsern als Indikatoren für Markup erkannt werden, müssen Sie die entsprechenden Zeichen in jedem Fall »maskieren«. Benutzen Sie dazu am besten die dafür vorgesehenen *HTML-Entities* wie in der nachfolgenden Tabelle gezeigt.

Zeichen	Notation	Bedeutung
<	`<`	`lt` steht für »less than«, also »kleiner als«.
>	`>`	`gt` steht für »greater than«, also »größer als«.
“	`"`	`quot` steht für »quotation mark«, also »Anführungszeichen«.
’	`'`	`apos` steht für »apostroph«.
&	`&`	`amp` steht für »ampersand«, also »kaufmännisches Und«.

Tabelle 4.1: Notation HTML-eigener Zeichen

Einige Beispiele:

```
<p>In HTML notiert sieht dieser Satz so aus: &lt;p&gt;In HTML notiert sieht
dieser Satz so aus:&lt;/p&gt;</p>
<p>Und das Baby sprach &quot;Hicks&quot;</p>
<p>Die Firma &apos;Meier &amp; Söhne&apos; existiert bereits seit 1865 und
produziert seitdem &lt;spitze Klammern&gt;.</p>
```

Achten Sie bei der Notation stets auf die korrekte Zeichenfolge, bestehend aus führendem `&`, gefolgt vom Namen der HTML-Entity und abgeschlossen durch ein Semikolon `;`. Webbrowser wandeln die Notationen bei der Anzeige in die gewünschten Zeichen um.

Viele HTML-Editoren oder Texteditoren mit HTML-Features unterstützen das automatische Einfügen solcher Notationen beim Eintippen der entsprechenden Zeichen oder erlauben zumindest ein bequemes Suchen und Ersetzen, um alle betroffenen Zeichen mit einem Befehl in ihre entsprechenden HTML-Notationen umzuwandeln.

Webseiten vorwiegend in Deutsch und/oder anderen westlichen Sprachen

Wenn die Inhalte Ihrer Webseiten vorwiegend deutschsprachig sind oder auch englisch, französisch, spanisch, portugiesisch, italienisch, schwedisch, dänisch, norwegisch oder finnisch oder gemischt in zwei oder mehreren dieser Sprachen, dann ist die nachfolgend beschriebene Vorgehensweise am einfachsten:

1. Verwenden Sie einen Text- oder HTML-Editor, der Ihnen beim Speichern explizit die Option anbietet, im Zeichensatz ISO-8859-1, ISO-8859-15 oder zumindest im »Default-Zeichensatz« zu speichern, wobei bei Letzterem maßgeblich ist, in welcher Betriebssystemumgebung und mit welcher Benutzersprache Sie arbeiten. Wenn Sie beispielsweise ein deutschsprachiges MS Windows einsetzen, werden Textdateien per Default in einem Zeichensatz gespeichert, der die Zeichensätze ISO-8859-1 bzw. ISO-8859-15 abdeckt.

2. Notieren Sie in allen HTML-Dokumenten im Kopfbereich, also zwischen <head> und </head>, folgende Angabe:

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

   oder:

```
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-15">
```

3. Notieren Sie im Text nur Zeichen aus dem ASCII-Zeichensatz und dem Wertebereich dezimal 32 bis 127 (siehe weiter unten) sowie aus dem Zeichensatz ISO-8859-1 oder ISO 8859-15 und dem Wertebereich dezimal 160 bis 255 (siehe weiter unten).
4. Notieren Sie für alle Zeichen, die nicht in diesem Zeichenvorrat vorkommen, die numerische Schreibweise oder, sofern es eine benannte HTML-Entity dafür gibt, die entsprechende Zeichenfolge. Mehr zu diesen Notationsmöglichkeiten finden Sie weiter unten im Abschnitt »Die Lösung für Sonderzeichen: HTML-Entities und numerische Notation«.

Die folgenden beiden Tabellen enthalten die Zeichen, die Sie bei der zuvor beschriebenen Vorgehensweise direkt, also ohne numerische Umschreibung oder Entity-Notation, verwenden können.

+	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~			

Tabelle 4.2: ASCII-Zeichensatz

+	0	1	2	3	4	5	6	7	8	9
160		¡	¢	£	¤	¥	¦	§	¨	©
170	ª	«	¬	-	®	¯	°	±	²	³
180	´	µ	¶	·	¸	¹	º	»	¼	½
190	¾	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç
200	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ

Tabelle 4.3: Zeichensatz ISO-8859-1

+	0	1	2	3	4	5	6	7	8	9
210	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û
220	Ü	Ý	Þ	ß	à	á	â	ã	ä	å
230	æ	ç	è	é	ê	ë	ì	í	î	Ï
240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù
250	ú	û	ü	ý	þ	ÿ				

Tabelle 4.3: Zeichensatz ISO-8859-1 (Fortsetzung)

Den dezimalen Zeichenwert eines Zeichens ermitteln Sie aus diesen Tabellen, indem Sie die Werte aus Spalten- und Zeilenüberschrift des Zeichens addieren. Ein großes K hat in der Tabelle zum ASCII-Zeichensatz beispielsweise den Zeilenwert 70 und den Spaltenwert 5. Daraus ergibt sich der Wert 75 für dieses Zeichen.

Zeichen aus diesen Tabellen, die Sie nicht auf der Tastatur finden, können Sie entweder über Softwaretools (unter MS Windows z. B. über die »Zeichentabelle«) einfügen oder indem Sie die ALT-Taste gedrückt halten und dabei auf dem separaten Nummernblock der Tastatur eine 0 und anschließend den dezimalen Zeichenwert des gewünschten Zeichens eingeben, also z. B. ALT und 0 2 3 4 zum Erzeugen des Zeichens ê.

Weil der Zeichensatz ISO-8859-1 das mittlerweile so häufig benötigte Euro-Symbol vermissen lässt und in der Praxis auch einige andere Schwächen aufweist, wurde der neuere Standard ISO-8859-15 geschaffen. Die nachfolgende Tabelle listet die Unterschiede zwischen ISO-8859-1 und ISO-8859-15 auf.

Zeichenwert	ISO-8859-1	ISO-8859-15
164	¤	€
166	¦	Š
168	¨	š
180	´	Ž
184	¸	ž
188	¼	Œ
189	½	œ
190	¾	Ÿ

Tabelle 4.4: Unterschiede zwischen ISO-8859-1 und ISO-8859-15

Beachten Sie aber Folgendes: Wenn Sie den Zeichensatz ISO-8859-15 in einem <meta>-Tag explizit angeben, wird beim Editieren auch tatsächlich dieser Zeichensatz unterstützt. Tippen Sie beispielsweise unter MS Windows zum Test bei gedrückter ALT-Taste auf dem separaten Nummernblock der Tastatur die Ziffern 0 1 6 4. Wenn Sie dadurch das Zeichen ¤ erzeugen, wird bei Ihnen ISO-8859-1 unterstützt. Wenn das Euro-Zeichen € erzeugt wird, wird dagegen ISO-8859-15 unterstützt.

Falls Sie ISO-8859-1 verwenden und das wichtige Euro-Zeichen in HTML benötigen, geben Sie im Editor einfach die dafür vorgesehene HTML-Entity ein: €.

Webseiten in bestimmten Sprachen mit anderem Alphabet

Wenn Sie komplette Webseiten etwa in kyrillischer oder arabischer Schrift erstellen möchten, empfiehlt sich dringend eine entsprechende Editierumgebung. Richten Sie also Ihr Betriebssystem und Ihre Software so ein, dass die entsprechenden Zeichensätze unterstützt werden, und verwenden Sie eine geeignete Tastatur.

Ansonsten ist die Vorgehensweise ganz ähnlich zu der beim Arbeiten mit den Zeichensätzen ISO-8859-1 oder ISO-8859-15. Speichern Sie statische HTML-Dateien im entsprechenden Zeichensatz ab und geben Sie den korrekten Zeichensatz im `<meta>`-Tag ein.

Die nachfolgende Tabelle listet geeignete Angaben für einige wichtige Sprachen bzw. Schriftkulturen auf.

Sprache/Schrift	Geeignete Meta-Angabe in HTML
Arabisch	`<meta http-equiv="content-type" content="text/html; charset=ISO-8859-6">`
Griechisch	`<meta http-equiv="content-type" content="text/html; charset=ISO-8859-7">`
Hebräisch	`<meta http-equiv="content-type" content="text/html; charset=ISO-8859-8">`
Japanisch	`<meta http-equiv="content-type" content="text/html; charset=Shift_JIS">`
Koreanisch	`<meta http-equiv="content-type" content="text/html; charset=iso-2022-kr">`
Kyrillisch	`<meta http-equiv="content-type" content="text/html; charset= ISO-8859-5">`
Thai	`<meta http-equiv="content-type" content="text/html; charset= ISO-8859-11">`
Türkisch	`<meta http-equiv="content-type" content="text/html; charset= ISO-8859-9">`

Tabelle 4.5: Meta-Angaben für diverse Sprachen bzw. Schriftkulturen

Die moderne Universallösung: Webseiten mit UTF-8-Kodierung

In den beiden vorhergehenden Unterabschnitten wurden zeichensatzorientierte Lösungen vorgestellt. Mittlerweile entscheiden sich aber viele Entwickler von Webseiten für eine zeichensatzunabhängige Lösung. Dabei ist es erlaubt, im Editor beliebige Zeichen aus dem gesamten Zeichenvorrat des Unicode-Systems direkt einzugeben. Voraussetzung ist allerdings, dass der Editor und gegebenenfalls auch die verwendete Betriebssystemumgebung das Unicode-System intern und auf Schriftebene unterstützen.

Wenn Sie diese moderne Universallösung bei Ihren Webseiten verwenden möchten, gehen Sie wie folgt vor:

1. Verwenden Sie einen Text- oder HTML-Editor, der Ihnen beim Speichern explizit die Option anbietet, Dateien im Kodierungsformat UTF-8 zu speichern.
2. Notieren Sie in allen HTML-Dokumenten folgende Angabe:

```
<meta http-equiv="content-type" content="text/html;
      charset=UTF-8">
```

Nun können Sie mitten im Text Zeichen aller Art eintippen und beliebig mischen. Um jedoch etwa arabische oder japanische Schriftzeichen verwenden zu können, muss auf dem

System eine Schriftart vorhanden und im Texteditor eingestellt sein, die das Unicode-System zu größeren Teilen oder vollständig abdeckt und solche Schriftzeichen abbilden kann.

UTF steht übrigens für *USC Transformation Formats* (»USC-Umwandlungsformate«) und USC wiederum für *Universal Character Set* (»universeller Zeichensatz«). Das Prinzip besteht darin, dass alle Zeichen, die zum klassischen ASCII-Zeichensatz gehören, mit einem Byte Breite abgespeichert werden, während alle übrigen Zeichen intern mit zwei oder mehr Bytes abgespeichert werden. Ein ausgeklügelter Algorithmus sorgt dafür, dass die Bytefolgen beim Lesen solcher Dateien wieder korrekt in Zeichenwerte umgesetzt werden. Gerade diese Tatsache ist jedoch auch dafür verantwortlich, dass falsch interpretierte UTF-8-Dateien zu sehr hässlichen Darstellungen führen können. Das ist auch dann der Fall, wenn Sie im `<meta>`-Tag UTF-8 angeben, die Datei aber gar nicht als UTF-8 abspeichern.

Die Lösung für Sonderzeichen: HTML-Entities und numerische Notation

Wenn Sie mit einer zeichensatzorientierten Lösung arbeiten, brauchen Sie in Ihren Texten dennoch nicht auf beliebige Zeichen aus dem Unicode-Zeichenvorrat zu verzichten. Das Gleiche gilt auch für den Fall, dass Sie die UTF-8-Kodierung verwenden, jedoch ein benötigtes Zeichen nicht direkt eintippen können. HTML erlaubt für diese Fälle verschiedene Notationsmöglichkeiten, um beliebige Zeichen direkt im Quelltext einzugeben.

Für häufiger benötigte Schrift- und Sonderzeichen stellt HTML ein ganzes Arsenal an so genannten Entities dar. Dies sind sinnige Namen oder Abkürzungen für Zeichen, denen ein kaufmännisches Und (&) vorangestellt wird und die von einem Semikolon (;) beendet werden. Ein paar Entities haben Sie bereits kennen gelernt: nämlich diejenigen, mit deren Hilfe HTML-eigene Zeichen maskiert werden müssen, also z. B. die Notation `<`, um das Zeichen < in HTML darzustellen. Ebenfalls hingewiesen wurde bereits auf die Entity `€` zur Darstellung des €-Symbols.

HTML stellt bereits seit Version 3.2 für alle höherwertigen Zeichen des Zeichensatzes ISO-8859-1 Entities zur Verfügung. Die nachfolgende Tabelle listet die HTML-Entities für höherwertige Zeichen im Deutschen auf:

Zeichenwert	Zeichen	HTML-Entity
196	Ä	`Ä`
214	Ö	`Ö`
220	Ü	`Ü`
223	ß	`&szlig ;`
228	ä	`&auml ;`
246	ö	`&ouml ;`
252	ü	`ü`

Tabelle 4.6: HTML-Entities für Sonderzeichen im Deutschen

HTML 4.0 erweitert die Entities um zahlreiche weitere Symbole, etwa für das komplette große und kleine griechische Alphabet mit Entities wie `Α`, `α`, `Β` und

`β` oder für mathematische Symbole wie `∑` für ? usw. In der Referenz im hinteren Buchteil (***Abschnitt 17.1.3***) finden Sie eine systematische Auflistung aller verfügbaren HTML-Entities.

Beliebige Zeichen, für die es keine Entity gibt, können Sie in HTML numerisch umschreiben. Dabei existieren zwei Schemata: `&#[Dezimalwert];` oder `&#x[Hexadezimalwert];`. In beiden Fällen beginnt die Zeichenfolge mit `&#`. Wenn Sie den Zeichenwert dezimal angeben, notieren Sie direkt dahinter den Wert. Bei einer hexadezimalen Angabe des Zeichenwerts müssen Sie noch ein `x` oder `X` einfügen. Falls Sie ein kleines `x` einfügen, sollten Sie die Hexadezimalziffern `a` bis `f` klein schreiben und bei einem großen `X` groß. So können Sie beispielsweise durch Eingabe von `☢` (dezimale Angabe) oder `☢` (hexadezimale Angabe) das übliche Symbol für Radioaktivität erzeugen.

Die numerischen Zeichenwerte müssen Sie in diesem Fall natürlich kennen. Da das Unicode-System mittlerweile zigtausend Zeichen mit zugeordneten Zeichenwerten umfasst, kann auch der Referenzteil dieses Buches in diesem Fall nicht weiterhelfen. Um nach Zeichen und deren Zeichenwerten zu suchen, müssen Sie direkt die Zeichentabellen von Unicode aufrufen. Auf *http://www.unicode.org/* werden solche Zeichentabellen in PDF-Form zur Verfügung gestellt.

Links zu weiteren Informationen:

Wikipedia-Artikel zu ISO 8859:
http://de.wikipedia.org/wiki/ISO_8859
Wikipedia-Artikel zu Unicode:
http://de.wikipedia.org/wiki/Unicode
RFC 3629 (UTF-8):
http://www.faqs.org/rfcs/rfc3629.html
ECMA-Standards (in der Tabelle nach ISO 8859 suchen):
http://www.ecma-international.org/publications/standards/Standard.htm

4.3.4 Dokumenttyp-Deklarationen

Im Quelltextbeispiel in ***Abschnitt 4.2*** haben wir gleich am Anfang der Datei, noch vor und außerhalb des alles umschließenden html-Elements, eine recht kompliziert wirkende Angabe notiert:

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

Solche Angaben finden sich in allen SGML- und XML-basierten Markup-Sprachen. Es handelt sich um die so genannte Dokumenttyp-Deklaration, die der verarbeitenden Software mitteilen, um welche Markup-Sprache in welcher Version es sich nachfolgend handelt.

Die gesamte Angabe wird in spitze Klammern eingeschlossen. Direkt hinter der öffnenden spitzen Klammer folgt ein Ausrufezeichen. Dies markiert einen Verarbeitungshin-

weis. Das Wort `DOCTYPE` bezeichnet die Art des Verarbeitungshinweises: In diesem Fall betrifft der Hinweis also den Dokumenttyp. Anschließend folgt die Angabe `HTML`, die der verarbeitenden Software mitteilt, dass es sich um die Markup-Sprache HTML handelt. Der Zusatz `PUBLIC` bedeutet, dass die verarbeitende Software das Markup gegen eine öffentlich verfügbare ***DTD*** (»Document Type Definition«, die SGML- oder XML-basierte Definition des Regelwerks von HTML) prüfen kann. Die URL der DTD wird am Ende der Dokumenttyp-Deklaration explizit genannt.

Die Zeichenfolge `//W3C//DTD HTML 4.01//EN` enthält weitere Angaben zur Markup-Sprache. `W3C` (also das W3-Konsortium) ist Herausgeber und verantwortlich für den Sprachstandard. Die ***Document Type Definition (DTD)*** betrifft die HTML-Version `4.01`, mit `EN` wird darauf hingewiesen, dass die natürliche Sprache für Elementnamen und andere Bezeichnungen der Markup-Sprache auf der englischen Sprache basieren.

HTML-Varianten

Von besonderer Bedeutung bei der Dokumenttyp-Deklaration zu HTML sind der Mittelteil und die URL am Ende. Diese sind veränderlich, und zwar je nachdem, für welche Variante von HTML 4.01 Sie sich entscheiden. HTML 4.01 hat nämlich leider noch mit Sünden der Vergangenheit zu kämpfen. Durch führende Browser-Anbieter wie Netscape und Microsoft, aber auch durch das W3-Konsortium selbst ist HTML in den Anfangsjahren des Webbooms in eine andere Richtung als heute gewünscht entwickelt worden. Da die Browser noch kein CSS kannten oder es nur völlig unzureichend unterstützten, entstand eine Fülle von HTML-Elementen und -Attributen, deren Zweck die optische Gestaltung von Inhalten war.

Viele dieser heute unerwünschten Elemente und Attribute wurden noch in HTML 4.01 übernommen, damit durch den neuen Sprachstandard nicht gleich Millionen von Webseiten völlig fehlerhaft würden. Um jedoch deutlich zu machen, dass HTML mittlerweile in eine andere Richtung zielt, hat sich das beim W3C für HTML verantwortliche Entwicklerteam einen Kompromiss ausgedacht. Dieser besteht darin, dass HTML 4.01 in drei verschiedenen Varianten angeboten wird, wobei eine Variante die eigentlich erwünschte ist, während die beiden anderen Varianten eher als ein vorübergehendes Tolerieren früher üblicher HTML-Praxis zu betrachten sind. Je mehr Jahre allerdings ins Land gehen, desto mehr schwindet diese Toleranz, und auch wenn nach wie vor alle HTML-Varianten offizieller Standard sind, so ist mittlerweile doch dringend anzuraten, für neu zu erstellende Webseiten nur noch die eigentlich gewünschte Variante zu verwenden.

Die eigentlich gewünschte Variante – das ist die ***Strict-Variante*** von HTML. Die dazu notwendige Dokumenttyp-Deklaration haben wir bereits in unserem ersten Quelltextbeispiel verwendet:

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

Falls Sie aus besonderen Gründen auf die früher verbreiteten HTML-Elemente und -Attribute zurückgreifen wollen oder müssen, müssen Sie stattdessen die *Transitional-Variante* von HTML verwenden und folgende Deklaration notieren:

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

Die dritte Variante von HTML 4.01 betrifft ausschließlich Dateien, in denen Framesets oder eingebettete Frames definiert werden. Diesen Spezialfall behandeln wir in *Abschnitt 6.1*.

Neben den Varianten für HTML 4.01 gibt es auch noch folgende möglichen Dokumenttyp-Deklarationen für ältere HTML-Versionen:

- `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 2.0//EN">` für HTML 2.0 (älteste heute noch deklarierbare Version von HTML)
- `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">` für HTML 3.2 (Zwischenversion aus den 90er Jahren)

Diese Deklarationen sollten Sie jedoch nur in wirklich begründeten Ausnahmefällen verwenden.

Dokumenttypen für XHTML

Wenn Sie Ihre Dokumente in XHTML auszeichnen, müssen Sie andere Dokumenttyp-Deklarationen verwenden. In XHTML entspricht die Spezifikationsversion 1.0 dem HTML-Standard 4.01. Dabei wird auch wieder zwischen der strict-Variante und der transitional-Variante unterschieden.

Die Dokumenttyp-Deklaration für die zu bevorzugende Variante *strict* lautet:

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

Die Dokumenttyp-Deklaration für die *transitional*-Variante lautet:

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

Neben XHTML 1.0 ist mittlerweile auch eine Version 1.1 offizieller Standard. In XHTML 1.1 gibt es keine Varianten mehr – alle Dokumente müssen der *strict*-Variante entsprechen. Die Dokumenttyp-Deklaration für XHTML 1.1 lautet:

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

Es ist jedoch nicht unbedingt sinnvoll, aus reinem Modernitätseifer die XHTML-Version 1.1 der Version 1.0 vorzuziehen. Denn erstens ist die Version 1.0 von den HTML-Parsern der meisten Browser problemloser zu verarbeiten und zweitens bringt XHTML 1.1 keine

für die meisten HTML-Autoren relevanten Neuerungen, abgesehen vielleicht von der Möglichkeit, so genannten *Ruby-Text* auszeichnen zu können (Ruby-Text ist eine Textform, die vor allem in fernöstlichen Sprachen verwendet wird, um Silben- oder Wortzeichen mit zusätzlichen Bedeutungshinweisen zu versehen). Die Zwischenversion von XHTML wurde hauptsächlich geschaffen, um ein XML-spezifisches Konzept für XHTML zu etablieren, nämlich die Möglichkeit der Modularisierung der Sprache. Dies ist jedoch recht komplex und wird in der heutigen Praxis so gut wie kaum genutzt, weshalb wir nicht näher darauf eingehen. Gleiches gilt für die Weiterentwicklung von XHTML 1.1, nämlich XHTML 2.0, das zur Drucklegung dieses Buches noch in der Entwicklung ist.

4.3.5 Editierregeln

HTML-Text wird vom Browser ähnlich wie in einem Textverarbeitungsprogramm automatisch umgebrochen. Darum brauchen Sie sich also nicht zu kümmern.

HTML hat eine besondere Art, mit Leerraum umzugehen: Mehrere so genannte *Whitespace-Zeichen* (das können Leerzeichen, Tabulatoren oder Zeilenumbrüche sein) werden per Default zu einem einzigen Leerzeichen zusammengefasst. Einzelne Tabulator- oder Zeilenumbruchzeichen werden bei der Darstellung im Browser zu einem Leerzeichen. Nur wenn Sie Text mit dem pre-Element für »präformatierten Text« auszeichnen oder in CSS für ein Element die Eigenschaft »präformatierte Darstellung« vergeben, wird der Text so wiedergegeben wie im Editor eingegeben.

Dieses Grundverhalten von HTML erleichtert das Bearbeiten auf Quelltextebene, da Sie den Quelltext sauber formatieren können, ohne dabei die HTML-Darstellung im Browser zu beeinflussen.

Das genannte Verhalten ist allerdings auch dafür verantwortlich, dass Sie, um mitten in einem darzustellenden Text mehrere Leerzeichen hintereinander zu erzwingen, das »erzwungene Leerzeichen« mit der numerischen Dezimalnotation ` ` mehrfach hintereinander eingeben müssen.

Tabulatoren gibt es in HTML überhaupt nicht. Da diese auch nicht wirklich für die saubere Formatierung taugen, sollten Sie auf Alternativen ausweichen, z. B. das Arbeiten mit randlosen (»blinden«) Tabellen.

Um innerhalb eines Block-Elements, also etwa eines Textabsatzes (zwischen `<p>` und `</p>`), einen Zeilenumbruch an einer bestimmten Stelle zu erzwingen, müssen Sie das dafür vorgesehene HTML-Tag `<br>` verwenden.

Kommentare

HTML erlaubt es, beliebige Passagen im Quelltext auszukommentieren. Das bedeutet, dass solche Passagen vom HTML-Parser ignoriert und etwa im Webbrowser nicht angezeigt werden. Die Kommentarfunktion von HTML ist nützlich, um aktuell nicht anzuzeigenden, aber mit zu speichernden Text auszuzeichnen. Sie lassen sich aber auch dazu

einsetzen, um alternative Markup-Konstrukte zu speichern, jedoch nur eines davon zu verwenden, oder um bestimmte Markup-Passagen zu erläutern. Dazu ein Quelltextbeispiel:

```
<div id="Inhalt">
<h1>Mein Leben mit <!-- Susi --><!-- Rosi -->Sissi</h1>
<!-- <p>Da gibt es gar nichts zu berichten</p> -->
<p>Es war langweilig</p>
</div><!-- Ende von div mit id="Inhalt" -->
```

Alles, was zwischen einem einleitenden `<!--` und einem schließenden `-->` steht, gilt in HTML als Inhalt eines Kommentars.

Sie können beliebig viele solcher Kommentare notieren, aber Sie können keine Kommentare ineinander verschachteln. Achten Sie auch darauf, dass durch das Setzen von Kommentaren kein syntaktisch fehlerhaftes (invalides) Markup entsteht. Verwenden Sie innerhalb eines Kommentars niemals zwei Bindestriche (--) hintereinander, da dies den HTML-Parser dazu veranlassen kann, das Ende des Kommentars zu erwarten.

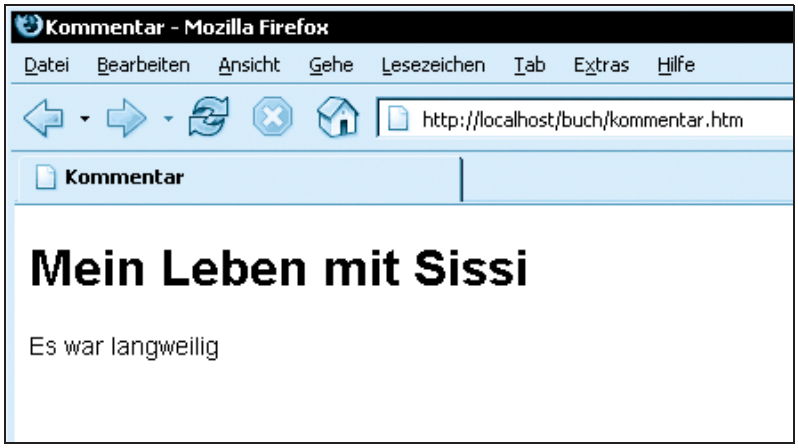


Abbildung 4.4: Wie der Webbrowser HTML-Kommentare verschweigt

In der Abbildung ist zu sehen, was von dem Quelltext-Beispiel am Ende im Browser zu sehen ist.

4.3.6 XHTML-spezifische Syntaxregeln

Auf die unterschiedlichen Mime-Typen bei HTML und XHTML sind wir bereits eingegangen. Auch die unterschiedlichen Dokumenttyp-Deklarationen wurden bereits behandelt. Daneben gibt es jedoch auch konkrete Unterschiede bei der Notation. Die folgende Aufzählung fasst die wichtigsten Unterschiede kurz zusammen:

- Für XHTML-Dateien empfiehlt sich unter Umständen eine XML-Deklaration.
- Namensraumangabe im html-Element.

- Zwingende Kleinschreibung von Element- und Attributnamen.
- Abweichende Notation von Elementen ohne Inhalt.
- Keine Elemente mit optionalem Abschluss-Tag.
- Keine Attribute ohne Wertzuweisung.

Einige weitere Unterschiede betreffen konkrete Elemente oder Attribute. Wir werden im Buch an entsprechender Stelle darauf hinweisen.

XML-Deklaration

XML-Daten beginnen normalerweise in der ersten Zeile mit einer XML-typischen Deklaration. Diese ist in spitze Klammern eingeschlossen. Unmittelbar nach der öffnenden Klammer und vor der schließenden Klammer wird ein Fragezeichen notiert. Ein Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Bei XHTML können Sie diese Zeile weglassen, wenn Sie als Zeichenkodierung UTF-8 oder UTF-16 wählen. Das bedeutet aber auch, dass Sie, wie weiter oben bereits beschrieben, Ihre Dateien mit dieser Zeichenkodierung abspeichern müssen. Wenn Sie eine andere Kodierung verwenden möchten, etwa den Zeichensatz ISO-8859-1, dann müssen Sie die XML-Deklaration unbedingt notieren, und zwar mit der entsprechenden Angabe, also z. B.:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Namensraumangabe im html-Element

Das alles einschließende Wurzelelement in HTML hat keine zwingend erforderlichen Attribute und wird meist einfach als `<html>[Inhalt]</html>` notiert. In XHTML muss das Wurzelelement eines Dokumenttyps jedoch eine Namensraumangabe besitzen. Dazu steht das xmlns-Attribut zur Verfügung, das es in HTML gar nicht gibt. Notieren Sie bei XHTML folgende Angabe:

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

Zwingende Kleinschreibung

Viele HTML-Editoren fügen HTML-Tags in Großschreibweise ein. Für XHTML ist dies auf jeden Fall verkehrt. Alle Element- und Attributnamen müssen durchgängig klein geschrieben werden.

Falsch ist also:

```
<H1 STYLE="font-size:150%">Das Neue ist <Em>notwendig</Em></H1>
```

Richtig ist dagegen:

```
<h1 style="font-size:150%">Das Neue ist <em>notwendig</em></h1>
```

Elemente ohne Inhalt (Singleton-Elemente)

So genannte Singleton-Elemente oder Standalone-Tags, also Elemente, die nur aus einem Anfangs-Tag bestehen, sind in der XML-Syntax im Gegensatz zur SGML-Syntax nicht möglich. Für entsprechende Elemente gibt es zwei mögliche Schreibweisen.

Verkürzte Schreibweise:

```
<meta name="author" content="halogen@gmx.de" />
```

Ausführliche Schreibweise:

```
<meta name="author" content="halogen@gmx.de"></meta>
```

Bei der verkürzten Schreibweise wird nur ein Tag notiert. Dieses wird jedoch durch einen Schrägstrich unmittelbar vor der schließenden spitzen Klammer als leeres Element gekennzeichnet. Lassen Sie vor dem Schrägstrich stets ein Leerzeichen, um den Parser nicht durcheinander zu bringen. Bei der ausführlichen Schreibweise werden »leere« Elemente einfach wie Elemente mit Inhalt notiert, nämlich durch Anfangs- und Abschluss-Tag. Dazwischen darf jedoch nichts stehen, nicht einmal ein Leerzeichen.

Aus praktischer Sicht ist die verkürzte Schreibweise vorzuziehen. Sie wird auch häufiger eingesetzt, da sie weniger fehleranfällig ist.

Keine Elemente ohne optionale Abschluss-Tags

In HTML gibt es einige Elemente, bei denen laut Spezifikation nur ein Anfangs-Tag zwingend vorgeschrieben ist, jedoch kein Abschluss-Tag. Das betrifft vor allem in Serie vorkommende Tags wie `<td>` (Tabellenzelle), `<li>` (Listenpunkt) oder `<option>` (Eintrag in einer Auswahlliste). Obwohl die Notation ohne Abschluss-Tag ohnehin eine zweifelhafte Freiheit ist, von der man besser keinen Gebrauch machen sollte, besteht bei XHTML die zwingende Pflicht, solche Elemente durch Tags wie `</td>`, `</li>` oder `</option>` zu schließen.

Keine Attribute ohne Wertzuweisung

Eine andere Spezialität von SGML-basierten Sprachen sind Attribute, deren Wert als »Boolean« zu verstehen ist in dem Sinne, dass bei Nennung des Attributs der Wert als 1 oder wahr interpretiert wird und ansonsten auf 0 oder falsch stehen bleibt. In HTML kommen einige wenige Attribute dieser Art vor, z. B. `<td nowrap>` (Tabellenzelle, deren Inhalt nicht automatisch umgebrochen werden darf). In XHTML müssen solche Attribute eine Wertzuweisung erhalten. Per Konvention hat man festgelegt, dass als Wert einfach der Attributname zugewiesen wird. Die korrekte XHTML-Version von `<td nowrap>` lautet daher `<td nowrap="nowrap">`.

4.3.7 Beispiel einer vollständigen XHTML-Datei

Um die Unterschiede zwischen HTML und XHTML am konkreten Beispiel zu sehen, werden wir nun noch einmal das Quelltextbeispiel aus ***Abschnitt 4.2.1*** listen, diesmal jedoch in XHTML-Syntax. Vergleichen Sie die Quelltexte und testen Sie sich selbst, ob Sie die Unterschiede gleich erkennen.

Listing 4.2: Eine vollständige XHTML-Datei

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!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 http-equiv="content-type" content="text/html;
            charset=ISO-8859-1" />
      <title>Gemeinden mit A...</title>
   </head>
   <body>
      <div id="Kopf">
         <h1>Gemeinden mit A...</h1>
      </div>
      <div id="Inhalt">
         <p>Die Links führen zu den offiziellen Internet-
         Präsenzen der jeweiligen Gemeinden!</p>
         <ul>
            <li><a href="http://www.aachen.de/">Aachen</a> -
            Regierungsbezirk Köln</li>
            <li><a href="http://www.aalen.de/">Aalen</a> -
            Regierungsbezirk Stuttgart</li>
            <li><a href="http://www.achern.de/">Achern</a> -
            Regierungsbezirk Freiburg</li>
            <li><a href="http://www.ahaus.de/">Ahaus</a> -
            Regierungsbezirk Münster</li>
            <li><a href="http://www.ahlen.de/">Ahlen</a> -
            Regierungsbezirk Münster</li>
         </ul>
      </div>
   </body>
</html>
```

Im Beispiel sind die Unterschiede ausschließlich im oberen Teil zu finden: XML-Deklaration wegen Zeichensatz ISO-8859-1, XHTML-Dokumenttyp-Deklaration, html-Element mit Namensraumangabe für XHTML, alleinstehendes <meta>-Tag für HTML-interne Zeichensatzangabe mit XHTML-spezifischer Abschlussmarkierung. Im Beispiel ist das jedoch schon alles. Wie Sie daraus ersehen können, sind die Unterschiede zu XHTML bei sauberer HTML-Kodierung eher geringfügig und eine spätere automatisierte Umstellung auf XHTML sollte keine größeren Probleme bereiten.

Links zu weiteren Informationen zu XHTML:

Wikipedia-Artikel zu ISO 8859:
http://de.wikipedia.org/wiki/ISO_8859
Artikel »Keine Angst vor XHTML«:
http://www.webwriting-magazin.de/sogehts/xhtml.shtml
Modularization of XHTML (W3C-Recommendation): Wikipedia-Artikel zu ISO 8859:
http://www.w3.org/TR/xhtml-modularization/
Modularisierung von XHTML (deutsche Übersetzung):
http://www.edition-w3c.de/TR/xhtml-modularization

4.4 Kopfinformationen einer Webseite

Es gibt noch viel mehr Webseiten, als es Bücher gibt. Und für Bücher existieren bewährte Systeme, die eine Unterscheidung und Bestellung der Bücher im Handel erlauben: ISBN-Nummer, ZIP-Kurztitel, Erscheinungsangabe, Auflagenangabe usw. Bei Webseiten sind es jedoch keine Bibliothekare und Buchhändler, sondern Softwareprodukte wie Webserver, Such-Robots von Suchmaschinen und Browser, die Webseiten korrekt identifizieren, einordnen und in Beziehung zueinander setzen wollen oder müssen. Deshalb sind in HTML eine Reihe wichtiger Kopfdaten vorgesehen. Leider wird diesen Daten von vielen HTML-Autoren immer noch zu wenig Sorgfalt gewidmet.

Unter Kopfinformationen einer Webseite verstehen wir alles, was in HTML zwischen `<head>` und `</head>` notiert wird.

4.4.1 Dokumenttitel

Ein HTML-Dokument (nicht XHTML!) benötigt nicht zwingend ein umschließendes html-Element, nicht mal ein head- oder body-Element. Aber ein title-Element ist immer vorgeschrieben, wenn gültiges HTML erzeugt werden soll. In der Praxis sollten Sie zwar niemals auf das html-, das head- und das body-Element verzichten, doch soll dieser Hinweis einfach verdeutlichen, welch große Bedeutung dem Titel des Dokuments beigemessen wird. Notiert wird er ganz einfach:

```
<title>Was auch immer</title>
```

Da es sich um wichtigen sprachlichen Inhalt handelt, kann es nichts schaden, die Sprache mit anzugeben, z. B.:

```
<title lang="de">Was auch immer</title>
```

Wenn Sie übrigens bei XHTML das lang-Attribut einsetzen wollen, notieren Sie stets folgende Doppelform:

```
<[elementname] lang="de" xml:lang="de">
```

Die immense Bedeutung des Dokumenttitels ist einfach zu erklären:

- Der Titel erscheint bei Aufruf des Dokuments bei den meisten Browsern in der Titelleiste.
- Der Titel wird angeboten, wenn der User ein Bookmark/Favoriten auf die Seite setzt.
- Der Titel erscheint in der History-Liste der bereits besuchten Webseiten im Browser.
- Der Titel ist bei Suchmaschinen der anklickbare Verweistext, wenn die betreffende Seite dort als Treffer zu einer Suche aufgelistet wird.

Es lohnt sich also, dem Titel die nötige Aufmerksamkeit zu widmen. Da Websites ja in aller Regel aus einer Menge von einzelnen Seiten bestehen, die aber irgendwie zusammengehören und ein Ganzes bilden, liegt es nahe, sich ein einheitliches Schema für die Dokumenttitel der einzelnen Seiten zu überlegen. Es ist beispielsweise schlecht, wenn der Titel der Produktübersichtsseite »Firma XY: Produktübersicht« lautet und der Titel der Kontaktübersichtsseite »Kontaktseite«. Sinnvoller wäre ein einheitliches Schema wie »XY: Produktübersichtsseite« und »XY: Kontaktübersichtsseite«.

Der Titel sollte nicht zu lang sein. Stellen Sie sich den Titel dazu am besten als Menüeintrag in den Favoriten des MS Internet Explorers oder als Eintrag in der als Sidebar zuschaltbaren Bookmark-Baumstruktur bei Browsern wie Firefox oder Opera vor.

4.4.2 Meta-Angaben

Im Gegensatz zum Dokumenttitel sind die Meta-Angaben (häufig auch *Meta-Tags* genannt) in HTML ein Fass ohne Boden, ein Gegenstand von Voodoo-Glaube und ein Instrument, das von Gremien des Bibliothekswesens bis hin zum Jugendschutz gehätschelt wird. Über Meta-Angaben in HTML gibt es im Web fast ebenso viele Feature-Seiten wie zum Thema HTML allgemein.

Der Grund ist, dass Meta-Angaben die HTML-eigene Schnittstelle für jede Art von Information über ein Dokument darstellen. HTML selber gibt nur zwei Pflichtattribute vor, die ein Meta-Tag enthalten muss: entweder `name=` oder `http-equiv=` und `content=`. Dazu zwei Beispiele:

```
<meta http-equiv=" Last-Modified" content=" Tue, 15 Nov 2004 12:45:26 GMT">
<meta name="keywords" content="Fahrrad, Bike, Tour">
```

Das Attribut `http-equiv=` stellt eine Information für den Webserver bereit. Es handelt sich im Grunde um Kopfanweisungen des HTTP-Protokolls (HTTP-Header), die vom Webserver an einen aufrufenden Client (z. B. einen Browser) gesendet werden, wenn dieser die Seite aufruft.

Dagegen enthalten Meta-Angaben mit dem Attribut `name=` Informationen, die sich an diverse auslesende Programme oder Scripts richten. Das kann beispielsweise der aufrufende Browser sein, eine Zusatz-Software wie z. B. eine Kindersicherung oder der Robot einer Suchmaschine.

Die Beispielangabe zu `http-equiv` verwendet das HTTP-Header-Kommando `Last-Modified`. Dieses enthält eine Angabe darüber, wann das Dokument zuletzt geändert wurde. Ein Webserver kann diese Angabe bei der Abstimmung mit einem aufrufenden Browser darüber nutzen, ob eine Seite aus dem Browser-Cache geladen werden kann oder ob sie neu vom Server übertragen werden soll.

Die Meta-Angabe zu den keywords (wichtige Stichwörter zum Dokumentinhalt) orientiert sich dagegen eher an Suchmaschinen-Robots. Diese können beispielsweise so reagieren, dass die Webseite bei der Suche nach einem der in dieser Angabe genannten Stichwörter besonders weit oben in den Suchtreffern aufgelistet wird, da davon auszugehen ist, dass das Dokument zu den Stichwörtern besonders viele oder wichtige Informationen enthält. Leider ist diese schöne Theorie aber von der Schäbigkeit und dem Geltungsdrang vieler Seitenanbieter zunichte gemacht worden, die dieses Verhalten von Such-Robots dazu nutzten, um ihre keyword-Angaben mit lauter oft gesuchten Begriffen zu schmücken, ohne Bezug zum tatsächlichen Inhalt des Dokuments. Da wurde beispielsweise versucht, zu vermietende Ferienhäuser durch sexuelle Stichwörter in den Suchmaschinen besser zu platzieren. Daher werden keyword-Angaben von heute maßgeblichen Robots wie dem von Google gar nicht mehr besonders gewichtet.

Dringend zu empfehlende Meta-Angaben

Folgende Meta-Angaben sind in jedem Fall zu empfehlen und sollten auf keiner Webseite fehlen:

- Angabe zur Zeichenkodierung bzw. zum Zeichensatz
- Angabe mit einer allgemeinen Beschreibung des Inhalts

Die Angabe zum Zeichensatz bzw. zur Zeichenkodierung haben Sie bereits kennen gelernt. Hier noch einmal zwei Beispiele:

```
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<meta http-equiv="content-type" content="text/html;
      charset=UTF-8">
```

Die Angabe richtet sich durch das Attribut `http-equiv=` also bereits an den Webserver, damit dieser das Dokument korrekt verarbeiten kann. Dem Attribut wird als Wert `content-type` zugewiesen. Das obligatorische Attribut `content=` erhält in diesem Fall einen Wert, der einem bestimmten Schema folgt. Der Wert besteht aus zwei Angaben, die durch ein Semikolon voneinander getrennt sind. Der erste Teil ist der Mime-Type, mit dem der Webserver das Dokument an aufrufende Clients ausliefern soll, und der zweite Teil enthält die Angabe zum Zeichensatz oder zum Zeichenkodierungsprinzip. Wertzuweisungen, die einem solchen Schema entsprechen, sind bei Meta-Angaben übrigens nicht selten. Das Schema ist jedoch nicht in der HTML-Spezifikation vorgeschrieben, sondern in der Spezifikation oder Konvention der Software, für die die Angabe gedacht ist.

Während die Meta-Angabe zum Mime-Type und Zeichensatz so wichtig ist, dass beispielsweise der HTML-Validator des W3-Konsortiums (*http://validator.w3.org/*) bei deren Fehlen sofort meckert, ist unsere zweite dringend empfohlene Angabe eher für die Praxis besonders wichtig. Nicht wenige Suchmaschinen listen nämlich, wenn eine Webseite zu den Treffern einer Suche gehört, neben dem anklickbaren Dokumenttitel auch den Inhalt der Meta-Angabe mit einer allgemeinen Beschreibung auf.



Abbildung 4.5: Beschreibungen aus Meta-Angaben in der Suchmaschine Fireball

Nehmen wir als Beispiel den ersten Suchtreffer aus der Abbildung. Er führt uns zur Startseite der Domain *http://www.sibiweb.de/*. Im Quelltext dieser Seite findet sich folgende Meta-Angabe:

```
<meta name="description" content="Informationsforum für alle Themen rund um die
Region Siebenbürgen und die Siebenbürger Sachsen.">
```

Wie Sie der Abbildung entnehmen können, verwendet die Suchmaschine Fireball diese vom Webautor eingepflegte Meta-Angabe, um unterhalb des anklickbaren Suchtreffers den Inhalt der Seite kurz zu beschreiben. Es ist also in Ihrem eigenen Interesse, dass diese Meta-Angabe in Ihren HTML-Dokumenten nicht fehlt und einen nicht zu langen, aber aussagekräftigen Inhalt hat, der die Angaben im Dokumenttitel sinnvoll ergänzt.

Die Meta-Angabe zur Seitenkurzbeschreibung wird durch die Attribut-Wert-Kombination `name="description"` realisiert. Bei `content=` können Sie dann die gewünschte Kurzbeschreibung eingeben. Innerhalb des Beschreibungstextes ist übrigens kein weiteres HTML-Markup erlaubt!

Weitere Meta-Angaben zum Dokumentinhalt

Meta-Daten zur einzelnen Webseite können nicht nur für fremde Software, sondern auch für eigene Zwecke sinnvoll sein, etwa als Dateninput in eine eigene, site-spezifische Suchmaschine. Aber auch, wenn Sie einfach nur dokumentspezifische Daten wie das Veröffentlichungsdatum oder die Autoren des Inhalts festhalten wollen, haben sich entsprechende Meta-Angaben etabliert. Nachfolgend einige Beispiele:

```
<meta name="author" content="Anna Hirsch, ahirsch@ahirsch.info">
<meta name="date" content="2005-10-15T08:30:00+01:00">
<meta name="keywords" content="Sommersalate, leichte Kost, Rezepte">
```

Mit den Beispielangaben wird Anna Hirsch als Autorin der Seite ausgewiesen, als Veröffentlichungszeitpunkt der Seite wird der 15. Oktober 2005, 8:30 Uhr (gemessen mit + 1 Stunde gegenüber Greenwich-Zeit) angegeben und ein paar Stichworte charakterisieren den Inhalt der Seite. Die Datums- und Zeitangabe folgt auch wieder einem festen Schema, das von vielen Softwareprodukten in dieser Form erkannt wird.

Meta-Angaben nach Dublin Core

Für die meisten Seiten reichen die zuvor genannten Meta-Angaben zum Dokumentinhalt aus. Bei umfangreicheren Publikationen mit wissenschaftlichem Anspruch entstehen jedoch weiterreichende Wünsche. HTML macht Ihnen wie schon erwähnt keine Vorschriften, was den Inhalt von Meta-Tags betrifft. Doch ist es immer sinnvoller, wenn ein Standardisierungsgremium sich um ein allgemein anerkanntes Schema bemüht, das dann auf möglichst breiter Basis Akzeptanz findet. So hat sich eine internationale Gruppe von Experten unter dem Namen *Dublin Core* vereint, um ein Meta-Angaben-Schema für Webseiten zu etablieren, das wissenschaftlichen und bibliothekarischen Ansprüchen gerecht wird.

Nachfolgend ein Beispiel für ein Set an Meta-Angaben nach dem Dublin-Core-Schema:

```
<meta name="DC.Title" content="Zur Urknall-Theorie">
<meta name="DC.Creator" content="Harald Sternguck">
<meta name="DC.Subject" content="Astronomie">
<meta name="DC.Description" content="Pro- und Contra-Argumente zur Urknall-
Theorie aus philosophischer Sicht">
<meta name="DC.Publisher" content="Web-Magazin: Wissen gestern, heute, morgen">
<meta name="DC.Contributor" content="Pia Lichtblau">
<meta name="DC.Date" content="2005-10-15">
<meta name="DC.Type" content="Text">
<meta name="DC.Format" content="text/html">
<meta name="DC.Identifier" content="http://server/urknall/">
<meta name="DC.Source" content="Buch / ISBN:7-23094-2389047-5">
```

```
<meta name="DC.Language" content="de">
<meta name="DC.Relation" content="Artikel Printausgabe 05/2005, S.47ff">
<meta name="DC.Coverage" content="21st century">
<meta name="DC.Rights" content=" http://server/copyright.htm">
```

Alle Meta-Angaben nach Dublin Core werden durch das name-Attribut identifiziert. Die Wertzuweisung an dieses Attribut besteht stets nach dem Schema `DC.Angabetyp`, wobei das DC für »Dublin Core« steht.

`DC.Title` wiederholt noch einmal den Dokumenttitel als Meta-Angabe. `DC.Creator` ist der Autor oder die inhaltsverantwortliche Person oder Organisation. `DC.Subject` soll den zugehörigen Themenbereich eingrenzen. `DC.Description` entspricht der gewöhnlichen Meta-Angabe zur Kurzbeschreibung des Dokumentinhalts. `DC.Publisher` kann sich von `DC.Creator` unterscheiden, wenn die Webseiten nicht vom Autor selber betreut werden, sondern von einem Drittanbieter. `DC.Contributor` ist für Personen oder Organisationen gedacht, die zum Inhalt oder zur Form beigetragen haben, z.B. Übersetzer oder Illustratoren. `DC.Date` enthält das Datum der Veröffentlichung, und zwar ohne Uhrzeit in der Form JJJJ-MM-TT.

Bei `DC.Type` soll der Dokumenttyp charakterisiert werden – Dublin Core schlägt dazu die folgenden Angaben vor: `Collection` (Zusammenstellung), `Dataset` (für Listen oder Tabellen), `Event` (wenn der Inhalt z.B. eine Terminankündigung oder dergleichen ist), `Image` (wenn der Inhalt im Wesentlichen eine Grafik ist), `InteractiveResource` (wenn der Inhalt z.B. wesentlich aus einem Flash-Movie für User-Interaktion besteht), `MovingImage` (wenn der Inhalt wesentlich aus einem Video besteht), `PhysicalObject` (wenn der Inhalt wesentlich aus einer 3D-Darstellung oder -Animation besteht), `Service` (wenn es sich z.B. um einen User-Service wie Online-Banking handelt), `Software` (wenn der Inhalt wesentlich aus der Download- oder Ausführmöglichkeit einer Software besteht), `Sound` (wenn der Inhalt wesentlich aus Musik oder Geräuschen besteht, z.B. bei einem Webradiosender), `StillImage` (eine Spezialform von Image) und `Text` (wohl der häufigste Typ).

`DC.Format` ist der Mime-Type des Dokuments, bei HTML also im Normalfall text/html. `DC.Identifier` ist eine Angabe zum »Original« des Dokuments. Bei Webseiten ist dies sinnvollerweise die Original-URL, wo das Dokument gespeichert ist. `DC.Source` ist die Angabe zu einer Quelle, die für das aktuelle Dokument inhaltlich maßgeblich war. `DC.Relation` stellt den Bezug zu einer anderen Publikation her, z.B. zu einer Printversion des aktuellen Dokuments. `DC.Coverage` ist etwas schwer zu fassen und soll einen räumlichen oder zeitlichen Bezug herstellen. `DC.Rights` schließlich ist ein Hinweis auf juristisch relevante Hintergrundinformation zum aktuellen Dokument, z.B. auf AGBs, Copyright-Hinweise usw. Idealerweise wird dazu eine URL-Adresse angegeben, unter der die betreffende Information zu finden ist.

Niemand zwingt Sie, stets das ganze DC-Set zu notieren, wenn Sie mit den DC-Meta-Tags arbeiten möchten. Was sich aus dem Set letztlich verwendet lässt, hängt selbstverständlich vom Inhalt und von der Art der eigenen Webseiten ab. Andersherum sind auch Mehrfachangaben zulässig, z.B. wenn mehrere Personen bei `DC.Contributor` zu nennen sind.

In Suchmaschinen wie Google spielen die DC-Angaben bislang eine eher untergeordnete oder gar keine Rolle.

Spezielle Meta-Angaben für Such-Robots

Einige Meta-Angaben orientieren sich speziell an Such-Robots, also an den Programmen der Suchmaschinen, die Webseiten scannen und den Inhalt in die Datenbank der Suchmaschine übertragen. Folgende Meta-Angaben sind je nach Situation sinnvoll:

```
<meta name="robots" content="noindex">
```

Mit dieser Angabe verbieten Sie einem Suchprogramm, den Inhalt des HTML-Dokuments an seine Suchdatenbank zu übermitteln. Das kann z. B. für Impressumsseiten interessant sein, auf denen Sie möglicherweise persönliche Daten wie Wohnadresse, Umsatzsteuer-ID, Bankverbindung usw. angeben, von denen Sie aber nicht unbedingt wünschen, dass diese über Google und andere Suchmaschinen direkt auffindbar sind.

```
<meta name="robots" content="nofollow">
```

Diese Angabe ergänzt die Angabe für `noindex`, indem Sie dem Such-Robot verbieten, Links, die in dem aktuellen HTML-Dokument enthalten sind, zu folgen.

```
<meta name="revisit-after" content="14 days">
```

Diese Angabe empfiehlt dem Such-Robot, die Seite spätestens nach 14 Tagen erneut zu scannen, da anzunehmen ist, dass sich bis dahin ihr Inhalt geändert hat.

Spezielle Meta-Angaben für Webserver

Dies sind Meta-Tags mit dem Attribut `http-equiv=`. Es besteht keine Garantie, dass jede Angabe von jedem Webserver oder von einem von ihm ausgeführten Script tatsächlich und korrekt interpretiert wird. Dennoch haben sich eine Reihe von Meta-Angaben etabliert, die in vielen Fällen funktionieren und nützliche Dienste leisten können. Folgende Angaben gehören dazu (Beispiele):

```
<meta http-equiv="content-encoding" content="gzip">
```

Diese Angabe weist den Webserver an, das Dokument wenn möglich ZIP-komprimiert an den Browser auszuliefern. Bei textlastigen Dokumenten kann sich dadurch die zu übertragende Datenrate erheblich reduzieren. Das Dokument kann deutlich schneller im Browser angezeigt werden.

```
<meta http-equiv="expires" content="Wed, 1 Jan 2006 00:00:00 GMT">
```

Diese Angabe bedeutet, dass das Dokument nach dem angegebenen Zeitpunkt bei Anforderung nicht mehr aus einem Cache-Speicher (z. B. aus dem Browser-Cache) geholt werden darf, sondern in jedem Fall neu vom Server übertragen werden muss. Sinnvoll ist

diese Angabe, wenn Sie wissen, dass das Dokument bis zu diesem Zeitpunkt auf jeden Fall einen aktuelleren Inhalt haben wird.

```
<meta http-equiv="refresh" content="3;URL=http://server/">
```

Diese in der Praxis sehr häufig verwendete Angabe bietet HTML-Autoren die Möglichkeit, ohne weitere Serverkenntnisse eine Weiterleitung zu einer anderen Adresse zu notieren. Das aktuelle Dokument wird dabei zwar in den Browser geladen, doch dieser führt nach der angegebenen Anzahl Sekunden (im obigen Beispiel 3) einen automatischen Redirect auf die angegebene Adresse aus.

```
<meta http-equiv="cache-control" content="no-cache">
```

Mithilfe dieser Angabe können Sie verhindern, dass ein aufrufender Browser das Dokument in seinem Browser-Cache speichert. Sie stellen stattdessen sicher, dass das Dokument bei jedem Aufruf neu vom Server übertragen werden muss. Sinnvoll ist dies vor allem bei Dokumenten, die sich sehr oft ändern, und vor allem bei serverseitig dynamisch erzeugten Dokumenten (also z. B. bei HTML-Output von PHP-Scripts).

```
<meta http-equiv="cache-control" content="maxage=86400">
```

Auch diese Angabe beeinflusst das Cache-Verhalten des Browsers. Die maxage-Angabe legt fest, wie viele Sekunden (im Beispiel 86400, das entspricht einem Kalendertag) das Dokument höchstens im Cache verbleiben darf, bevor es bei einer neuen Anforderung neu vom Server übertragen werden muss. Sinnvoll ist eine solche Angabe z. B. für die Datei eines Newstickers, der einmal täglich aktualisiert wird.

Links zu weiteren Informationen über Meta-Angaben:

Dublin Core:
http://dublincore.org/
Diplomarbeit »Metadaten im Web«:
http://wwwai.wu-wien.ac.at/usr/albert/html/
Artikel »Meta-Tags und Markenrecht«:
http://www.devmag.net/projektpflege/recht_meta_tags.htm

4.4.3 Logische Verlinkung mit anderen Inhalten

Die logische Verlinkung mit anderen Seiten ist eines der spannendsten Hypertext-Features von HTML, obwohl sie immer noch viel zu selten auf Webseiten genutzt wird. Der Grund dafür ist, dass das entsprechende link-Element, das im Dateikopf notiert wird und verschiedenste Arten der Verlinkung erlaubt, lange Zeit von allen Browsern ignoriert wurde. Was um so erstaunlicher ist, als das Element ein Urgestein von HTML ist und seit den frühen 90er Jahren zum HTML-Standard gehört.

Immerhin ist das link-Element vielen Autoren dadurch vertraut geworden, dass es die HTML-übliche Methode zum Einbinden externer Stylesheets ist. Doch das Element kann

viel mehr und wer eine anwenderfreundliche Website anbieten will, sollte sich die Mühe machen, eine zusätzliche Navigation mithilfe des link-Elements zu realisieren.

Das link-Element wird im Dateikopf, also zwischen `<head>` und `</head>` notiert. Ein erstes Beispiel:

```
<link rel="search" title="Suchen" href="/cgi-bin/suche.cgi">
```

Das Beispiel verweist auf eine site-spezifische Suche. Browser, die das link-Element in der angedachten Weise interpretieren (z.B. Firefox mit entsprechendem Zusatz-Plug-In oder Opera), bieten eine Navigationsleiste an, über die ein entsprechender Eintrag verfügbar ist.

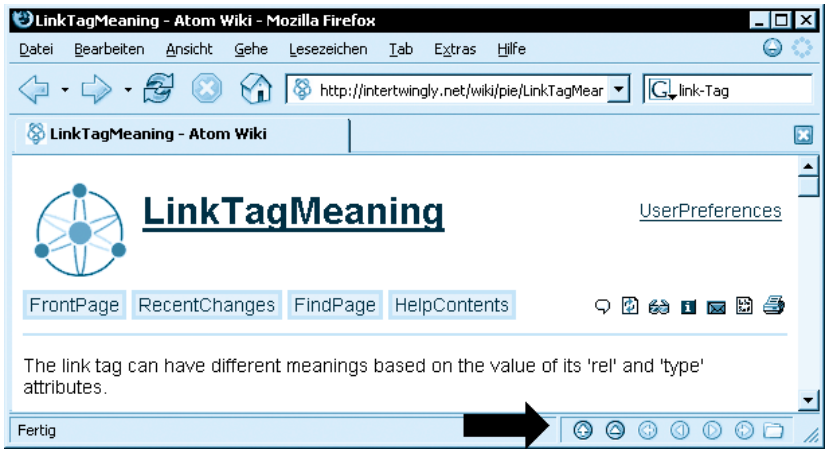


Abbildung 4.6: link-Navigationsleiste im Firefox-Browser

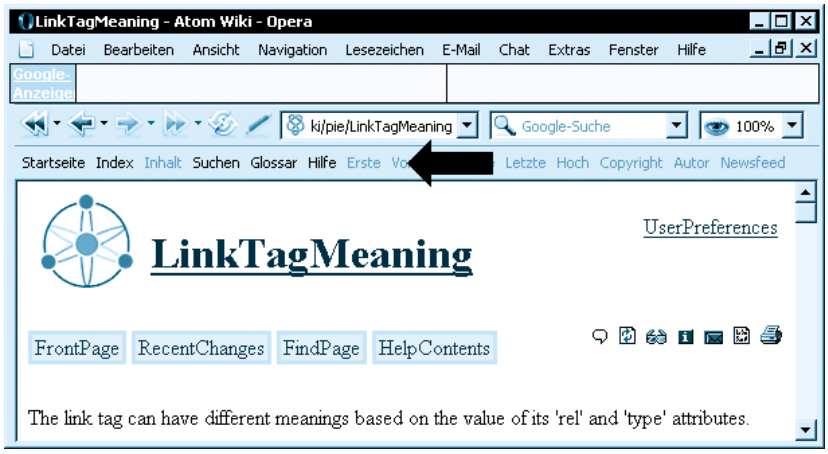


Abbildung 4.7: link-Navigationsleiste im Opera-Browser

Die schwarzen Pfeile in den Screenshots zeigen die Navigationsmöglichkeiten, die entsprechend ausgerüstete Browser anbieten, um dem Anwender die logischen Links des link-Elements verfügbar zu machen.

In welcher Weise die Navigation im Browser realisiert wird, ist nicht vorgeschrieben und produktspezifisch. Anwender, die sich an die Bedienung jedoch erst mal gewöhnen, wissen sie zu schätzen.

Aufbau von logischen Links

Betrachten wir das Markup von logischen Links genauer:

```
<link rel="search" title="Suchen" href="/cgi-bin/suche.cgi">
```

Jeder logische Link wird durch ein `<link>`-Tag realisiert, das in HTML kein Abschluss-Tag hat. In XHTML muss daher am Ende die Zeichenfolge `/>` notiert werden. Links dieser Art haben stets drei Attribute. Mit dem Attribut `rel=` wird der Typ des Links bestimmt. Mit `title=` vergeben Sie die Beschriftung der Schaltfläche oder des Menüeintrags, die der Browser dem Anwender in der Navigation für diesen Link anbietet. Bei `href=` geben Sie die URL-Adresse an, die der Browser aufrufen soll, wenn der Anwender den Link auswählt. Es kann sich um Adressierungen wie `http://mein.server.xy/irgendwo.html` handeln, aber auch um lokale Adressierungen wie `/irgendwo.html` oder `../../irgendwo.html`.

Als Alternative zum rel-Attribut ist im HTML-Standard auch noch ein rev-Attribut für Rückverweise vorgesehen. Dies hat jedoch in der Praxis bislang keine Bedeutung.

Nützliche logische Links

Nachfolgend Beispiele für logische Links, die für viele Websites geeignet sind:

```
<link rel="contents" href="sitemap.htm" title="Inhalt">
```

Mit `rel="contents"` verweisen Sie auf eine Seite innerhalb Ihres Projekts, die ein Inhaltsverzeichnis darstellt. Viele Sites, die kein zentrales Inhaltsverzeichnis im Stil eines Buches haben, bieten als Alternative eine so genannte ***Sitemap*** an.

```
<link rel="chapter" href="index.htm" title="Kapitel">
```

Mit `rel="chapter"` verweisen Sie auf die Startseite eines Kapitels. Eine Kapitelunterteilung, die sich stark an der Buchmetapher orientiert, findet sich vor allem bei wissenschaftlich orientierten Webpublikationen oder bei technischen Dokumentationen im Web.

```
<link rel="index" href="stw.htm" title="Stichwörter">
```

Mit `rel="index"` sollten Sie nicht, wie das Wort nahe legt, auf die Startseite der Website verweisen, sondern auf ein Begriffs- oder Stichwortverzeichnis oder zumindest auf irgendeine vergleichbare Art von alphabetisch oder anderweitig sortierter Übersicht mit anklickbaren Links.

```
<link rel="copyright" href="agb.htm" title="AGB">
```

Mit `rel="copyright"` sollten Sie eigentlich auf eine Seite verweisen, die Angaben zum geistigen Schutz Ihrer Inhalte enthält. Da das Copyright im deutschen Recht allerdings ohnehin durch das Urheberrechtsgesetz klar geregelt ist, lässt sich der Copyright-Link auch »umbiegen«, um damit juristisch relevante Informationen zu verlinken, die je nach Webangebot im Gegensatz zum Urheberrechtshinweis explizit veröffentlicht werden müssen. So ist beispielsweise ein Shop-Betreiber verpflichtet, seine allgemeinen Geschäftsbedingungen (AGB) explizit auf seinen Webseiten abrufbar zu halten. Für diesen Zweck ist der logische Link im obigen Beispiel entfremdet worden.

```
<link rel="next" href="page-5.htm" title="nächste Seite">
<link rel="prev" href="page-3.htm" title="vorherige Seite">
<link rel="up" href="index.htm" title="nach oben">
<link rel="top" href="augsburg.htm" title="Homepage">
```

Diese logischen Links ermöglichen dem Anwender ein bequemes Navigieren in der Hierarchie Ihres Webangebots. Mit `rel="next"` und `rel="prev"` können Sie in einer Serie von Seiten Verweise zum Vor- und Zurückblättern anbieten. Mit `rel="up"` verweisen Sie in Angeboten, die sich über mehrere logische Hierarchieebenen erstrecken, eine logische Ebene höher und `rel="top"` bietet sich zum Sprung auf die Projektstartseite an. Bei `rel="top"` scheiden sich allerdings die Geister – manche Anbieter nutzen diesen Link auch, um einfach auf der aktuell angezeigten Seite zum Anfang zurückzuspringen.

```
<link rel="alternate" href="en/index.htm" title="english">
```

Mit `rel="alternate"` können Sie auf eine andere Version der aktuellen Seite verweisen. Ideal geeignet ist dieser Verweistyp, wenn Sie ein mehrsprachiges Webangebot haben. In diesem Fall können Sie auf eine anderssprachige Version der gleichen Seite verweisen.

Es ist selbstverständlich erlaubt, logische Links bestimmter Typen mehrfach zu verwenden. Wenn Sie also beispielsweise auch französisch- und spanischsprachige Versionen der gleichen Seite anbieten, können Sie auch zwei oder drei Links der Sorte `rel="alternate"` anbieten. Wie ein Browser das an seiner Bedienoberfläche umsetzt, ist sein Problem, nicht Ihres. Ebenso wenig müssen Sie alle verfügbaren logischen Links einsetzen. Wenn Sie z. B. nur auf die Startseite und auf eine Sitemap-Seite verweisen wollen, spricht nichts dagegen.

4.4.4 Adressbasis für referenzierte Dateien

Eine weitere, mitunter nützliche Angabe im Kopf einer HTML-Datei ist die Adressbasis für referenzierte Dateien. Dazu ein Beispiel:

```
<head>
<base href="http://der.andere.server/bilder/">
  <!-- ... andere Angaben im Dateikopf ... -->
</head>
```

Mit dem base-Element können Sie zentral eine Adressbasis bestimmen. Wenn Sie nun im Dokument beispielsweise wie folgt eine Grafik referenzieren

```
<img src="bild001.gif" alt="Bild 1">
```

dann berücksichtigt der Browser die angegebene Adressbasis und fordert das Bild von folgender Adresse an: *http://der.andere.server/bilder/bild001.gif*.

Die zentral definierte Adressbasis gilt übrigens für alle im Dokument referenzierten Dateien.

4.5 Inhaltliche Strukturierung

Nachdem Sie nun eine Menge über Kopfdaten wissen, widmen wir uns dem sichtbaren Dateikörper des HTML-Dokuments, also dem, was im Browser-Fenster angezeigt wird. Es handelt sich um alles, was zwischen `<body>` und `</body>` notiert wird.

HTML stellt ein ganzes Arsenal an Elementen bereit, um Inhalte ordentlich zu strukturieren. Aufgrund historischer Entwicklungen sind einige dieser Elemente heute veraltet oder sie gehören noch zum Standard, werden jedoch in der Praxis nicht mehr verwendet. Wir werden uns in diesem Abschnitt daher auf eine Auswahl an solchen Elementen beschränken, die modernes HTML im Sinne von »semantischem Markup« repräsentieren und praxisrelevant sind.

Dabei sei noch mal ausdrücklich daran erinnert, dass diese Elemente nur der inhaltlichen Strukturierung dienen. Stören Sie sich beim Ausprobieren nicht an der Darstellung Ihrer Versuche im Browser. Die Feinheiten der Darstellung werden später mithilfe von CSS realisiert. Es ist sogar eine durchaus empfehlenswerte Vorgehensweise, bei inhaltslastigen Projekten nicht mit dem optischen Design zu beginnen, sondern mit dem logischen Markup in HTML. Wenn die Strukturierung der Inhalte stimmt, dann lässt sich die Struktur mithilfe von CSS auch in ein angemessenes und schickes Layout bringen.

4.5.1 Block-Elemente (Absatzformate) für den Fließtext

In HTML wird zwischen Block- und Inline-Elementen unterschieden. Aus Textverarbeitungsprogrammen sind Ihnen möglicherweise Begriffe wie Absatz- und Zeichenformate geläufig. Diese Begriffe können zumindest als Verständnishilfe dienen: Block-Elemente entsprechen Absatzformaten und Inline-Elemente entsprechen Zeichenformaten. Dass in HTML allerdings nicht von Absatz- und Zeichenformaten die Rede ist, sondern eben von Block- und Inline-Elementen, hängt mit der strengeren Strukturierungsphilosophie von HTML und dem Boxmodell der Ergänzungssprache CSS zusammen.

Ein Block-Element in HTML erstreckt sich über die gesamte verfügbare Breite und nimmt so viel Höhe ein wie vom Inhalt her erforderlich. »Verfügbare Breite« ist dabei relativ zu verstehen. Genauer werden wir darauf in Zusammenhang mit CSS eingehen.

Logische Bereiche

Wenn Webseiten aus logisch unterscheidbaren Bereichen bestehen, die später auch optisch als eigener Bereich erkennbar erscheinen sollen, dann sollten Sie diese Bereiche im Markup von vorneherein berücksichtigen. Ein Komplettlisting soll dies demonstrieren:

Listing 4.3: HTML-Datei mit Bereichen

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Bereiche</title>
</head>
<body>
<div id="Navigation">
   <a name="aktuelleSeite" id="aktuelleSeite">Startseite</a>
   <a class="Navi" href="produkte.html">Produkte</a>
   <a class="Navi" href="php/shop.php">Shop</a>
   <a class="Navi" href="unternehmen.html">Unternehmen</a>
   <a class="Navi" href="jobs.html">Jobs</a>
   <a class="Navi" href="impressum.html">Impressum</a>
   <a class="Navi" href="sitemap.html">Sitemap</a>
</div>
<div id="Inhalt">
   <div id="Seitenkopf">
      <h1>Willkommen!</h1>
   </div>
   <div id="Seitentext">
      <p>Für unsere Kunden bieten wir Produktinfos mit
      Direktbestellmöglichkeit (Shop) an. Unsere
      Geschäftspartner können sich über unser Unternehmen
      informieren. Besuchen Sie auch unsere Job-Seite!</p>
   </div>
</div>
</body>
</html>
```

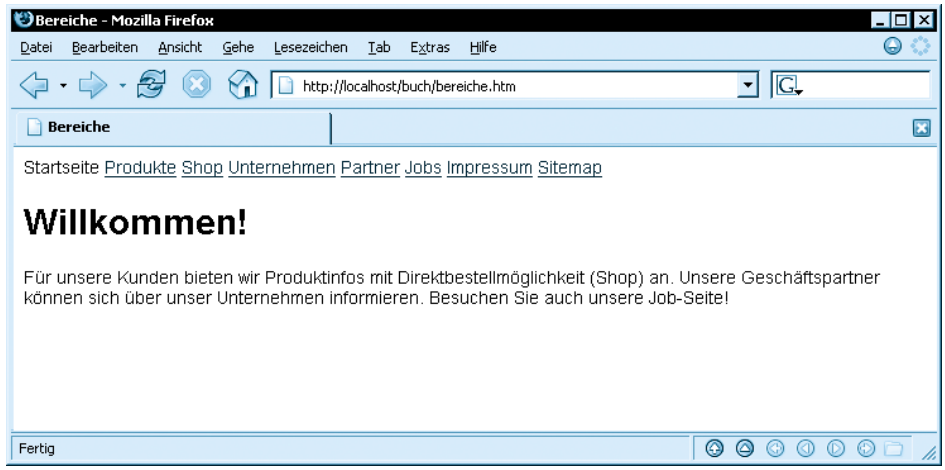


Abbildung 4.8: Die HTML-Datei aus Listing 4.3 im Browser

Die Darstellung der Seite im Browser ist noch sehr spartanisch. Aber dennoch enthält der HTML-Quelltext bereits eine Menge in Hinblick auf ein späteres Seitenlayout, indem darin logische sinnvolle Bereiche definiert sind. Auf der Hierarchieebene unterhalb des body-Elements sind zwei Bereiche definiert: einer für die Navigation, eingeleitet durch `<div id="Navigation">`, und einer für den Seiteninhalt (`<div id="Inhalt">`). Mit `<div>...</div>` werden in HTML Bereiche für logisch zusammenhängende Inhalte definiert. Das `div` steht für ***division*** (zu Deutsch: Abteilung, Bereich).

Das Attribut `id=` mit den frei wählbaren Namen ist nicht zwingend erforderlich. Doch da wir die Bereiche zu einem späteren Zeitpunkt mithilfe von CSS zum Errichten eines Seitenlayouts benötigen, setzen wir das Attribut jetzt bereits ein. Außerdem erhalten unsere Bereiche durch die Namenszuweisung gleich ihre logische Bedeutung, was das Verständnis des Quelltextes verbessert.

Das div-Element mit dem id-Namen `Inhalt` besteht im Beispiel selber wieder aus zwei untergeordneten Bereichen, die jeweils eigene Namen erhalten: `Seitenkopf` und `Seitentext`. Es ist also erlaubt und durchaus gängige Praxis, Bereiche ineinander zu verschachteln. Treiben Sie die Verschachtelung allerdings nicht auf die Spitze. Es gibt auch Webseiten, die sich »modern« schimpfen, im Grunde aber nur aus einer div-Orgie bestehen, die jeder logischen Nachvollziehbarkeit entbehrt. Auch in unserem Beispiel ist ein div-Bereich nicht wirklich zwingend logisch begründbar, nämlich der Bereich `Seitenkopf`. Da dieser nichts weiter als eine Überschrift enthält, könnte man ihn auch weglassen und z. B. der Überschrift selber den id-Namen `Seitenkopf` geben. Im Beispiel sähe das so aus:

```
<h1 id="Seitenkopf">Willkommen</h1>
```

Dennoch haben wir entschieden, die Überschrift in einen div-Bereich einzuschließen. Der Grund ist, dass wir zu einem späteren Zeitpunkt noch eine Grafikreferenz mit in diesen Bereich packen wollen, nämlich ein Logo.

Überschriften

HTML unterscheidet sechs Überschriftebenen. In der Praxis werden allerdings selten mehr als vier Ebenen benötigt und die Browser bieten auch meist nur für die ersten vier Ebenen sinnvolle Default-Formatierungen an.

Die Überschriftenebene ist an der Ziffer im einleitenden und abschließenden Überschriften-Tag erkennbar:

```
<h1>Überschrift 1. Ordnung</h1>
<h2>Überschrift 2. Ordnung</h2>
<h3>Überschrift 3. Ordnung</h3>
<h4>Überschrift 4. Ordnung</h4>
<h5>Überschrift 5. Ordnung</h5>
<h6>Überschrift 6. Ordnung</h6>
```

Alle Überschriftenelemente beginnen mit h für *heading* (zu Deutsch: Überschrift). Die Ziffer dahinter bezeichnet die Überschriftenebene. Achten Sie darauf, dass Sie beim Abschluss-Tag einer Überschrift stets die gleiche Ebenenziffer angeben wie beim Start-Tag.

Webbrowser verwenden für die Darstellung von Überschriften Default-Werte. Zweifellos entspricht diese kaum je den Wünschen, die Sie an die Darstellung haben: Ausrichtung, Schriftart, Schriftgröße, Farbe, Abstände, eventuell auch Schmuckrahmen, eigene Hintergrundfarbe usw. All das können Sie später mithilfe von CSS bewerkstelligen. An dieser Stelle wollen wir uns mit der Default-Darstellung begnügen.

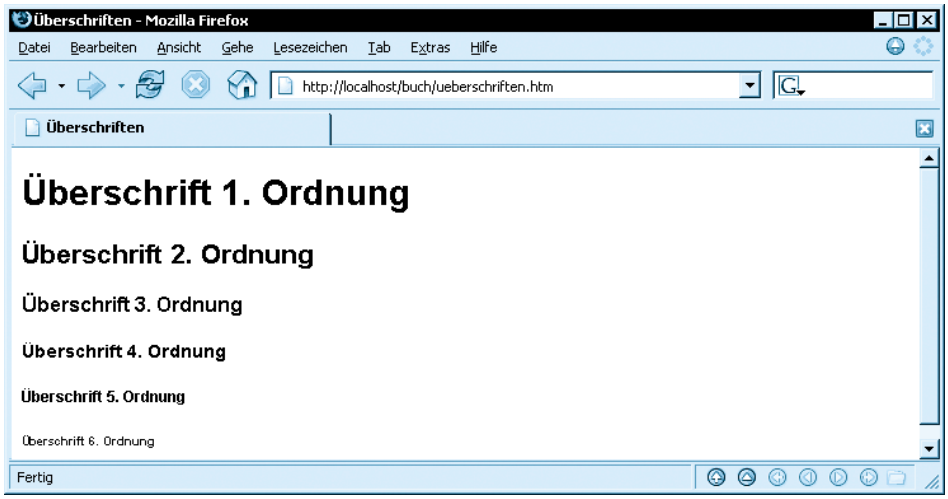


Abbildung 4.9: Default-Darstellung der sechs Überschriftenebenen im Browser

Absatzparagraphen

Um »Absatzschaltungen« in HTML zu erzeugen, müssen Sie das dafür vorgesehene Element verwenden. Schlampereien der Sorte »zweimal Enter eingeben und Leerzeile erzeugen« funktionieren nicht, da der Zeilenvorschub in HTML als Whitespace-Zeichen gilt und mehrere Whitespace-Zeichen einfach zu einem Leerzeichen zusammengefasst werden.

Gewöhnen Sie sich deshalb von vorneherein an, für Absätze in längerem Fließtext das p-Element zu benutzen:

```
<p>Hier der Text des Absatzes.</p>
<p>Und hier der nächste Absatz.</p>
```

Das p steht für *paragraph* (zu Deutsch: Absatz).

Wenn Sie wie empfohlen die strict-Variante von HTML verwenden, müssen übrigens alle Inhalte zwischen `<body>` und `</body>` in Block-Elemente eingeschlossen sein. Es ist also nicht erlaubt, zwischen `<body>` und `</body>` an irgendeiner Stelle direkt Text ohne umge-

bendes Block-Element zu notieren. Den Browsern sind solche Fehler zwar egal, doch die Validität des HTML-Dokuments geht durch solche Fehler unnötigerweise verloren.

Wenn Sie also sonst keine Strukturelemente im Dateikörper benötigen, beispielsweise, weil Sie eine reine Textdatei in HTML abbilden wollen, dann verwenden Sie dort, wo in der Textdatei Absätze durch Leerzeilen simuliert werden, das p-Element.

Für das p-Element wie für alle weiteren noch vorgestellten Elemente gilt wie für Überschriften: Die Default-Darstellung im Browser-Fenster können Sie später mit CSS beliebig ändern.

Aufzählungslisten

Darunter sind Listen zu verstehen, deren Listenpunkte mit einem so genannten ***Bullet-Symbol*** (mancherorts auch »Böller« genannt) beginnen. Die HTML-Spezifikation redet einfach nur von »unsortierten Listen« (im Gegensatz zu den nummerierten Listen). Gedacht sind die Aufzählungslisten für alle Arten von zusammengehörigen Daten, Argumenten oder dergleichen, wobei die Reihenfolge keine Rolle spielt.

Eine einfache Aufzählungsliste hat in HTML folgendes Aussehen:

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

Eingeschlossen wird die Liste in `<ul>…</ul>`. Das steht für ***unordered list*** (zu Deutsch: unsortierte Liste). Jeder einzelne Listenpunkt wird durch `<li>...</li>` markiert – die Abkürzung bedeutet ***list item*** (zu Deutsch: Listeneintrag). Das ul-Element darf nichts anderes als li-Elemente enthalten, also keinen nackten Text oder andere Elemente außerhalb der li-Elemente. Die li-Elemente dürfen dagegen Text, andere Block-Elemente und andere Inline-Elemente enthalten.

Listen können auch verschachtelt werden, wie die nachfolgende Abbildung zeigt.



Abbildung 4.10: Verschachtelte Aufzählungsliste im Browser

Der HTML-Quelltext dazu sieht folgendermaßen aus:

```
<ul>
   <li>Listenpunkt</li>
   <li>Unterliste:
      <ul>
         <li>Listenpunkt</li>
         <li>Listenpunkt</li>
      </ul>
   </li>
</ul>
```

Zu beachten ist dabei, dass die untergeordnete Liste als normaler Listenpunkt mit führendem Text beginnt und daran anschließend die komplette untergeordnete Liste notiert wird. Am Ende wird der Listenpunkt, der die untergeordnete Liste enthält, ordnungsgemäß mit `</li>` abgeschlossen.

Bei verschachtelten Listen wechseln die Browser üblicherweise das Bullet-Symbol. Um Einfluss darauf zu nehmen, welches Symbol angezeigt wird, müssen Sie CSS verwenden.

Nummerierte Listen

Nummerierte Listen ähneln vom syntaktischen Aufbau her den Aufzählungslisten. Der Unterschied besteht nur darin, dass sie mit `<ol>...</ol>` definiert werden, wobei `ol` für *ordered list* (zu Deutsch: geordnete Liste) steht. Die folgende Beispielnotation bewirkt eine einfache Nummerierung:

```
<ol>
   <li>Aufstehen</li>
   <li>Duschen</li>
   <li>Frühstücken</li>
</ol>
```

Aufgelöst wird dies im Browser zu:

1. Aufstehen
2. Duschen
3. Frühstücken

Das Verschachteln von nummerierten Listen ist genauso möglich wie bei Aufzählungslisten. Die Hoffnung, dass dadurch automatisch Nummerierungen wie 1.1, 1.2 usw. entstehen, wird jedoch enttäuscht. Solche Automatisierungsmechanismen unterstützt HTML leider nicht und nur durch spezielle CSS-Notationen lässt sich etwas Derartiges bewerkstelligen. In ***Abschnitt 6.2*** gehen wir darauf ein. Ebenfalls mit CSS lässt sich die Art der Nummerierung beeinflussen (z.B. alphabetische Nummerierung anstelle der numerischen).

Blöcke mit präformatiertem Text

Der Ausdruck »präformatierter Text« ist etwas gewöhnungsbedürftig. Gemeint ist damit, dass Text im Browser so wiedergegeben wird wie im Editor eingegeben, mit allen Leerzeichen, Zeilenumbrüchen, Einrückungen usw. Dieses Verhalten entspricht ja nicht den normalen Editierregeln von HTML. Manchmal ist ein solches Verhalten jedoch sehr wünschenswert, vor allem, wenn man in HTML-Texten Blöcke mit Programmlistings oder rein mit ASCII-Zeichen formatierte Tabellen oder Kunstwerke (»ASCII-Art«) im Text darstellen möchte.

In HTML können Sie solche Blöcke mit `<pre>...</pre>` auszeichnen. Dabei steht `pre` für *preformatted* (präformatiert). Alles, was Sie zwischen diesen beiden Tags eingeben, wird so interpretiert wie im Editor eingegeben. Beachten Sie, dass der Browser dabei eine dicktengleiche Schriftart wie z.B. Courier verwendet, also eine Schriftart, bei der alle Zeichen die gleiche Breite haben. Denn nur so lassen sich viele bei präformatiertem Text gewünschte Effekte überhaupt sinnvoll darstellen. Ein Beispiel:

```
<pre>
       /-----------------------------------------------\
       |          |      Average      |  other   | Misc  |
       |          |-------------------| category |-------|
       |          |  height |  weight |          |       |
       |-----------------------------------------|-------|
       | males    | 1.9     | 0.003   |          |       |
       |-----------------------------------------|-------|
       | females  | 1.7     | 0.002   |          |       |
       \-----------------------------------------------/
</pre>
```

Diese mit Strichen und positioniertem Text »gemalte« Tabelle können Sie in HTML mithilfe des pre-Elements so wie sie ist im Browser repräsentieren.

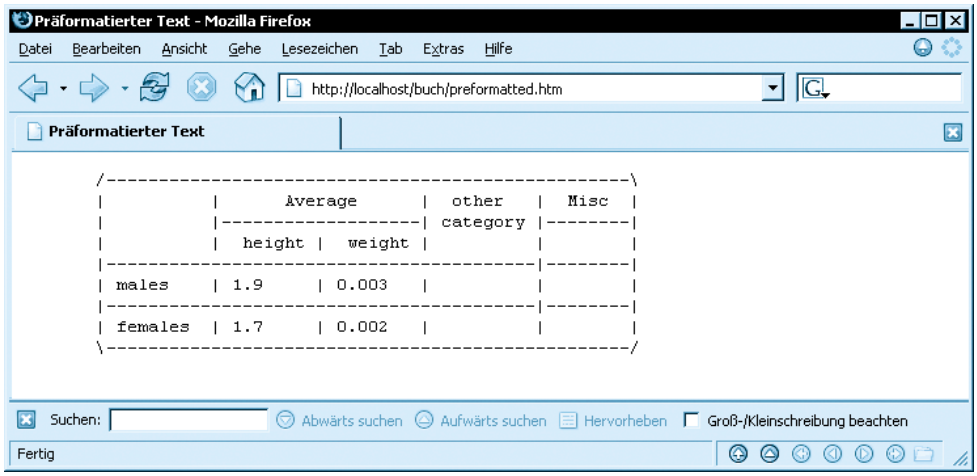


Abbildung 4.11: »Gemalte« Tabelle als präformatierter Text im Browser

Das pre-Element darf durchaus auch andere HTML-Elemente enthalten. Nicht selten wird das pre-Element auch dazu benutzt, um HTML-Quelltext im Browser darzustellen. Dabei müssen Sie jedoch wie bereits gelernt penibel darauf achten, dass Sie alle HTML-eigenen Zeichen maskieren. Andernfalls würde der Browser den HTML-Code interpretieren statt ihn darzustellen. Wir zeigen das Ganze am Beispiel der zuvor behandelten nummerierten Listen:

```
<pre>
&lt;ol&gt;
   &lt;li&gt;Aufstehen&lt;/li&gt;
   &lt;li&gt;Duschen&lt;/li&gt;
   &lt;li&gt;Frühstücken&lt;/li&gt;
&lt;/ol&gt;
</pre>
```

Im Browser wird dann der HTML-Quelltext der nummerierten Liste angezeigt, mit allen Einrückungen und Zeilenumbrüchen wie eingegeben.

Sonstige Block-Elemente für den Fließtext

Ergänzend zu den Standard-Blockelementen für Überschriften, Textabsätze, Listen und präformatierten Text bietet HTML noch ein paar weitere Block-Elemente zur Textgestaltung an, die bei konsequenter Verwendung zu einem reicheren semantischen Markup beitragen: Durch `<blockquote>...</blockquote>` markieren Sie ein Zitat, das als eigener Absatz dargestellt werden soll. Die meisten Browser stellen den Text eines so ausgezeichneten Inhalts eingerückt dar, was viele Möchtegern-Designer dazu verleitet hat, das blockquote-Element dazu zu verwenden, um Einrückungen jeder Art im Text zu realisieren. Solche Zweckentfremdungen, die auf eher zufälligen Default-Darstellungsweisen von Browsern beruhen, gehören aus heutiger Sicht zu den schlimmsten Verbrechen an HTML, die man begehen kann. Hüten Sie sich also davor, mit dem blockquote-Element irgendetwas anderes auszuzeichnen als ein alleinstehendes Zitat, und rücken Sie Texte mithilfe von CSS ein!

Falls Sie im HTML-Quelltext auch die Quelle eines Zitats nennen möchten, so steht dafür das Attribut `cite` zur Verfügung. Ein Beispiel:

```
<blockquote cite="Cluetrain Manifest, http://www.cluetrain.de/, These 7"
title="aus dem Cluetrain Manifest, http://www.cluetrain.de/">
    Hyperlinks untergraben Hierarchien.
</blockquote>
```

Das cite-Attribut hat jedoch keine sichtbare Wirkung im Browser. Deshalb haben wir im Beispiel außerdem noch das title-Attribut notiert, damit der Anwender die Quelle des Zitats erfährt, wenn er mit der Maus über den Text fährt.

Ein weiteres Block-Element stellt HTML mit <address>...</address> zur Verfügung. Dieses Element ist zur Auszeichnung von Autorenangaben mit Kontaktmöglichkeit gedacht, z. B.:

```
<address>
    Stefan Münz, stefan.muenz@selfhtml.org
</address>
```

Drei weitere Block-Elemente sind schließlich für den Texttyp »Glossar« vorgesehen. Ein Glossar ist eine Art Liste, deren Einträge jeweils aus einem Glossareintrag (»Term«) und einer Definition bestehen. In HTML sieht das wie im folgenden Beispiel aus:

```
<dl>
   <dt>dl-Element</dt>
   <dd>dient zur Markierung einer Glossarliste</dd>
   <dt>dt-Element</dt>
   <dd>dient zur Markierung eines Terms im Glossar</dd>
   <dt>dd-Element</dt>
   <dd>dient zur Markierung einer Definition im Glossar</dd>
</dl>
```

Das Beispiel erklärt sich bereits selbst: Mit dem dl-Element schließen Sie die gesamte Glossar-Liste ein. Innerhalb davon werden abwechselnd je ein dt- und ein dd-Element notiert. Das dt-Element markiert einen Glossareintrag und das anschließende dd-Element die zugehörige Definition.

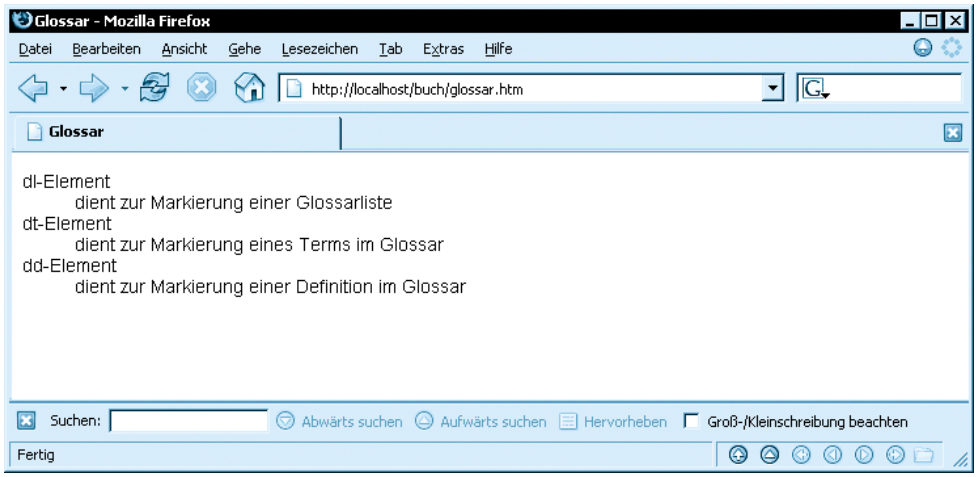


Abbildung 4.12: Glossar-Liste im Browser

Erwähnt werden soll abschließend noch das hr-Element: Durch Notieren von <hr> (in XHTML: <hr />) fügen Sie eine horizontale Trennlinie ein. Dies kann sinnvoll sein, um logische Textbereiche voneinander zu trennen.

4.5.2 Inline-Elemente (Zeichenformate) für den Fließtext

Als Inline-Elemente werden in HTML Elemente bezeichnet, die keine neue Zeile im Textfluss bewirken und deren Breite sich nur über den Raum erstreckt, den der Inhalt einnimmt. In der hier empfohlenen strict-Variante von HTML dürfen Inline-Elemente ebenso wenig wie nackter Text direkt im Dateikörper zwischen <body> und </body> vorkommen, sondern nur innerhalb von Block-Elementen oder bestimmten anderen Inline-Elementen.

Erzwungener Zeilenumbruch

Genaugenommen ist der erzwungene Zeilenumbruch weder ein Inline-Element noch ein Block-Element, weil er gar keinen eigenen Inhalt und Erstreckungsbereich hat. Wir behandeln ihn jedoch an dieser Stelle, da er typischerweise innerhalb von Block-Elementen vorkommt, um dort Zeilenumbrüche an gewünschten Stellen zu bewirken. Ein typischer Anwendungsfall sind Strophen eines Gedichts oder Songs, wie das nachfolgende Beispiel zeigt:

```
<p>
Yesterday<br>
All my troubles seemed so far away<br>
Now it looks as though they're here to stay<br>
Oh I believe in yesterday...
</p>
```

Das alleinstehende Tag <br> bewirkt einen einfachen Zeilenumbruch. Falls Sie in XHTML schreiben, dürfen Sie nicht vergessen, dass dort <br /> notiert werden muss.

Inline-Auszeichnungen mit semantischer Bedeutung

Mit der Zielsetzung, möglichst sinnvolles semantisches Markup zu erzeugen, sollten Sie die nachfolgenden Inline-Auszeichnungen bevorzugt benutzen.

Inline-Auszeichnung	Bedeutung
...	Betont, hervorgehoben
...	Noch stärker betont, besonders hervorgehoben
<code>...</code>	Quelltext-Fragment, z. B. aus Programmiersprachen
<samp>...</samp>	Beispiel, z. B. bei Ausgaben eines Programms oder Scripts
<kbd>...</kbd>	Tastatureingaben des Anwenders
<var>...</var>	Variable, verwendbar z. B. auch für Namen
<cite>...</cite>	Zitat
<q cite="">...</q>	Quotation mit Zitathinweis
<abbr>...</abbr>	Als Buchstabenfolge gesprochene Abkürzungen
<acronym>...</acronym>	Als Wort gesprochene Abkürzungen

Tabelle 4.7: Semantische Inline-Auszeichnungen in HTML

Einige dieser Elemente sind vor allem für Softwaredokumentation interessant, wie etwa die Elemente `code`, `samp`, `kbd` und `var`. Etwas Interpretationsfreiheit ist jedoch erlaubt. Es spricht nichts dagegen, wenn Sie beispielsweise mit dem var-Element unverlinkte URL-Adressen auszeichnen oder auch Produktbezeichnungen. Wichtig ist, dass Sie solche Auszeichnungen durchgängig gleich und in allen Texten eines Projekts konsequent verwenden.

Etwas erklärungsbedürftig sind die Elemente `abbr` und `acronym`. Unter `abbr` fallen Abkürzungen wie »WWW«, »HTML« oder »FBI«, also Abkürzungen, die sich nur Buchstabe für Buchstabe aussprechen lassen. Unter `acronym` fallen dagegen Ausdrücke wie »NASA« oder »Modem«, also als Wort gesprochene Ausdrücke, die jedoch eigentlich eine Abkürzung darstellen.

Wie ein Browser solche semantischen Auszeichnungen im Text darstellt, bleibt ihm überlassen. Die meisten modernen grafischen Browser stellen beispielsweise Inhalte, die mit `<em>...</em>` ausgezeichnet sind, in Kursivschrift dar, `<strong>...</strong>` dagegen in Fettschrift, `<dfn>...</dfn>` oftmals gepunktet unterstrichen und `<code>...</code>` in dicktengleicher Schrift. Verlassen sollten Sie sich jedoch in dieser Hinsicht auf nichts. Semantische Elemente schreien geradezu danach, mithilfe von CSS ordentlich formatiert zu werden.

Nachfolgend noch ein Beispiel, das den Einsatz semantischer Inline-Auszeichnungen zeigt:

```
<p><q cite="Heraklit">Alles fließt</q>, sprach <var
class="person_name">Heraklit</var>, für den die <dfn>archê</dfn>, also der
Urgrund allen Seins, in der Bewegung des ewig Werdenden lag. Genaugenommen
stammt der Leitspruch, dass alles fließt, zwar <em>nicht</em> von <var
class="person_name">Heraklit</var>, sondern wird ihm nur von <var
class="person_name">Platon</var> zugeschrieben - doch es drückt seine Lehre so
gut aus, dass man über diese kleine Ungenauigkeit gerne hinweg sieht.</p>
```

Das Beispiel zeigt, wie Sie mithilfe des universell einsetzbaren `class`-Attributs noch weitere semantische Genauigkeit im Text erzielen können.

Inline-Auszeichnung ohne semantische Bedeutung

Bei diesen Elementen handelt sich um solche, die direkt die Textformatierung beeinflussen.

Inline-Auszeichnung	Bedeutung
`<i>...</i>`	Kursiv
`<b>...</b>`	Fett
`<tt>...</tt>`	Dicktengleich
`<big>...</big>`	Größer als normal
`<small>...</small>`	Kleiner als normal

Tabelle 4.8: Inline-Auszeichnung ohne semantische Bedeutung

Inline-Auszeichnung	Bedeutung
_{...}	Tiefgestellt
^{...}	Hochgestellt
...	Inline-Element ohne Wirkung

Tabelle 4.8: Inline-Auszeichnung ohne semantische Bedeutung (Fortsetzung)

Die Tabelle enthält ausschließlich Elemente, die im strict-Standard von HTML vorkommen. Dennoch gelten vor allem die beiden erstgenannten Elemente, also das i- und das b-Element, bei Hardlinern des semantischen Markup als verpönt, weil sie direkte Formatieranweisungen darstellen. Bei Elementen wie dem tt-Element, das einfach nur die Verwendung dicktengleicher Schrift anweist, sind die Ansichten bereits geteilter, da seine mögliche Einsatzzwecke nicht unbedingt durch das verfügbare Repertoire an semantischen Inline-Auszeichnungen abgedeckt werden kann. Und für Elemente wie `sub` und `sup` gibt es ohnehin keine semantische Entsprechung.

Besondere Beachtung verdient das span-Element. Es bewirkt optisch rein gar nichts, wird aber vom Parser des Browsers registriert. Das span-Element eignet sich durch diese Eigenschaft besonders zur Formatierung mit CSS.

Auszeichnungen für Änderungsmarkierungen

Möglicherweise kennen Sie aus Textverarbeitungsprogrammen die Möglichkeit, Dokumente beim Bearbeiten mit Änderungsmarkierungen zu versehen. Dabei werden gelöschte Textpassagen meist rot und durchgestrichen dargestellt, eingefügte dagegen grün. Gedacht sind solche Markierungen für den Austausch etwa für Review-Prozesse oder Lektorat. In HTML stehen diese Funktionen ebenfalls zur Verfügung:

Mit <ins>...</ins> markieren Sie Textpassagen als neu eingefügt – `ins` steht für ***insert*** (zu Deutsch: einfügen) – und mit ~~...~~ können Sie Passagen zum Löschen markieren – `del` steht für ***delete*** (zu Deutsch: löschen). Mithilfe zweier Attribute können Sie Auszeichnungen dieser Art außerdem datieren und begründen. Dazu ein Beispiel:

```
<ins datetime="2005-06-25T11:00+01:00" cite="http://www.w3.org/TR/html4/
appendix/changes.html#19991224">Die Meta-Angabe für Refreshs wird von der HTML-
Spezifikation mittlerweile nicht mehr empfohlen. Stattdessen wird auf die
Möglichkeit hingewiesen, serverseitige Redirects einzusetzen.</ins>
```

Das datetime-Attribut erwartet eine Datum-Zeit-Angabe in der dargestellten Form (der Wert im Beispiel bedeutet: 25. Juni 2005, 11:00 Uhr bei plus 1 Stunde gegenüber Greenwich-Zeit). Die beim cite-Attribut angegebene URL-Adresse verweist auf eine Stelle in der HTML-4.01-Spezifikation, in der Änderungen gegenüber der älteren HTML-4.0-Spezifikation aufgelistet werden.

Die Elemente `ins` und `del` weisen übrigens eine Besonderheit auf: Obwohl sie selbst zu den Inline-Elementen gehören, dürfen Sie sowohl Block- als auch weitere Inline-Elemente enthalten. Ein Konstrukt wie das Folgende ist also durchaus zulässig:

```
<ins>
   <h3>Eine eingefügte Überschrift</h3>
   <p>und ein eingefügter Absatz</p>
</ins>
```

4.5.3 Tabellen

HTML unterstützt alle wichtigen Features, um tabellarische Informationen auszuzeichnen. Tabellen haben in HTML freilich eine lange und bewegte Geschichte. Viel zu lange hatte das W3-Konsortium gar keine Tabellenauszeichnungen in HTML vorgesehen. Seit sich jedoch 1995 der revolutionäre Browser Netscape 1.1 über den damals aktuellen HTML-Standard 2.0 hinwegsetzte und erstmals Tabellen in HTML unterstützte, tat sich für all jene, die sich endlich mehr Freiheiten bei der Gestaltung von Webseiten wünschten, eine neue Tür auf. Vor allem die rahmenlosen Tabellen erlaubten es, Inhalte wie Navigation und Information attraktiv nebeneinander zu positionieren. Die so genannten Tabellenlayouts waren geboren. Ganze Webdesigner-Generationen lernten, ins HTML-Grundgerüst gleich mal ein paar verschachtelte Tabellen einzubauen, um so das gewünschte Basislayout zu bestimmen. Es gab sogar renommierte WYSIWYG-Editoren, die am Bildschirm ein Zentimeterraster darstellten, in dem der Designer seine Webseiteninhalte frei positionieren konnte. Intern wurde das Raster dann als komplexe Tabelle aufgelöst. Der dabei entstehende HTML-Code war für Menschen kaum noch genießbar und hatte mit der ursprünglichen Idee von HTML nicht mehr viel zu tun.

Aus moderner Sicht sind solche Art von Tabellenlayouts längst verpönt. Es spricht zwar nichts dagegen, für nebeneinander darzustellende Informationen auch mal eine rahmenlose Tabelle zu verwenden, etwa, wenn ein Bild mit nebenstehendem Text dargestellt werden soll, oder ein Formular mit ordentlich untereinander stehenden Feldern und unterschiedlich langen Feldbeschriftungen davor. Doch Tabellen wie selbstverständlich als Basis für die grundsätzliche Informationsverteilung auf einer Seite einzusetzen, ist keine saubere Praxis mehr. Dazu gibt es mittlerweile in CSS die Möglichkeit, beliebige Elemente und Bereiche wie gewünscht zu positionieren und zueinander anzuordnen. Auf die Details dazu werden wir in Zusammenhang mit CSS eingehen.

Gewöhnen Sie sich also an, Tabellen nur für tabellarische Informationen zu verwenden und rahmenlose Tabellen für nebeneinander darzustellende Informationen nur für sinnvolle Einzelfälle innerhalb einer Webseite, jedoch nicht als Basis für das gesamte Seitenlayout.

Aufbau einer Tabelle

Das nachfolgende Quelltextbeispiel zeigt eine kleine Tabelle:

```
<table border="1">
   <caption>Vergleich von Äpfel und Birnen</caption>
   <thead>
      <tr>
         <th>Äpfel</th>
         <th>Birnen</th>
```

```
        </tr>
      </thead>
      <tfoot>
        <tr>
          <td>beliebter</td>
          <td>weniger beliebt</td>
        </tr>
      </tfoot>
      <tbody>
        <tr>
          <td>12 Einheiten Vitamin C</td>
          <td>5 Einheiten Vitamin C</td>
        </tr><tr>
          <td>regt die Verdauung an</td>
          <td>wirkt entwässernd</td>
        </tr>
      </tbody>
  </table>
```

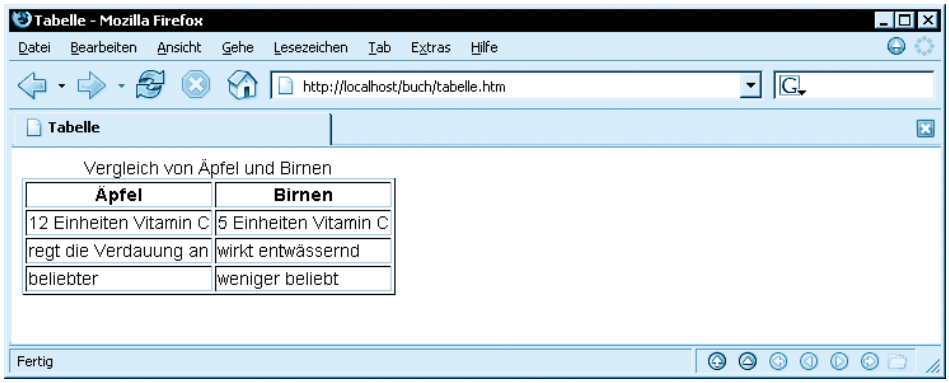


Abbildung 4.13: Einfache HTML-Tabelle im Browser

Eine vollständige Tabelle in HTML besteht aus einer Tabellenüberschrift, einem Tabellenkopf, einem Tabellenkörper und einem Tabellenfuß. Die gesamte Tabelle wird in die Tags `<table>` bzw. `</table>` eingeschlossen. Soll die Tabelle einen sichtbaren Rahmen und ein sichtbares Gitternetz haben, müssen Sie im einleitenden Tag das Attribut `border="1"` notieren (die Zahl steht für die Rahmendicke in Pixel; für dickere Rahmen können Sie also auch einen höheren Wert angeben). Fehlt dieses Attribut, bleiben alle Ränder der Tabelle unsichtbar. Die etwas klobig wirkende Default-Darstellung von Rahmen und Gitternetz im Browser, wie in der Abbildung zu sehen, können Sie später mithilfe von CSS beeinflussen.

Betrachten wir nun den internen Aufbau der Tabelle. Die Tabellenüberschrift, markiert durch `<caption>...</caption>`, muss direkt innerhalb von `<table>...</table>` notiert werden, am besten direkt nach dem einleitenden `<table>`-Tag. Die Tabellenüberschrift kann Text und andere Inline-Elemente enthalten. Selbstverständlich lässt sie sich später mithilfe von CSS formatieren und auch im Verhältnis zur Tabelle genau ausrichten und positionieren.

Die drei Bereiche für Tabellenkopf, Tabellenfuß und Tabellenkörper werden durch <thead>...</thead>, <tfoot>...</tfoot> und <tbody>...</tbody> markiert. Beachtenswert ist die Reihenfolge der Notation von Tabellenkopf, Tabellenfuß und Tabellenkörper in eben dieser Reihenfolge. Obwohl die Fußdaten am Ende der Tabelle erscheinen, müssen sie vor den Daten des Tabellenkörpers notiert werden!

Falls Sie eine Tabelle erstellen möchten, dabei aber weder eine Tabellenüberschrift wünschen noch explizit zwischen Tabellenkopf, Tabellenfuß und Tabellenkörper unterscheiden möchten, können Sie beruhigt sein: Es ist durchaus erlaubt, all diese Elemente wegzulassen und nur die Elemente zum Definieren von Reihen und darin enthaltenen Zellen zu notieren. Bedenken Sie jedoch, dass vor allem im Fall von komplexen Tabellen, bei denen sich z.B. bereits der Kopfbereich über mehrere Zeilen erstreckt, die explizite Unterscheidung der Tabellenteile im Markup durchaus sinnvoll sein kann. Immerhin können Sie auch für diese Elemente eigene CSS-Formatierungen definieren. Noch wichtiger aber sind die entsprechenden Auszeichnungen in Hinblick auf eine rein akustische Ausgabe der Webseite, wie etwa stark Sehbehinderte sie zum Surfen im Web benutzen. Weiter unten werden wir noch genauer auf diese Problematik eingehen. Die Tabelle aus dem obigen Beispiel darf jedenfalls auch wie folgt aussehen und wäre dennoch korrektes HTML:

```
<table border="1">
   <tr>
      <th>Äpfel</th>
      <th>Birnen</th>
   </tr><tr>
      <td>12 Einheiten Vitamin C</td>
      <td>5 Einheiten Vitamin C</td>
   </tr><tr>
      <td>regt die Verdauung an</td>
      <td>wirkt entwässernd</td>
   </tr><tr>
      <td>beliebter</td>
      <td>weniger beliebt</td>
   </tr>
</table>
```

In dieser Variante der Tabelle fehlen die Auszeichnungen für Tabellenüberschrift, Tabellenkopf, Tabellenkörper und Tabellenfuß. Die semantische Information zum Tabellenfuß geht dabei komplett verloren. Die Tabellenzeile, die in der Variante zuvor als Tabellenfuß ausgezeichnet wurde, wird nun einfach unten als letzte Zeile notiert. Die Kopfinformationen sind dagegen erhalten geblieben, wofür das th-Element verantwortlich ist.

Doch der Reihe nach: Tabelleninhalte werden in HTML Tabellenzeile für Tabellenzeile notiert. Jede Tabellenzeile wird durch <tr> eingeleitet und endet mit </tr> – tr steht für ***table row*** (zu Deutsch: Tabellenzeile). Zwischen <tr> und </tr> werden jeweils die Spalten in der betreffenden Zeile notiert, also die Tabellenzellen dieser Tabellenzeile. Es gibt zwei Arten, nämlich Kopf- und Datenzellen. Kopfzellen werden durch <th>...</th> markiert und Datenzellen durch <td>...</td>. Dabei steht th für ***table head*** (zu Deutsch: Tabellenkopf) und td für ***table data*** (zu Deutsch: Tabellendaten).

Mit th ausgezeichnete Zellen werden in den meisten Browsern optisch anders dargestellt als td-Zellen – in der Regel erscheinen Kopfzellen zentriert und in Fettschrift, während Datenzellen linksbündig und mit normalem Schriftgewicht dargestellt werden. Auch hier gilt wieder: Stören Sie sich nicht an den Default-Formatierungen. Mithilfe von CSS werden Sie später alle Zellen, Zellen eines bestimmten Typs oder einzelne Zellen nach Wunsch formatieren, ausrichten usw. können.

In unserem einfachen Beispiel enthalten die einzelnen Tabellenzellen für Kopfdaten und Tabellendaten nichts als kurzen Textinhalt. HTML-Tabellen sind jedoch sehr mächtig. Jede einzelne Tabellenzelle darf neben reinem Text auch beliebige Block- und Inline-Elemente enthalten, also strukturierte Bereiche, Überschriften, Listen und Textabsätze, Bild- und Multimediareferenzen und sogar komplette weitere Tabellen!

Erscheinungsbild von Außenrahmen und Gitternetz

Wenn Sie beispielsweise ein »offenes« Gitternetz in Ihrer Tabelle wünschen, also ohne sichtbaren Außenrahmen, oder beispielsweise nur sichtbare Linien zwischen den Tabellenspalten, nicht aber zwischen den Zeilen, so stehen hierfür eine Reihe von HTML-Attributen zur Verfügung.

Um für eine Tabelle ein offenes Gitternetz ohne sichtbaren Außenrahmen zu definieren, notieren Sie im einleitenden <table>-Tag:

```
<table border="1" frame="void">
```

Durch das Attribut frame= lässt sich eine Regel definieren, wie der Außenrahmen dargestellt werden soll. Mit der Wertzuweisung void wird der Außenrahmen komplett unterdrückt. Möglich sind aber auch die Wertzuweisungen above (Rahmen nur oben), below (Rahmen nur unten), hsides (Rahmen nur oben und unten), vsides (Rahmen nur links und rechts), lhs (Rahmen nur links) und rhs (Rahmen nur rechts).

Um nur zwischen den Spalten der Tabelle sichtbare Linien darzustellen, nicht aber zwischen den Zeilen, können Sie notieren:

```
<table border="1" frame="void" rules="cols">
```

Um zu definieren, welche Linien des Gitternetzes angezeigt werden sollen, dient das Attribut rules= im einleitenden <table>-Tag. Mit der Wertzuweisung cols bestimmen Sie, dass nur Linien zwischen Spalten angezeigt werden. Umgekehrt erreichen Sie mit dem Wert rows, dass nur Linien zwischen den Tabellenzeilen, nicht aber zwischen den Spalten angezeigt werden. Wenn Sie das Markup für Tabellenkopf, Tabellenfuß und Tabellenkörper vollständig notiert haben, ist außerdem noch der Wert groups zulässig. Damit werden nur waagerechte Linien zwischen den Teilen der Tabelle angezeigt.

Die Abbildung zeigt, wie sich eine solche Beeinflussung der Liniendarstellung auswirkt. In diesem Fall haben wir folgende Angaben notiert: <table border="1" frame="void" rules="groups">. Die Darstellung im Browser hat nun das gewünschte Grundaussehen.

Feinheiten der Formatierung wie Abstände innerhalb von Zellen usw. lassen sich nachträglich mit CSS realisieren.

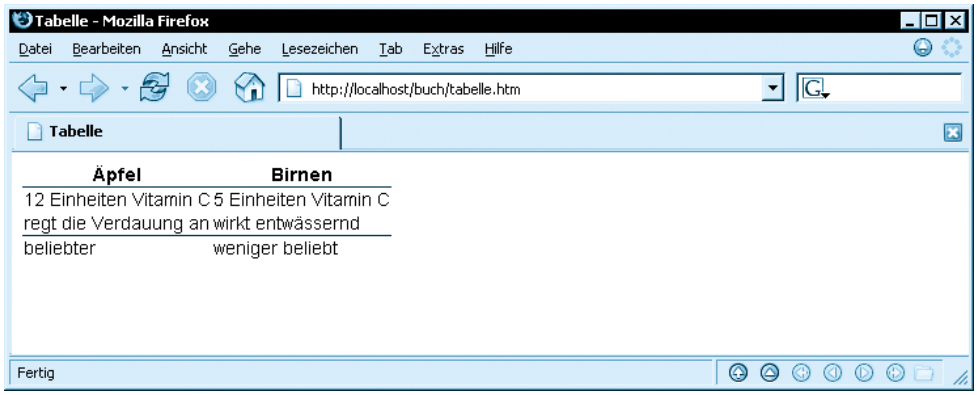


Abbildung 4.14: Tabelle ohne Außenrahmen und mit Linien nur zwischen Tabellenteilen

Tabellenbreite und Spalten vordefinieren

Wenn umfangreiche Tabellen über eine schwache Netzverbindung übertragen werden müssen, hat der Browser das Problem, dass er mit der Darstellung der Tabelle am Bildschirm warten muss, bis die gesamte Tabelle übertragen ist, da er vorher nicht weiß, welche Darstellungsbreite er für die Tabelle reservieren muss. Tabellen werden nämlich per Voreinstellung so breit dargestellt, wie es ihr Inhalt verlangt. Das Gleiche gilt für die einzelnen Spalten der Tabelle. Jede Spalte wird so breit, wie es ihr breitester, nicht umzubrechender Inhalt verlangt. Um dem Browser bei diesem Problem zu helfen, können Sie in HTML die Breiten von Tabelle und Spalten vordefinieren. Außerdem können Sie Spalten in Spaltengruppen aufteilen und das Breitenverhältnis der Spalten untereinander bestimmen.

Die gewünschte Gesamtbreite der Tabelle können Sie durch das width-Attribut im einleitenden `<table>`-Tag angeben. Ein Beispiel:

```
<table border="1" width="100%">
```

Erlaubt sind als Wertzuweisung reine Zahlen, die als Pixel interpretiert werden, oder Prozentangaben, wie im Beispiel gezeigt. 100% bedeutet: Die Tabelle soll die gesamte verfügbare Breite einnehmen. Bei einer solchen Angabe dehnt der Browser die Spalten nach Gutdünken aus. Wenn Sie mehr Kontrolle über die Spaltenbreiten haben möchten, ist es besser, zusätzlich zur width-Angabe im `<table>`-Tag Spalten vorzudefinieren und ihnen eine Breite zuzuweisen. Dazu das Beispiel einer kompletten kleinen Tabelle:

```
<table border="1" width="100%">
     <colgroup>
        <col width="2*">
        <col width="4*">
        <col width="3*">
```

```
    </colgroup>
    <tr>
      <th>Name</th>
      <th>Bedeutung</th>
      <th>Bemerkung</th>
    </tr>
    <tr>
      <td><code>getElementById</code></td>
      <td>Zugriff auf ein bestimmtes Element durch id-Namen</td>
      <td>DOM-1-Level</td>
    </tr>
</table>
```

Um Spalten vorzudefinieren, notieren Sie unterhalb des einleitenden `<table>`-Tags ein Konstrukt, das durch `<colgroup>` eingeleitet und mit `</colgroup>` beendet wird. Innerhalb davon können Sie für jede Spalte jeweils ein alleinstehendes `<col>`-Tag definieren. Mit dem Attribut `width=` definieren Sie die Spaltenbreite. Auch hierbei sind absolute Zahlen für Pixel und Prozentangaben erlaubt. Als Besonderheit ist jedoch auch eine weitere Notationsform zulässig, wie das Beispiel zeigt. Durch Angabe einer Zahl mit Sternzeichen dahinter können Sie das logische Breitenverhältnis der Spalten untereinander festlegen. Im obigen Beispiel nimmt die erste Spalte 2/9 Breite der Gesamttabellenbreite ein, die zweite Spalte 4/9 und die dritte Spalte 3/9 (also 1/3).

Wie Ihnen vielleicht aufgefallen ist, haben wir in unseren Beispielen ausschließlich relative Angaben verwendet. Relative Angaben sollten Sie gegenüber absoluten Pixelangaben bevorzugen. Zu absoluten Angaben sollten Sie auch wissen, dass Browser solche Angaben nicht zwingend umsetzen. Jede Tabelle oder Spalte wird so breit, wie es ihr breitester Inhalt verlangt. Zu geringe Breitenangaben werden dabei ignoriert. Die Möglichkeit, überbreite Inhalte abzuschneiden oder Spaltenbreiten wirklich festzuzementieren, besteht nur mithilfe von CSS.

Zellen über mehrere Spalten oder Zeilen verbinden

Bei komplexeren Tabellen kommt es häufig vor, dass einzelne Zellen über mehrere Spalten oder Zeilen hinweg verbunden werden. Auch das ist mit HTML möglich. Das erste Code-Beispiel zeigt eine Zelle, die sich innerhalb einer Tabellenzeile über zwei Spalten erstreckt:

```
<table border="1">
   <tr>
      <td colspan="2">A+B</td>
      <td>C</td>
   </tr>
   <tr>
      <td>A</td>
      <td>B</td>
      <td>C</td>
   </tr>
</table>
```

Durch das Attribut `colspan=` können Sie eine Tabellenzelle, in deren Start-Tag Sie das Attribut notieren, über so viele nachfolgende Zellen (Spalten in dieser Zeile) erstrecken wie angegeben. In unserem Beispiel umfasst die Zelle zwei Spalten. Eine dritte Spalte wird dann noch ganz normal notiert. In der zweiten Tabellenzeile des Beispiels sind dagegen alle drei Spalten einzeln definiert. Wir erreichen damit also folgenden Effekt:

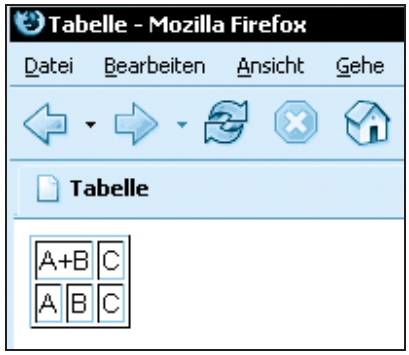


Abbildung 4.15: Tabellenzelle, die sich über zwei Spalten erstreckt

Etwas gewöhnungsbedürftiger ist die Notation, wenn sich eine Zelle über mehrere Zeilen erstrecken soll. Da in HTML die Tabelleninhalte im Quelltext zeilenweise notiert werden, muss man hierbei etwas mitdenken. Ein analoges Beispiel zu oben:

```
<table border="1">
   <tr>
      <td rowspan="2">A<br>+<br>A</td><td>B</td><td>C</td>
   </tr>
   <tr>
      <td>B</td><td>C</td>
   </tr>
</table>
```

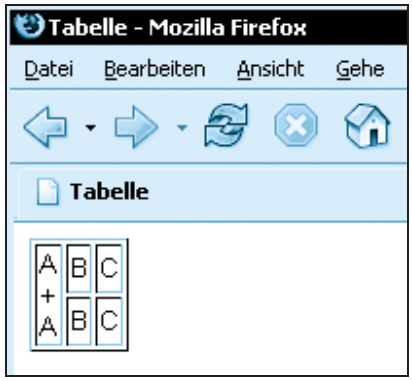


Abbildung 4.16: Tabellenzelle, die sich über zwei Zeilen erstreckt

Zum zeilenübergreifenden Verbinden von Zellen dient das Attribut `rowspan=`. Als Wert wird angegeben, über wie viele nachfolgende Zeilen sich die Zelle erstrecken soll. In den nachfolgenden Zeilen müssen Sie nun berücksichtigen, dass für die Spalte, bei der sich die Zelle erstreckt, keine Zelle notiert werden darf. Im Beispiel sind deshalb in der zweiten Tabellenzeile nur zwei Zellen definiert, obwohl die Tabelle eigentlich drei Spalten hat. Weggelassen wurde in diesem Fall die Zelle für die erste Spalte, da wir in der Zeile zuvor bei der Zelle für die erste Spalte angegeben haben, dass sich diese über zwei Zeilen erstrecken soll.

Die Kombination von `colspan=` und `rowspan=` in einer Zelle ist ebenfalls zulässig!

Features für die nicht visuelle Darstellung von Tabellen

Auch sehbehinderte Menschen nehmen am Informationsangebot im Web teil. Wenn die Sehbeeinträchtigung so stark ist, dass auch Lupen und Lichtverstärkung nichts mehr nutzen oder zu schnell ermüden, dann greifen solche Benutzer häufig zu Browser-Tools, die Webseiten mit synthetischer Stimme vorlesen, statt sie am Bildschirm anzuzeigen.

Wie Sie sicher bemerkt haben, muss man beim Notieren der einzelnen Zellen einer Tabelle im HTML-Quelltext gut mitdenken. Bei größeren Tabellen und umfangreichen Zelleninhalten kann auch schnell mal die Übersicht verloren gehen. Genau das gleiche Problem besteht, wenn Tabelleninhalte vorgelesen werden. Dabei geht nämlich der eigentliche Segen von Tabellen, also die übersichtliche Darstellung auf einen Blick, verloren. Stattdessen müssen die Zelleninhalte im Zeitkontinuum vermittelt werden. Vorgelesen werden die Zelleninhalte dabei so, wie sie im HTML-Quelltext stehen, also Tabellenzeile für Tabellenzeile. Doch wie erfährt ein Zuhörer, welche Inhalte zum Tabellenkopf gehören und welche zum Tabellenkörper? Oder wann sich eine Zelle über mehrere Spalten oder Zeilen erstreckt? Für diese Probleme stellt HTML eine Reihe von Attributen zur Verfügung. Wie die darin enthaltenen Informationen in einer nicht visuellen Repräsentation der Tabelle tatsächlich umgesetzt werden (z.B. durch eine zweite Stimme, die nur solche Meta-Informationen ausgibt, während die Hauptstimme die Tabellennutzdaten vorliest), ist Sache der verwendeten Software. Als HTML-Autor können Sie jedoch die erforderlichen Meta-Daten notieren.

Da das Vorlesen umfangreicher Tabellen etwas länger dauert, ist es sinnvoll, den Zuhörer darauf vorzubereiten und ihm vorweg eine kurze Zusammenfassung des Tabelleninhalts zu geben. Dazu stellt HTML das Attribut `summary=` für das einleitende `<table>`-Tag zur Verfügung. Ein Beispiel:

```
<table border="1" summary="Es folgt eine Kreuztabelle mit Entfernungen zwischen
großen deutschen Städten. Die Tabelle hat 25 Zeilen und 25 Spalten. ">
```

Auch für einzelne Tabellenzellen lässt sich eine zusammenfassende Kurzbeschreibung notieren. Das bietet sich an, wenn die Inhalte der Zellen selbst recht lang und komplex sind. Auch hierzu ein Beispiel:

```
<td abbr="Zeile 4 Spalte 8, Informationen zu den Lotto-Einnahmen des
Bundeslandes Niedersachsen der letzten 10 Jahre">
```

Das Attribut `abbr=` können Sie in einleitenden `<td>`- oder `<th>`-Tags notieren. Beim Vorlesen des Tabelleninhalts sollte eine entsprechende Software diese Kurzbeschreibungen mit ausgeben.

Besonders problematisch bei der zeilenweisen akustischen Präsentation von Tabellen ist die Zuordnung von Spalteninhalten zur zugehörigen Spaltenüberschrift. Auch dazu bietet HTML einen Mechanismus an. In den Spaltenüberschriften (Kopfdaten) erhält jede Zelle einen eindeutigen id-Namen. Die Datenzellen der jeweiligen Spalten können dann auf diese id-Namen Bezug nehmen. Ein einfaches Beispiel:

```
<table border="1">
   <tr>
     <th id="Mo">Montag</th>
     <th id="Di">Dienstag</th>
     <th id="Mi">Mittwoch</th>
   </tr>
   <tr>
     <td headers="Mo">Hühnerfrikassee mit Gemüsereis</td>
     <td headers="Di">Rostzwiebelbraten mit Pommes Frites</td>
     <td headers="Mi">Allgäuer Käsespätzle</td>
   </tr>
</table>
```

In den Kopfzellen `<th>`, von denen jede eine Spaltenüberschrift darstellt, wird ein id-Attribut notiert. Dies ist eigentlich ein Allgemein-Attribut, das jedem HTML-Element zugewiesen werden kann. Wichtig ist, dass die Wertzuweisung ein dokumentweit eindeutiger Name ist. Im Beispiel sind es Abkürzungen für Wochentage. Der Bezug zu den Spaltenüberschriften kann dann in den Datenzellen `<td>` über das Attribut `headers=` hergestellt werden.

Falls Sie eine Kreuztabelle definieren, also eine Tabelle, bei der sowohl die erste Zeile als auch die erste Spalte Kopfdateninformationen enthält und jede Datenzelle eine Schnittinformation aus den jeweiligen Zeilen-/Spaltenkoordinaten darstellt, können Sie auch bei den Zeilenüberschriften id-Attribute notieren. In den Datenzellen können Sie dann beim headers-Attribut beide id-Namen angeben, durch Leerzeichen getrennt.

Einen noch einfacheren Weg, Kopfzelleninformation in Datenzellen zu wiederholen, bietet das scope-Attribut an. Ein Beispiel:

```
<table border="1">
   <tr>
     <th scope="col">Montag</th>
     <th scope ="col">Dienstag</th>
     <th scope ="col">Mittwoch</th>
   </tr>
   <tr>
     <td>Hühnerfrikassee mit Gemüsereis</td>
     <td>Rostzwiebelbraten mit Pommes Frites</td>
     <td>Allgäuer Käsespätzle</td>
   </tr>
</table>
```

Bei dieser Variante wird in den Kopfzellen <th> das Attribut scope= notiert. Falls es sich um Spaltenüberschriften handelt, wie im Beispiel, muss scope="col" notiert werden. Falls es sich um Zeilenüberschriften handelt, lautet die Angabe scope="row". Auch die Angaben scope="colgroup" (bei Spaltengruppen) und scope="rowgroup" (für Tabellenkopf, Tabellenkörper und Tabellenfuß) sind zulässig. Akustische Ausgabesysteme sollten bei Verwendung des scope-Attributs den Zelleninhalt der zugehörigen Kopfzelle bei jeder Datenzelle wiederholen, um den Bezug transparent zu machen.

4.6 Einfache Formatierung mit CSS

In den voranstehenden Abschnitten haben Sie die wesentlichen HTML-Elemente zur Strukturierung von Textinhalten kennen gelernt. Dabei wurden Sie mehrfach darauf vertröstet, dass eine schicke Optik mit HTML allein nicht zu bewerkstelligen ist, sondern dass dafür *Cascading Style Sheets (CSS)* erforderlich sind. Bevor wir deshalb weitere HTML-Elemente behandeln, möchten wir Sie nicht länger auf die Folter spannen. In diesem Abschnitt werden Sie mit den Formatiermöglichkeiten von CSS vertraut gemacht.

4.6.1 Das style-Attribut

Zunächst werden wir mit dem Attribut style= arbeiten. Dieses Attribut dürfen Sie in allen HTML-Elementen notieren, die »sichtbare Inhalte« haben, egal ob es sich um Block- oder Inline-Elemente handelt. In der Wertzuweisung an das style-Attribut können Sie dann mithilfe CSS-eigener Syntax Formateigenschaften notieren, die dem jeweiligen Element zugeordnet werden. Beginnen wir mit dem Komplettlisting einer kleinen HTML-Datei, in der diverse Elemente mithilfe des style-Attributs formatiert werden:

Listing 4.4: HTML-Datei mit CSS-Formatierungen über das style-Attribut

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Arbeiten mit dem style-Attribut</title>
</head>
<body style="background-color:#FFFFC0; margin-left:30px;
             margin-right:30px;
             font-family:Tahoma,Helvetica,sans-serif">
<h1 style="font-size:28px; padding-bottom:20px;
           border-bottom:solid blue 4px;">
Das <em style="color:red">style</em>-Attribut</h1>
<p style="font-size:15px;">
Das <em style="color:red; font-weight:bold">style</em>-Attribut
eignet sich zur Formatierung einzelner HTML-Elemente.</p>
</body>
</html>
```



Abbildung 4.17: HTML-Datei mit CSS-Formatierungen über das style-Attribut



Abbildung 4.18: Gleiche HTML-Datei ohne CSS-Formatierungen

Die Abbildungen verdeutlichen, wie durch wenige CSS-Formatdefinitionen aus einer »nackten« HTML-Datei eine »gestaltete« Seite werden kann.

Zunächst hat das `<body>`-Tag ein style-Attribut erhalten. Da das body-Element den gesamten sichtbaren Dateikörper auszeichnet, wirken sich CSS-Definitionen für dieses Element auf die gesamte sichtbare Seite aus. Im Beispiel wird eine Hintergrundfarbe (`background-color`), ein linker und rechter Rand (`margin-left`, `margin-right`) sowie eine Angabe zur Grundschriftart (`font-family`) für die gesamte Seite notiert.

Die Hintergrundfarbe färbt das gesamte Browser-Fenster ein. Die Ränder links und rechts werden bei allen Inhalten innerhalb des Dateikörpers eingehalten und ebenso die Grundschriftart – solange in einem einzelnen Element nichts definiert wird, was diese Definitionen überschreibt. Damit wird bereits eine Grundeigenschaft von CSS deutlich: nämlich

das Prinzip der Vererbung von Eigenschaften. Wir werden darauf später noch genauer eingehen.

Innerhalb des body-Elements folgt im Beispiel eine Seitenüberschrift 1. Ordnung. Diese erbt die Schriftart des body-Elements, bekommt aber eine individuelle Schriftgröße (`font-size`), einen Rahmen unterhalb (`border-bottom`) sowie einen Innenabstand unten (`padding-bottom`) zugewiesen. Der Innenabstand sorgt dafür, dass die blaue Linie, die in Wirklichkeit der Rahmen unten ist, nicht zu nah am Überschriftentext klebt.

Innerhalb der h1-Überschrift ist das Wort »style« durch ein em-Element hervorgehoben. Dieses erbt wiederum alle Formateigenschaften der Überschrift und bekommt aber außerdem eine rote Schriftfarbe (`color`) zugewiesen. Dass das Wort dann in den meisten Browsern außerdem kursiv dargestellt wird, liegt an den Default-Einstellungen, mit denen Browser HTML-Elemente darstellen. Diese gehören ebenfalls zum »Erbe«. CSS-Formatierungen wirken immer nur additiv oder überschreibend. Um die Kursivdarstellung etwa beim em-Element wegzubekommen, könnte man die CSS-Formatierung `font-style:normal` notieren. Dadurch würde die Browser-Default-Einstellung »kursiv« überschrieben.

Auch das p- und das em-Element im Beispiel erhalten individuelle CSS-Formatierungen durch ein style-Attribut im Start-Tag.

4.6.2 Syntaxregeln für CSS-Formatdefinitionen

Betrachten wir nun den Aufbau der Formatdefinitionen. Die gesamte CSS-Syntax steht in der Wertzuweisung an das style-Attribut. Für die Formatdefinition gelten folgende wichtige Regeln:

- Jede Formatdefinition hat die Form `Eigenschaft:Wert`. Ein Doppelpunkt trennt den Namen der CSS-Eigenschaft von der Wertzuweisung.
- Wenn mehr als ein Eigenschaft-Wert-Paar notiert wird, müssen alle außer dem letzten jeweils mit einem Semikolon (`;`) abgeschlossen werden. Wird nur ein Eigenschaft-Wert-Paar notiert, darf es ebenfalls ein Semikolon am Ende erhalten (muss aber nicht). Das Gleiche gilt für das letzte von mehreren notierten Eigenschaft-Wert-Paaren.
- Bei den Namen von CSS-Eigenschaften spielt Groß-/Kleinschreibung keine Rolle. Üblich ist es jedoch, alle CSS-Eigenschaften klein zu schreiben.
- Vor und nach dem Doppelpunkt zwischen Eigenschaft und Wert dürfen auch Leerzeichen notiert werden.
- Wenn eine Eigenschaft mehrere Werte zugewiesen bekommt und Werte dabei selbst Leerzeichen enthalten, sollten die Werte in Anführungszeichen (`""`) gesetzt werden. Ein Beispiel:

```
font-family:"MS Sans Serif", Helvetica, sans-serif;
```

Die Namen der möglichen CSS-Eigenschaften sowie das Format oder das Schema der erlaubten Werte, die den Eigenschaften zugewiesen werden können, sind in der CSS-Spezifikation des W3-Konsortiums festgelegt.

In unserem Beispiel aus Listing 4.4 sind vermutlich noch einige Fragen zu den Wertzuweisungen an die CSS-Eigenschaften offen geblieben. Der Quelltext enthält CSS-Wertzuweisungen wie `background-color:#FFFFC0`, `color:red`, `border-bottom:solid blue 4px` oder `font-size:15px`. Daran ist zu erkennen, dass es bestimmte Maßeinheiten und Schemata für die Notation von Werten gibt. In den beiden folgenden Abschnitten werden wir zunächst auf die Regeln bei den Wertzuweisungen eingehen, bevor wir daran anschließend einzelne CSS-Eigenschaften beschreiben.

4.6.3 Farbangaben in CSS

CSS unterstützt das RGB-Farbmodell. Über dieses Modell sind ca. 16,7 Millionen unterschiedliche Farben definierbar. RGB steht für ***Red Green Blue*** (zu Deutsch: Rot Grün Blau). Im RGB-Farbmodell wird jede Farbe aus den Anteilen der Grundfarben Rot, Grün und Blau definiert. Die drei Werte aus diesen Grundfarben werden additiv behandelt, wodurch Mischfarben entstehen. Jeder der drei Werte kann zwischen 0 und 255 betragen. Dadurch entstehen 256 × 256 × 256, also ca. 16,7 Millionen mögliche Farben. Wie viele unterschiedliche Farben bei einem Anwender tatsächlich anzeigbar sind, hängt natürlich von dessen Hard- und Software ab, aber letztlich auch vom Sehvermögen.

Der Wert 0 bedeutet: keinen Anteil an der entsprechenden Grundfarbe. Der Wert 255 bedeutet: maximalen Anteil an der entsprechenden Grundfarbe. Angegeben werden die Werte in der Reihenfolge für die Farben Rot, Grün und Blau.

Wenn die Werte aller drei Grundfarben gleich sind, bedeutet dies eine Grauabstufung. Die Wertekombination 0, 0, 0 (Rot, Grün, Blau) erzeugt die Farbe Schwarz, die Kombination 255, 255, 255 die Farbe Weiß, eine Kombination wie 160, 160, 160 ein mittleres Grau. Durch Überwiegen einzelner Werte entstehen Farbtöne und deren Mischung. Die Wertkombination 0, 0, 255 bewirkt beispielsweise ein maximal kräftiges und reines Blau, und 255, 0, 255 ein maximal kräftiges Magenta, da in diesem Fall Rot und Blau gemischt werden.

Die nachfolgende Abbildung zeigt die sieben möglichen Mischungsergebnisse der drei Grundfarben, wenn man nur mit dem Maximalwert 255 arbeitet und mindestens eine der drei Grundfarben diesen Maximalwert erhält.

Zum Auswählen von Mischfarben sollten Sie ein Farbwähl-Tool benutzen. Grafikprogramme verfügen in der Regel über die Möglichkeit, RGB-Farben auszuwählen, aber auch HTML-Editoren.

Bei der interaktiven Farbauswahl in solchen Dialogen können Sie entweder RGB-Werte direkt eingeben oder durch Eingabe anderer Faktoren wie Farbton, Sättigung und Helligkeit berechnen lassen. Auch durch Maussteuerung anhand eines Farbtabletts lässt sich die gewünschte Farbe intuitiv ermitteln.

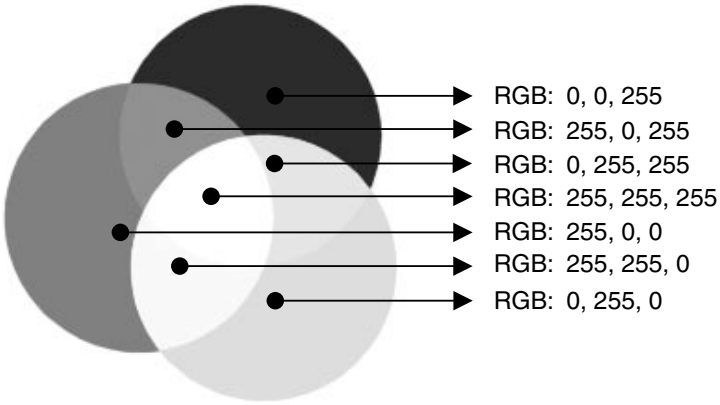


Abbildung 4.19: Mischfarben bei maximalen RGB-Farbanteilen (siehe Innenseite des hinteren Buchdeckels)

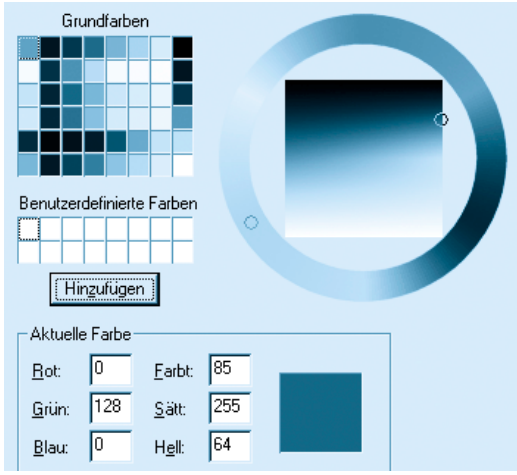


Abbildung 4.20: Typischer RGB-Farbauswahldialog in Grafikprogrammen

RGB-Farbangaben in CSS

In CSS können Sie RGB-Farbangaben folgendermaßen notieren (Beispiel):

```
<h2 style="color:rgb(0,0,255)">Blaue Überschrift</h2>
```

Die Wertzuweisung hat also das Schema `rgb(Rotwert, Grünwert, Blauwert)`. Anstelle der Zahlen 0 bis 255 können Sie auch Prozentwerte von 0 bis 100% angeben. Das nachfolgende Beispiel hat den gleichen Effekt wie das obige:

```
<h2 style="color:rgb(0%,0%,100%)">Blaue Überschrift</h2>
```

Hexadezimale RGB-Farbangaben

Obwohl die Notationsform `rgb(Rotwert, Grünwert, Blauwert)` viel praktischer ist, findet man in der heutigen Webpraxis noch immer wesentlich häufiger eine eher kryptische Variante von Farbangaben. Der Grund dafür ist, dass Farben auch in verschiedenen, heute allerdings nicht mehr empfehlenswerten HTML-Attributen angegeben werden können und HTML im Gegensatz zu CSS ausschließlich die hexadezimale Notationsform von Farbangaben unterstützt.

Diese Notationsform hat das Schema `#XXXXXX`, also ein Gatterzeichen, gefolgt von sechs Zeichen, ohne Leerzeichen dazwischen. Die sechs hier mit `X` bezeichneten Zeichen stellen Hexadezimalziffern dar. Hexadezimale Ziffern sind die dezimalen Ziffern `0` bis `9` sowie die Buchstaben `A`, `B`, `C`, `D`, `E` und `F`. Eine Hexadezimalziffer kann also insgesamt 16 Werte annehmen (deshalb: Hexadezimalsystem).

Die sechs Ziffern der hexadezimalen Notationsform für Farben sind folgendermaßen zu lesen:

- Ziffer 1 und 2 bestimmen den Rotanteil,
- Ziffer 3 und 4 bestimmen den Grünanteil,
- Ziffern 5 und 6 bestimmen den Blauanteil.

Da mit jeder Hexadezimalziffer 16 Zahlenwerte darstellbar sind, lassen sich mithilfe von zwei solchen Ziffern 16 × 16 = 256 Zahlenwerte darstellen – also genau der Wertebereich eines Farbwerts im RGB-Modell. Es genügen demnach zwei Hexadezimalziffern pro Farbanteil.

Der Wert 255 entspricht beispielsweise dem Hexadezimalwert FF und der Wert 192 lautet hexadezimal C0 (C entspricht dezimal 12, C0 bedeutet also 12 × 16 + 0 = 192). Die Notation `#FFFFC0` bedeutet also das Gleiche wie `rgb(255, 255, 192)` – ein helles, mattes Gelb.

Selbst wenn Sie die Hexadezimalnotation nicht mehr benötigen, weil CSS die praktischere RGB-Schreibweise anbietet, so sollten Sie diese ältere Notationsform dennoch kennen und verstehen, da einfach noch so viele Webseiten damit arbeiten. Hin und wieder stößt man auch auf eine abgekürzte Hexadezimalnotation mit nur drei Ziffern, also etwa `#FFF` oder `#99C`. Die verkürzte Schreibweise bedeutet intern eine Verdopplung der Ziffernfolge. `#FFF` bedeutet in Wirklichkeit `#FFFFFF`, `#99C` bedeutet `#9999CC`.

Farbnamen

Noch einfacher als die RGB-Schreibweise für Farbdefinitionen sind Farbnamen. CSS selbst unterstützt folgende Farbnamen.

Farbname	Hexadezimal	RGB
`black`	`#000000`	`rgb(0, 0, 0)`
`maroon`	`#800000`	`rgb(128, 0, 0)`

Tabelle 4.9: Erlaubte Farbnamen

Farbname	Hexadezimal	RGB
green	#008000	rgb(0, 128, 0)
olive	#808000	rgb(0, 128, 128)
navy	#000080	rgb(0, 0, 128)
purple	#800080	rgb(128, 0, 128)
teal	#008080	rgb(0, 128, 128)
gray	#808080	rgb(128, 128, 128)
silver	#C0C0C0	rgb(192, 192, 192)
red	#FF0000	rgb(255, 0, 0)
lime	#00FF00	rgb(0, 255, 0)
yellow	#FFFF00	rgb(255, 255, 0)
blue	#0000FF	rgb(0, 0, 255)
fuchsia	#FF00FF	rgb(255, 0, 255)
aqua	#00FFFF	rgb(0, 255, 255)
white	#FFFFFF	rgb(255, 255, 255)
orange	#FFA500	rgb(255, 165, 0)

Tabelle 4.9: Erlaubte Farbnamen (Fortsetzung)

Insider werden erkennen, dass es sich bei den ersten 16 der 17 aufgelisteten Farben um die klassischen VGA-Farben handelt. Die 17. Farbe, `orange`, wurde vom W3-Konsortium zusätzlich für CSS definiert, weil Orange im klassischen VGA-Schema nicht vorkommt. Ein Beispiel:

```
<h2 style="color:blue">Blaue Überschrift</h2>
```

Neben diesen Farbnamen erlaubt CSS auch die Angaben umgebungsspezifischer Farben. Diese Farben bezeichnen Farbwerte, die Sie als Designer gar nicht kennen, da es sich um Farben handelt, die bei jeden Anwender anders sind und die er sich selber in seiner Benutzeroberfläche eingestellt hat. Wenn Sie ausschließlich mit diesen Farbnamen arbeiten, geben Sie zwar die farbliche Kontrolle ab, doch Sie kommen Ihren Anwendern optimal entgegen – die Annahme vorausgesetzt, dass alle Anwender von sich aus für sie optimale und angenehme Farben in ihrer Benutzeroberfläche einstellen. Ein Beispiel:

```
<h2 style="background-color:ActiveBorder; color:ActiveCaption">Überschrift in
Systemfarben</h2>
```

Im Beispiel wird für die Überschrift als Hintergrundfarbe (`background-color`) die beim Anwender eingestellte Hintergrundfarbe für aktive Fenstertitel definiert und als Textfarbe (`color`) die zugehörige Vordergrundfarbe des aktiven Fenstertitels.

Die nachfolgende Tabelle listet erlaubte Umgebungsfarbnamen auf.

Farbname	Bedeutung
ActiveBorder	Farbe der aktiven Fenstertitelzeile
ActiveCaption	Farbe des Titels in der aktiven Fenstertitelzeile
AppWorkspace	Farbe des Hintergrunds der aktiven Anwendung
Background	Farbe des Desktop-Hintergrunds
ButtonFace	Farbe von Buttons in Dialogfenstern
ButtonHighlight	Helle Farbe für 3D-Schatten von Buttons
ButtonShadow	Helle Farbe für 3D-Schatten von Buttons
ButtonText	Farbe von Button-Beschriftungen
CaptionText	Farbe von Überschriften in Dialogfenstern
GrayText	Farbe von deaktiviertem Text Dialogfenstern
Highlight	Farbe von selektierten Einträgen in Auswahllisten
HighlightText	Farbe von selektiertem Text
InactiveBorder	Farbe einer inaktiven Fenstertitelzeile
InactiveCaption	Hintergrundfarbe des Titels der inaktiven Fenstertitelzeile
InactiveCaptionText	Farbe des Titels der inaktiven Fenstertitelzeile
InfoBackground	Hintergrundfarbe für Tooltips und Hints
InfoText	Textfarbe für Tooltips und Hints
Menu	Farbe für Menüleisten
MenuText	Farbe für Menüeinträge
Scrollbar	Farbe der Scrollleisten in Fenstern
ThreeDDarkShadow	Dunkle Farbe bei Schatten von 3D-Elementen
ThreeDFace	Farbe von 3D-Elementen
ThreeDHighlight	Farbe von gerade angeklickten 3D-Elementen
ThreeDLightShadow	Helle Farbe bei Schatten von 3D-Elementen
ThreeDShadow	Dunkle Farbe bei Schatten von 3D-Elementen
Window	Hintergrundfarbe von Dokumentfenstern
WindowFrame	Farbe von Fensterrahmen
WindowText	Farbe von normalem Text in Dokumentfenstern

Tabelle 4.10: Umgebungsabhängige Farbnamen

Websichere Farben

Bei älteren Computern, aber auch bei Mini-Computern wie Handhelds, Tablet-PCs usw. ist das Spektrum gleichzeitig darstellbarer Farben oft reduziert, häufig auf nur 256 Farben. Eine Palette legt fest, welche Farben das sind. Jedes System kann dabei seine eigene Default-Farbpalette einsetzen. Stimmt eine in CSS definierte Farbe nicht mit einer in der Palette verfügbaren Farbe überein, muss das System irgendeine Lösung finden, um den Konflikt zu lösen. Es könnte z. B. die nächstähnliche Farbe aus seiner Palette auswählen. Oder es könnte versuchen, durch so genanntes Dithering die gewünschte Farbe zu erzeugen. Dabei werden Pixelmuster aus zwei oder mehreren Farben gebildet, um die

gewünschte Farbe einigermaßen echt darzustellen. Das Dithering ist jedoch je nach Pixelauflösung und Muster durchaus sichtbar und kann z. B. als Textfarbe angewendet die Lesbarkeit beeinträchtigen.

Aus diesem Grund hat sich schon früh eine Farbpalette durchgesetzt, die praktisch alle Webbrowser in allen geeigneten Hard- und Softwareumgebungen anzeigen können. Die in dieser Palette definierten Farben gelten als *websichere* Farben. Die Tabelle am Ende des Buchs listet alle 216 websicheren Farben auf.

Die Farben dieser Palette bestehen aus RGB-Farben mit den Dezimalwerten 0, 51, 102, 153, 204 oder 255 (hexadezimal: 00, 33, 66, 99, CC, FF). Alle daraus resultierbaren Farben sind erlaubt, also z. B. `rgb(51, 102, 255)` oder `#990000`. Da es sich um sechs mögliche Farbwerte handelt und drei Farbwerte jede Farbe bestimmen, sind dadurch insgesamt 6 × 6 × 6 = 216 Farben möglich. Die typische Farbtiefe von 256 Farben wird damit nicht ganz ausgereizt – es verbleiben 40 Farben, die z. B. für die Benutzeroberfläche des Browsers benötigt werden.

Links zu weiteren Informationen zu Farben:

Farbenlehre:
http://www.farbenlehre.com/
Farbentheorie und Farbgestaltung:
http://www.ipsi.fraunhofer.de/~crueger/farbe/
Farben im Webdesign:
http://www.metacolor.de/

4.6.4 Maßangaben in CSS

Wenn Sie in CSS numerische Werte für Breiten, Höhen, Abstände usw. definieren, müssen Sie eine Maßeinheit angeben.

Maß	Bedeutung	Typ
`pt`	***point*** (zu Deutsch: Punkt). Typografische Maßeinheit. `1pt` entspricht ca. `1.39in`.	absolut
`pc`	***pica***. Typografische Maßeinheit. `1pc` entspricht `12pt`.	absolut
`in`	***inch***. Allgemeines Längenmaß im angelsächsischen Raum. `1in` entspricht `2.54cm`.	absolut
`mm`	***millimeter***. `1mm` entspricht `0.1cm`.	absolut
`cm`	***centimeter***. Allgemeines Längenmaß.	absolut
`px`	***pixel***. Abhängig von der Pixeldichte des Ausgabegeräts, relativ also von Ausgabegerät zu Ausgabegerät, absolut dagegen auf ein und dasselbe Ausgabegerät bezogen.	relativ
`em`	***element***. Schriftgröße des relevanten Elements.	relativ
`%`	***percent***. Prozentangabe.	relativ

Tabelle 4.11: Maßangaben in CSS

Generell wird zwischen absoluten Maßangaben und relativen Maßangaben unterschieden. So ist beispielsweise `cm` (Zentimeter) eine absolute Angabe, während `em` (elementeigene Schriftgröße) eine relative Angabe darstellt. Hardliner empfehlen immer wieder, man solle ausschließlich relative Angaben verwenden, da man als Webdesigner niemals wissen kann, welche Art von Ausgabegerät ein Anwender verwendet oder wie groß sein Browser-Fenster ist. Für die Praxis empfiehlt sich kein völliger Verzicht auf absolute Angaben, jedoch ein behutsamer Umgang damit. Für reine Bildschirmpräsentationen wird in der Praxis am häufigsten mit der `px`-Bemaßung gearbeitet, also mit Pixelangaben, sowie mit Prozentangaben. Angaben wie `pt`, `pc`, `in`, `mm` und `cm` eignen sich dagegen eher für Print-Ausgaben von CSS-formatierten HTML-Dokumenten. Die Angaben `em` und `ex` schließlich sind besonders zum relativen Abstimmen von Schriftgrößen geeignet.

Regeln für numerische Werte

Folgende Regeln müssen Sie bei numerischen Werten in CSS beachten:

- Das Dezimalzeichen wird in der englisch/amerikanischen Form notiert, d.h. als Punkt, nicht das Komma. Tausenderpunkte und Tausenderkommas sind nicht zulässig.
- Manche CSS-Eigenschaften erlauben auch die Angabe negativer numerischer Werte. Solche Werte werden wie üblich durch ein voranstehendes Minuszeichen gekennzeichnet.
- Bei Zuweisung des Werts 0 für Längen, Breiten, Abstände und Größen ist keine Maßangabe zwingend erforderlich, bei allen anderen Werten dagegen schon.

Ein paar gültige Beispiele:

- `width:300px` definiert eine Breite von 300 Pixel.
- `margin-top:0.7cm` definiert einen Abstand oben von 0,7 Zentimetern.
- `font-size:0.95em` definiert eine Schriftgröße von 95%, gemessen an der Schriftgröße des Elternelements.
- `border-width:0.06in` definiert eine Rahmendicke von ca. 0,15 Zentimetern.
- `max-width:90%` legt fest, das ein Element maximal 90% Breite seiner verfügbaren Breite einnehmen darf.

4.6.5 CSS-Eigenschaften für Schriftformatierung

Die Formatierung von texttragenden Elementen wie Überschriften, Textabsätzen, Listenpunkten oder Tabellenzellen gehört zu den vordringlichsten Wünschen der meisten Autoren, die ein HTML-Dokument ohne CSS-Angaben im Browser betrachten. Die Times-Schriftart, welche die meisten Browser als Default-Schriftart eingestellt haben, wirkt auf viele HTML-Autoren nicht sehr ästhetisch. Auch die Browser-Defaults für Schriftgrößenverhältnisse zwischen Überschriften und Text werden oftmals als unbefriedigend empfunden. Und last but not least wirken schwarzweiße Textwüsten am Bildschirm viel ermüdender als etwa in einem Buch. Farben sollen also her.

CSS erfüllt fast alle Formatierwünsche. Allerdings stößt gerade die Schriftformatierung in der Praxis an sehr enge Grenzen. Der Grund ist, dass Webseiten auf den unterschiedlichsten Rechnertypen, Benutzeroberflächen und Anzeigegeräten dargestellt werden können. Wenn Sie beispielsweise Ihre Webseiten unter MS Windows entwickeln, können Sie nicht erwarten, dass Linux/KDE-Anwender Schriftarten wie Arial, die unter Windows zum Standard gehören, installiert haben. Umgekehrt sind dort andere Schriftarten viel verbreiteter. Kaum weniger problematisch sind auch die Schriftgrößen. Im Internet herrscht ein regelrechter Glaubenskrieg zwischen aggressiven Vertretern, die einem Webdesigner am liebsten jede Einflussnahme auf Schriftgrößen verbieten wollen, und ignoranten Design-Freaks, die leider meistens auch unter einem schlimmen Schriftminiaturisierungswahn leiden.

Als Grundregel für die Schriftformatierung kann gelten: Außer bei Ausnahmefällen (z.B. individuell formatierte, auffällige Schriftzüge) sollten Sie mit starken Abweichungen gegenüber den Browser-Defaults stets behutsam sein.

Schriftart

Schriftarten können Sie mit `font-family` definieren. Es besteht jedoch keine Garantie, dass die angegebene Schriftart bei allen Anwendern angezeigt werden kann, da dies sehr stark von der jeweiligen Hard- und Softwareumgebung abhängt. Damit sich die Trefferquote verbessert, ist es bei der font-family-Eigenschaft nicht nur möglich, sondern auch dringend zu empfehlen, stets mehrere Schriftarten anzugeben. Ein Beispiel:

```
<p style="font-family:Tahoma, Arial, Helvetica, sans-serif">
   Text
</p>
```

Im Beispiel werden drei Schriftarten angegeben. Alle Schriftarten werden durch Kommata getrennt. Die Reihenfolge entspricht der Priorisierung. Das bedeutet im Beispiel: Wenn die Schriftart Tahoma verfügbar ist, soll diese verwendet werden. Ist sie nicht verfügbar, soll Arial verwendet werden. Ist Arial ebenfalls nicht verfügbar, soll Helvetica verwendet werden. Ist auch Helvetica nicht verfügbar, soll irgendeine verfügbare serifenlose Schriftart verwendet werden.

Die angegebene Kombination im Beispiel geht von folgender Überlegung aus: Eigentlich gewünscht (aus welchen Gründen auch immer) ist die Schriftart Tahoma. Diese ist jedoch eine typische MS-Windows-Schriftart, die auch üblicherweise nur zusammen mit MS Office installiert wird. Wegen der hohen Verbreitung von MS Office und MS Windows ist natürlich anzunehmen, dass Tahoma bei vielen Anwendern angezeigt werden kann. Die zweite Wahl, Arial, stützt sich auf die Überlegung, dass die überwiegende Anzahl der Anwender MS Windows einsetzt, wo Arial seit jeher eine der Grundschriftarten ist. Ebenso wie Tahoma ist Arial eine serifenlose Schrift, auch wenn sie ansonsten typografisch durchaus von Tahoma abweicht. Die dritte Wahl, Helvetica, bedient sich des Wissens, dass auf vielen anderen Benutzeroberflächen als MS Windows die Schriftart Helvetica als eine der meist verfügbaren Grundschriftarten installiert ist. Sollte das alles nichts helfen, so wird der Browser durch die Angabe der generischen Schriftart sans-serif

angewiesen, irgendeine verfügbare serifenlose Schrift zu verwenden. Voraussetzung dafür ist natürlich, dass die zugrunde liegende Benutzeroberfläche dem Browser eine Möglichkeit der Kategorisierung von Schriftarten anbietet.

Die letzte Angabe bei einer Wertzuweisung an `font-family` sollte stets eine so genannte *generische Schriftart* sein. CSS stellt folgende Angaben für generische Schriftarten zur Verfügung:

Wert	Bedeutung
`serif`	Schriftart mit Serifen
`sans-serif`	Schriftart ohne Serifen
`cursive`	Schriftart für Schreibschrift
`fantasy`	Schriftart für eine ungewöhnliche Schrift
`monospace`	Nichtproportionale Schriftart (dicktengleiche Zeichen)

Tabelle 4.12: Generische Schriftarten in CSS

Verwenden Sie auf ein und derselben Webseite nicht zu viele unterschiedliche Schriftarten. Das gilt als grober typografischer Fehler und wird auch tatsächlich von den meisten Anwendern als eher unprofessionell empfunden. Als Faustregel gilt: nicht mehr als zwei unterschiedliche Schriftarten.

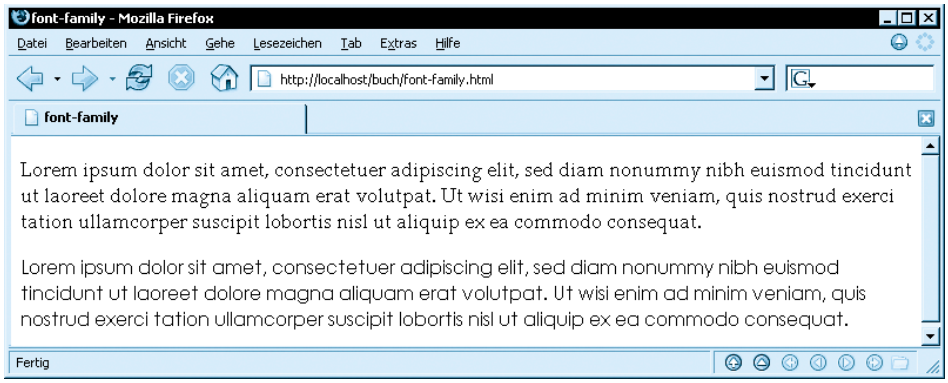


Abbildung 4.21: Gleicher Text mit Serifenschrift und serifenloser Schrift

Die angemessene Wahl der Schriftart hängt auch von den Textinhalten ab. Bei einem lateinischen Text wie dem klassische Blindtext in Abbildung 4.21 wirkt eine Serifenschrift wie die obere Book Antiqua edler und inhaltstypischer. Bei einer technischen Dokumentation dagegen sind serifenlose Schriften üblicher. Doch auch dabei gibt es Unterschiede. Schriftarten wie die untere Century Gothic in der Abbildung wirken schick, können aber bei längeren Texten am Bildschirm schwerer lesbar sein als etwa Arial.

Schriftgröße

Die Schriftgröße können Sie mit der CSS-Eigenschaft `font-size` bestimmen. Ein Beispiel:

```
<body>
<h1 style="font-size:36px">
   <span style="font-size:1.5em">Große</span> Überschrift
</h1>
<h1 style="font-size:1.5em">
   Kleine Überschrift
</h1>
<h1>
   Unformatierte Überschrift
</h1>
</body>
```

Das Beispiel verdeutlicht die Zusammenhänge bei der Schriftgrößendefinition an Hand von drei h1-Überschriften. In der ersten h1-Überschrift wird durch die Definition `font-size:36px` eine gemessen am Ausgabegerät absolute Größe festgelegt. Das innere span-Element erhält dagegen die relative Größenangabe `font-size:1.5 em`. Für die Maßeinheit `em` gilt: Wird sie auf andere Größenangaben als die Schriftart angewendet, ist stets `1em` die Größe der Schriftart des aktuellen Elements. Wird sie dagegen auf die Schriftgröße selbst (CSS-Eigenschaft `font-size`) angewendet, so entspricht `1em` der Schriftart des Elternelements. In unserem Beispiel wird `em` auf die Schriftgröße selbst angewendet. Zur Berechnung der anzuzeigenden Schriftgröße bei der Angabe `font-size:1.5 em` greift der Browser daher auf die Schriftgröße des Elternelements zurück, also auf das h1-Element. Diese beträgt `36px`, also wird das Wort »Große« mit dem Faktor 1.5 angezeigt, d.h. in einer Schriftgröße von 54 Pixel.

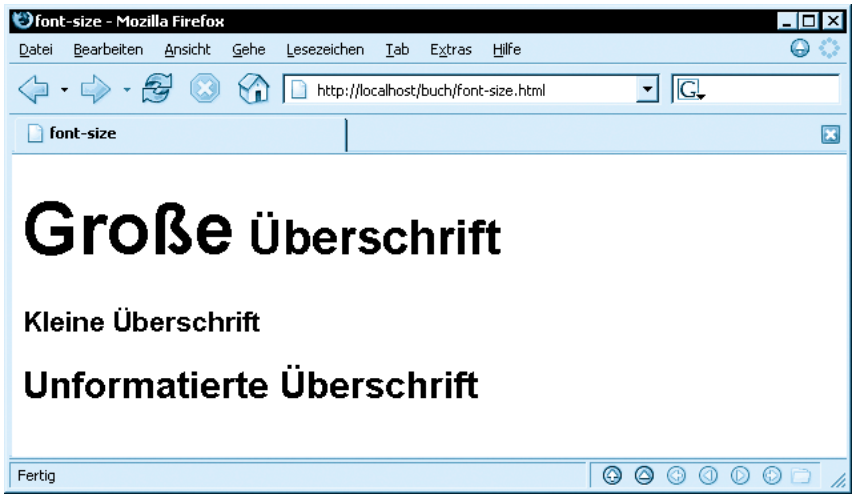


Abbildung 4.22: Schriftgrößenangaben und ihre Wirkung im Browser

Möglicherweise verwirrend ist die zweite h1-Überschrift, wenn man die Wirkung im Browser betrachtet. Obwohl sie die Angabe `font-size:1.5em` enthält, erscheint sie im Browser kleiner als die dritte h1-Überschrift, der gar keine CSS-Formatierung zugewiesen ist. Der Grund ist auch hier das zuvor beschriebene Verhalten bei der Bemaßung mit `em`. Zur Berechnung der tatsächlich anzuzeigenden Schriftgröße wird die Schriftgröße des Elternelements verwendet. Das gemeinsame Elternelement der h1-Überschriften ist das body-Element, also der Dateikörper selbst. Dessen Grundschriftgröße entspricht natürlich der von normalem Fließtext, und nicht der von Überschriften erster Ordnung. Angenommen, für Fließtext ist im Browser eine Grundschriftgröße von 15 Pixel eingestellt, so führt die Angabe 1.5em zu einer berechneten Schriftgröße von 22,5 Pixel. Halbe Pixel können natürlich aus technisch nachvollziehbaren Gründen nicht angezeigt werden – der Browser wird also 22 oder 23 Pixel Größe verwenden.

Die in der Drucktechnik üblichen Schriftgrößenbemaßungen ***Punkt*** (pt) und ***Pica*** (pc) sind für Webseiten, die vornehmlich auf Bildschirmen angezeigt werden und nicht zum Ausdrucken bestimmt sind, weniger geeignet. Der Grund ist, dass diese Angaben bei der Bildschirmanzeige intern auf Pixel umgerechnet werden müssen. Der Umrechnungsfaktor ist jedoch systemabhängig. Unter MS Windows beispielsweise entspricht 1 Punkt intern 1,33 Pixel, weil Windows mit 96 dpi arbeitet, während 1 Punkt auf einem Macintosh- oder Linux-Rechner genau in 1 Pixel umgerechnet wird, weil diese Systeme mit 72 dpi umrechnen. In der Praxis bedeutet dies, dass eine mit `font-size:10pt` definierte Schrift auf einem Windows-Rechner 13 Pixel groß erscheint, während die gleiche Schrift etwa auf einem Macintosh-Rechner nur 10 Pixel groß dargestellt wird. Dieser Unterschied ist durchaus erheblich – viele Anwender empfinden 13 Pixel Darstellungsgröße als angenehm, aber 10 Pixel als zu klein.

Schriftfarbe

Die Vordergrundfarbe von Text definieren Sie in CSS mit der Eigenschaft `color`. Ein Beispiel:

```
<body style="background-color:rgb(204,255,204)">
<h1 style="color:rgb(0,80,0)">
   Rettet unser Grün!
</h1>
</body>
```

Zur Bestimmung der Schriftfarbe notieren Sie eine Farbangabe wie in ***Abschnitt 4.6.3*** beschrieben. Beim Definieren von Schriftfarben müssen Sie vor allem auf die Kombination mit der Hintergrundfarbe achten. Ein häufig begangener Fehler besteht darin, dass nirgends eine Hintergrundfarbe definiert wird. Die Annahme, dass die dunkelblaue Schrift auf dem weißen Default-Browser-Fensterhintergrund ordentlich aussieht, trügt nämlich: Ein Anwender könnte auch Schwarz als seinen Default-Fensterhintergrund eingestellt haben. In diesem Fall (Dunkelblau auf Schwarz) würde Ihre Seite gar keinen guten Eindruck machen.

Das obige Beispiel vermeidet diesen Fehler, indem im body-Element mit `background-color` eine hellgrüne Hintergrundfarbe definiert wird. Die Farbdefinition in der h1-Überschrift (dunkles Grün) kann sich damit auf eine passende Hintergrundfarbe verlassen.

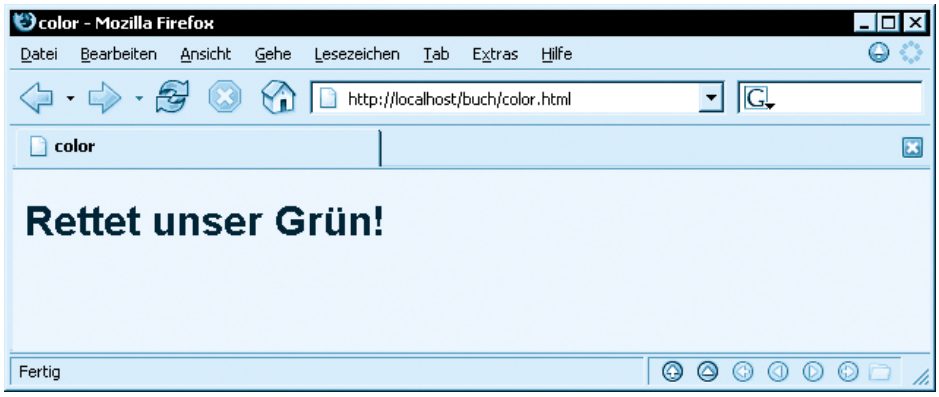


Abbildung 4.23: Textfarbe und Hintergrundfarbe

Vorder- und Hintergrundfarbe sollten einen guten Kontrast haben. Zu schwache Kontraste bereiten Mitmenschen mit Sehschwächen schnell Leseschwierigkeiten. Es gibt verschiedene Arten von Farbkontrasten, z.B. Komplementärfarben wie Gelb und Blau, oder Hell-Dunkel-Kontraste mit der gleichen Grundfarbe, wie im obigen Beispiel Dunkelgrün und Hellgrün. Nicht wenige Menschen leiden an einer Rot-Grün-Sehschwäche. Mischungen von Rot und Grün können dabei leicht zu Problemen führen. Auch das sollte bei der Farbwahl berücksichtigt werden.

Weitere Angaben zur Schriftformatierung

Weitere Formatierungsmöglichkeiten zur Beeinflussung des Schriftbilds betreffen das Schriftgewicht, den Schriftstil, den Wort- und Zeichenabstand, die Zeilenhöhe sowie verschiedene Effekte.

Das *Schriftgewicht* können Sie mit `font-weight` festlegen. Die Angabe `font-weight:bold` erzwingt eine Fettschrift, während `font-weight:normal` ein normales Schriftgewicht bewirkt. Bei der Default-Darstellung von HTML-Elementen verwenden moderne grafische Browser teilweise Fettschrift, teilweise Normalschrift. Für Überschriften bis zur vierten Ebene werden meistens Fettschriften verwendet. Durch eine Angabe wie `<h1 style="font-weight:normal"> ... </h1>` können Sie also beispielsweise die Default-Darstellung aushebeln. Umgekehrt können Sie mit einer Angabe wie `<p style="font-weight:bold"> ... </p>` das Schriftgewicht eines Elements, dessen Default-Darstellung normalerweise nicht fett ist, Fettschrift erzwingen.

Der *Schriftstil* bestimmt die Neigung der Schrift und kann durch `font-style` festgelegt werden. Wie beim Schriftgewicht wird eine abweichende Schriftneigung von vielen Browsern bei bestimmten Elementen, z.B. beim em-Element, zur Darstellung verwendet. Durch

`font-style:italic` oder `font-style:oblique` erzwingen Sie eine Kursivschrift und durch `font-style:normal` eine normale Darstellung.

Manche Schriftarten benötigen Korrekturen bei den Defaults für Zeichen- und Wortabstände und bei der Zeilenhöhe. Erst dadurch ergibt sich ein abgerundetes und angenehmes Schriftbild.

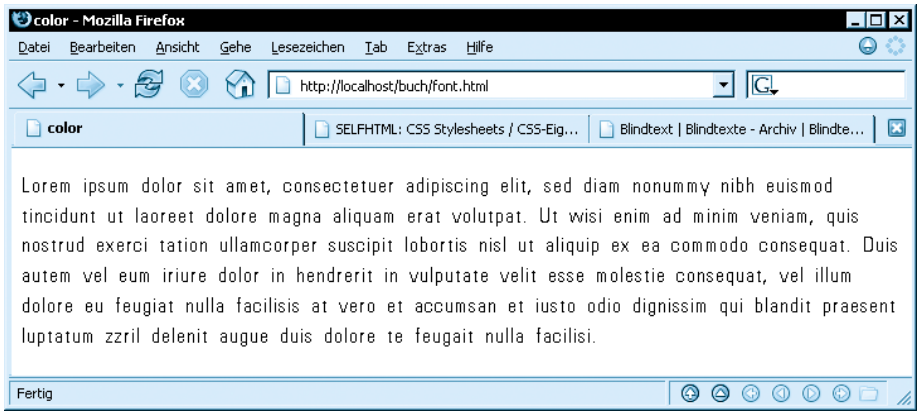


Abbildung 4.24: Schriftbildbeeinflussung durch Zeichen-/Wortabstand und Zeilenhöhe

Den ***Zeichenabstand*** bestimmen Sie durch `letter-spacing` und den Abstand zwischen Wörtern mit `word-spacing`. Die Zeilenhöhe eines Block-Elements kann mit `line-height` verändert werden. Bei fest definierter Zeilenhöhe muss die Schriftgröße aller davon betroffenen Textinhalte dazu passen, andernfalls könnte Text bei der Darstellung abgeschnitten werden. Das gilt besonders, wenn ein Block-Element Inline-Elemente enthält, die wiederum größere Schriftgrößen erzwingen. Auch im Text platzierte Grafiken können bei definierter Zeilenhöhe abgeschnitten werden. Das Experimentieren mit Zeichenabstand, Wortabstand und Zeilenhöhe empfiehlt sich vor allem bei schwer lesbaren Schriften, z. B. bei engen Schriften mit schmalen Zeichen.

Eine weitere Möglichkeit besteht darin, Text unterstrichen oder durchgestrichen darzustellen. Dazu dient die CSS-Eigenschaft `text-decoration`. Ein Beispiel:

```
<p>
Unser Rasierer - jetzt nur noch <span style="text-decoration:line-through">49,95
&amp;euro;</span><span style="text-decoration:underline">39,95 &amp;euro;</span>
</p>
```

Mit `text-decoration:underline` können Sie Text unterstreichen und mit `text-decoration:line-through` streichen Sie ihn durch.

Eine interessante Möglichkeit der Formatierung bietet schließlich auch noch die Angabe `font-variant:small-caps`. Dadurch wird Text in Kapitälchen dargestellt, d.h., Kleinbuchstaben werden als Großbuchstaben angezeigt, jedoch in der Größe von Kleinbuchstaben.

Links zu weiteren Informationen zu Schriftformatierung und Schriftarten:

Artikel »Das Windows/Mac font-size-Problem«:
http://aktuell.de.selfhtml.org/artikel/css/fontsize/
Artikel »Eine Frage der Größe«:
http://www.fabrice-pascal.de/artikel/sizediscussion/
Typograph online:
http://www.typo-info.de/

4.6.6 Das Boxmodell von CSS

Bevor wir uns weiteren CSS-Eigenschaften widmen, ist ein wichtiger Theorieabschnitt notwendig. Nur wenn Sie diese Grundlagen verstehen, können Sie souverän mit den anschließend beschriebenen CSS-Eigenschaften umgehen.

Das Boxmodell von CSS beschreibt den Erstreckungsraum und das Anordnungsverhalten von Block- und Inline-Elementen in HTML. Jedes Element wird dabei als rechteckige Box betrachtet.

Die Boxen der Basiselemente

Es beginnt beim html-Element und beim body-Element. Diese beiden stellen bereits eigene Boxen mit unterschiedlichen Eigenschaften dar. Ein Test soll das verdeutlichen. Das folgende Listing zeigt ein vollständiges HTML-Dokument, bei dem das html- und das body-Element mithilfe von CSS je einen Rahmen erhalten:

Listing 4.5: HTML-Dokument zur Visualisierung von html- und body-Element

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html style="border:solid 4px blue">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Boxmodell</title>
</head>
<body style="border:solid 4px red">
<h1>Some body</h1>
<p>Some thing</p>
</body>
</html>
```

Im `<html>`-Tag wird ein 4 Pixel dicker blauer Rahmen definiert und im `<body>`-Tag ein ebenso dicker roter Rahmen. Das Ergebnis im Browser ist in mehrfacher Hinsicht aufschlussreich:

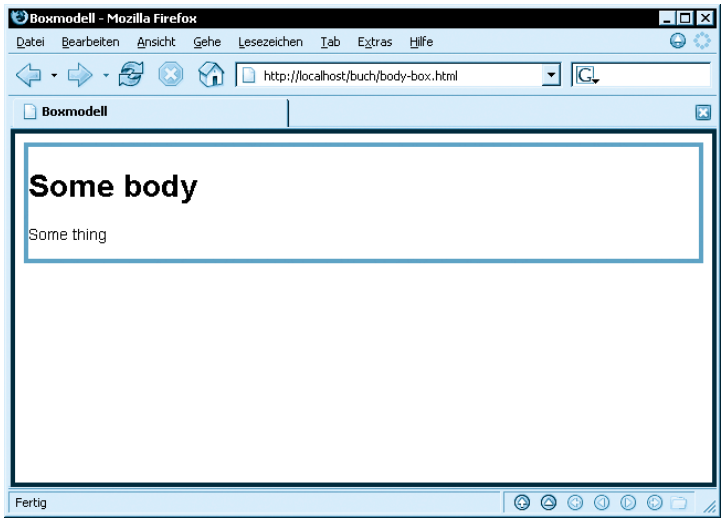


Abbildung 4.25: Boxvisualisierung des html- und des body-Elements

Gut erkennbar ist, dass das html-Element eine andere Box darstellt als das body-Element. Das html-Element hat kein anderes Elternelement. Sein Bezugspunkt ist daher einfach das Browserfenster, im CSS-Fachjargon als *Viewport* bezeichnet. Der blaue Rahmen des html-Elements erstreckt sich, wie die Abbildung zeigt, direkt bis zu den Fensterrändern.

Das body-Element ist dagegen ein Kindelement des html-Elements. Seine Box hat ganz andere Eigenschaften. Zunächst fällt auf, dass es einen gewissen Abstand zur Box des html-Elements hat, obwohl mit CSS kein solcher Abstand definiert wurde. Der Abstand zwischen dem blauen und dem roten Rahmen ist einfach ein Default-Abstand, den der Browser benutzt, damit der Inhalt eines HTML-Dokuments nicht direkt am Fensterrand klebt.

Weiter fällt auf, dass die Box des body-Elements in der Breite so viel Raum einnimmt wie verfügbar ist, obwohl der Inhalt an keiner Stelle die volle Breite erreicht. Dagegen nimmt die Box in der Höhe so viel Raum ein, wie es der Inhalt erfordert. Genau dies sind die typischen Verhaltensweisen für Block-Elemente: so breit wie möglich, so hoch wie nötig.

Die Boxen von Block-Elementen

Im zweiten Schritt versehen wir die beiden Textelemente des Beispiels ebenfalls mit Rahmen:

```
<h1 style="border:solid 4px lime">Some body</h1>
<p style="border:solid 4px violet">Some thing</p>
```

Der Browser zeigt nun Folgendes an:

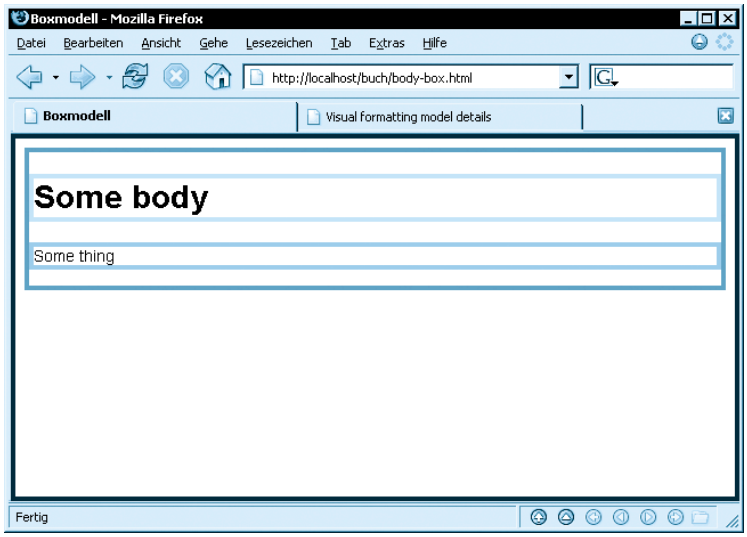


Abbildung 4.26: Boxvisualisierung von Block-Elementen

Erkennbar ist, dass auch die beiden Block-Elemente `h1` und `p` dem Gesetz folgen: so breit wie möglich, so hoch wie nötig. Ebenfalls erkennbar sind jedoch auch diverse Default-Werte, die der Browser zur Darstellung der Elemente verwendet. So gibt es Abstände zwischen den Rahmen der Überschrift und des Textabsatzes, über der Überschrift und unter dem Textabsatz, die nirgendwo in CSS definiert wurden. Es handelt sich hierbei um Default-Abstände, die der Browser bei der Darstellung eines HTML-Dokuments verwendet. Ebenso erkennbar ist, dass der Text der h1-Überschrift zwar links, nicht aber oben und unten die Ränder der Box der Überschrift berührt. Auch das liegt an einer Default-Einstellung des Browsers: Er benutzt einen intern eingestellten Innenabstand zwischen Elementgrenze und Elementinhalt. Besonders bei Elementen wie Tabellenzellen ist dies sinnvoll. Würde der Browser dort keinen Innenabstand darstellen, so würden die Zelleninhalte direkt am Gitternetz der Tabelle kleben.

Die Boxen von Inline-Elementen

Betrachten wir nun noch, wie sich typische Inline-Elemente im Boxmodell verhalten. Dazu zeichnen wir ein Wort in der Überschrift des Beispiels separat mit dem em-Element aus und verpassen diesem mit CSS einen orangefarbenen Rahmen:

```
<h1 style="border:solid 4px lime">Some <em style=" border:solid 4px
orange">body</em></h1>
```

Im Browser wird die zusätzliche Box angezeigt:

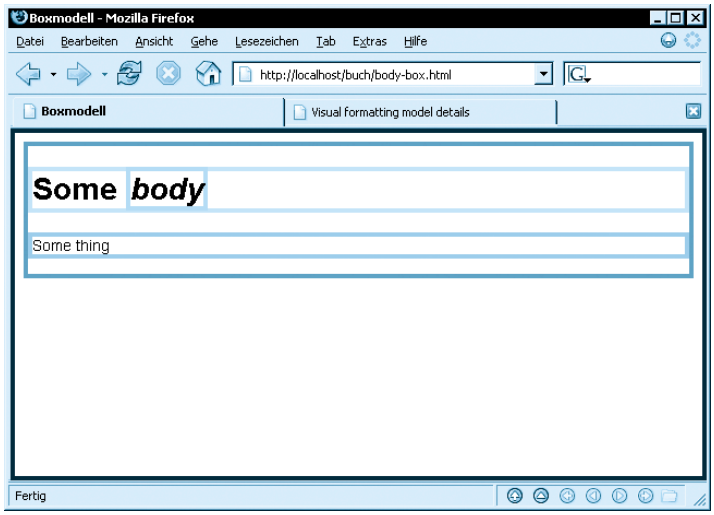


Abbildung 4.27: Boxvisualisierung von Block- und Inline-Elementen

Für Inline-Elemente gilt wie gut erkennbar die Grundregel: so breit wie nötig und so hoch wie nötig.

Boxbestandteile

Es war bereits die Rede von Abständen zu Elternelementen, von Abständen zwischen Elementen und von Innenabständen. Zum Boxmodell gehört auch die Beschreibung, wie diese Bestandteile zusammenwirken. Eine Grafik veranschaulicht dies:

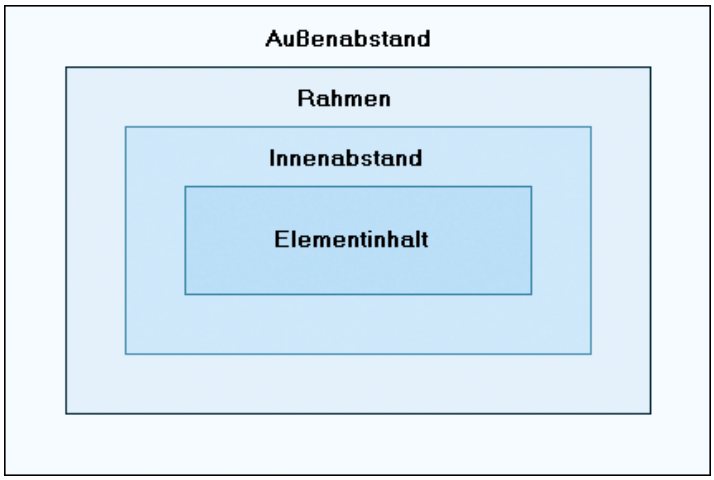


Abbildung 4.28: Bestandteile des CSS-Boxmodells

Die innerste Schicht einer Element-Box ist der ***Elementinhalt***. Das kann – je nach Element und Gegebenheiten – reiner Text sein oder ein bzw. mehrere andere Elemente oder eine Mischung aus reinem Text und anderen Elementen.

Die nächste Schicht ist die des ***Innenabstands***. Der Innenabstand ist der Abstand zwischen dem Elementinhalt und dem ***Rahmen*** des Elements. Dieser bildet die nächste Schicht. Hat ein Element keinen definierten Rahmen, d.h., beträgt die Rahmendicke 0, dann ist die Außengrenze des Rahmens gleich der Außengrenze des Innenabstands.

Die äußerste Schicht bildet der Rand des Elements, d.h. der Abstand zu umgebenden Elementen oder den Elternelementen. Hat ein Element keinen definierten Abstand, d.h., beträgt der Randabstand zu den umgebenden Elementen 0, dann ist die Außengrenze des Rands gleich der Außengrenze des Rahmens und, falls dieser ebenfalls 0 beträgt, gleich der Außengrenze des Innenabstands.

Wenn Sie beispielsweise für ein Element mithilfe von CSS eine Breite von 500 Pixel angeben und außerdem einen Innenabstand des Elementinhalts von 10 Pixel, einen Rahmen von 5 Pixel und einen Außenabstand (Rand) von 30 Pixel bestimmen, so sollte das sichtbare Element insgesamt eine Breite von 500 + (2 * 10) + (2 * 5) Pixel, also 530 Pixel einnehmen. Der Rand geht in die Rechnung nicht mit ein, da er nicht die sichtbare Breite des Elements beeinflusst, sondern nur eine »Lücke« zu umgebenden Elementen darstellt.

An einem Quelltextbeispiel und dessen Auswirkung im Browser wollen wir den Zusammenhang verdeutlichen:

```
<div style="width:500px; padding:10px; border:solid black 5px;
            background-color:rgb(255,255,128)">
   <div style="font-size:32px;
               background-color:rgb(128,128,255);
               color:white">Elementinhalt</div>
</div>
<div style="width:530px; background-color:rgb(255,192,255); font-size:16px;
font-weight:bold">530px</div>
<div style="width:500px; padding:10px; border:solid black 5px;
            background-color:rgb(255,255,128); margin:30px">
   <div style="font-size:32px;
               background-color:rgb(128,128,255);
               color:white">Elementinhalt</div>
</div>
```

Im Beispiel wird mit Bereichen, also div-Elementen gearbeitet. Diese haben den Vorteil, dass sie vom Browser für Außenabstand (Rand), Rahmen und Innenabstand die Default-Werte 0 erhalten und sich damit anzeigeneutral verhalten. Das Beispiel definiert zunächst einen div-Bereich mit einer Breite von 500 Pixel (`width:500px`), einem Innenabstand von 10 Pixel (`padding:10px`) und einem schwarzen durchgezogenen Rahmen von 5 Pixel (`border:solid black 5px`). Innerhalb davon wird ein weiterer div-Bereich notiert. Dieser stellt den Elementinhalt des zuvor definierten Bereichs dar. Beide div-Bereiche erhalten unterschiedliche Hintergrundfarben (`background-color`), damit sie sich optisch gut unterscheiden lassen.

Als Nächstes folgt ein Kontrollbereich. Bei diesem Bereich bleiben alle Werte für Innenabstand, Rahmen und Rand auf dem Default-Wert 0. Der Kontrollbereich bekommt jedoch eine Breite von 530 Pixel (`width:530px`) sowie eine sichtbare Hintergrundfarbe zugewiesen. Damit dient er zur Verifizierung der Berechnung, dass sich beim davor notierten Bereich eine sichtbare Elementbreite von 530 errechnet.

Zuletzt wird noch mal das gleiche div-Konstrukt wie zu Beginn definiert, diesmal jedoch zusätzlich mit 30 Pixel Rand an allen vier Seiten (`margin:30px`). Der folgende Screenshot zeigt die Wirkung im Browser.

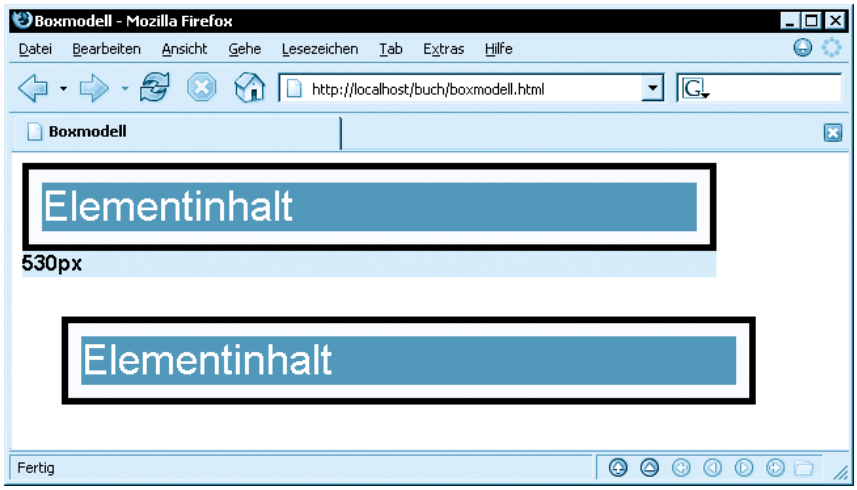


Abbildung 4.29: Boxmodell-Test im Browser

Die Abbildung zeigt, dass unsere Rechnung korrekt war: Obwohl als Breite für den ersten div-Bereich 500 Pixel gewählt wurden, ergibt sich eine Gesamtbreite von 530 Pixel, da die Angaben zu Innenabstand und Rahmen in die Berechnung mit einfließen. Der Kontrollbereich mit dem Textinhalt »530px«, der eine Eigenlänge von 530 Pixel hat, bestätigt dies. Der untere div-Bereich schließlich, der eine Kopie des obersten mit zusätzlich definiertem Rand von 30 Pixel darstellt, ist um 30 Pixel zum Inhalt oberhalb und 30 Pixel von links entfernt. Der Rand ist einfach eine Lücke zwischen den Elementen.

Das Boxmodell, wie es in dieser Form spezifiziert ist, hat Vor- und Nachteile. Vorteile hat es insofern, als eine angegebene Breite nicht mit zusätzlichen Angaben zu Innenabstand und Rahmen kollidiert. D.h. die angegebene Breite steht dem Elementinhalt auf jeden Fall zur Verfügung und die zusätzlichen Angaben führen einfach zu einer faktischen Verbreiterung des Elements. Zum Ärgernis wird dieses Verhalten jedoch, wenn mit einer Angabe wie `width:100%` gearbeitet wird, um z.B. das Browserfenster exakt in der Breite auszufüllen. Soll ein solches Element dann noch einen Innenabstand und/oder Rahmen erhalten, ergibt sich eine tatsächliche Breite, die größer ist als 100%. Der Browser zeigt Quer-Scrollbalken an und die Breite des Inhalts ist etwas breiter als das Anzeigefenster, was nicht sehr elegant ist. Nur mithilfe von Hacks (Tricks) lässt sich dieses Problem umgehen.

Zuletzt soll noch eine Ausnahme beim Boxmodell angesprochen werden: das body-Element. In Abbildung 4.25 ***auf Seite 126*** haben wir die Ausdehnung des body-Elements visualisiert. Dabei hat sich gezeigt, dass das body-Element so viel Höhe einnimmt, wie es sein Inhalt erfordert, und dass die Browser Default-Abstände zwischen Fensterrand und body-Element einfügen. Dennoch wird, wenn man in CSS eine Hintergrundfarbe (`background-color`) für das body-Element definiert, das gesamte Browserfenster eingefärbt. Diese Ausnahme stellt ein Entgegenkommen der Theorie gegenüber der Praxis dar. Denn es musste einfach eine Handhabung her, um eine Seite mit einem Hintergrund zu versehen.

Link zur CSS-Boxmodell-Problematik:

Artikel »Internet Explorer 5.x – 6.x Boxmodell Bug«:
http://www.fabrice-pascal.de/artikel/ie5boxmodel/

4.6.7 CSS-Eigenschaften für Abstände und Ausrichtung

Während die Schriftformatierung alle sichtbaren Elemente betrifft, sind die nachfolgend vorgestellten CSS-Eigenschaften vornehmlich für Block-Elemente gedacht. Zwischen Überschriften und Text sind oft andere als die vom Browser gewählten Default-Abstände oben und unten wünschenswert. Absätze sollen möglicherweise auch mal eingerückt werden und das Standard-Feature der Textausrichtung, also linksbündig, zentriert, rechtsbündig und Blocksatz, wird ebenfalls häufig verlangt.

Abstände zwischen Elementen

Die CSS-Basiseigenschaft zum Definieren von Abständen lautet `margin` (zu Deutsch: Rand). Im Fließtext sind jedoch vor allem die Abstände zwischen aufeinanderfolgenden Block-Elementen interessant, also etwa die Abstände zwischen Überschrift und Textabsätzen oder zwischen zwei Textabsätzen. Deshalb erlaubt es CSS zusätzlich, zwischen den Einzeleigenschaften `margin-top` (Abstand oben), `margin-bottom` (Abstand unten), `margin-left` (Abstand links) und `margin-right` (Abstand rechts) zu unterscheiden.

Gerade bei Abständen zwischen Blockelementen im Fließtext ist jedoch eine Besonderheit zu beachten. Dazu ein Beispiel:

```
<p style="margin-bottom:8px">Ein Textabsatz</p>
<p style="margin-top:5px">Noch ein Textabsatz</p>
```

Vom CSS-Boxmodell her betrachtet, müssten sich die beiden Randangaben einfach addieren, so dass zwischen den beiden Absätzen ein Abstand von 13 Pixel entsteht. Dies ist jedoch nicht der Fall. Im normalen Fließtext kollabieren zwei Angaben zu einer. Maßgeblich ist dabei einfach der höhere Wert. Da dieser im obigen Beispiel 8 Pixel beträgt, bleibt ein tatsächlicher Abstand von 8 Pixel zwischen den beiden p-Elementen.

Das Kollabierungsprinzip gilt jedoch nur für Elemente, die im Textfluss vertikal aufeinander folgen. Für Elemente, die nebeneinander angeordnet sind und die Angaben zu `margin-left` bzw. `margin-right` enthalten, gilt das Prinzip nicht.

Ausrichtung von Inhalten

Block-Elemente, so haben wir bei der Betrachtung des Boxmodells gesehen, nehmen so viel Breite ein wie verfügbar ist. Der Elementinhalt eines solchen Elements kann innerhalb dieser Breite links, zentriert oder rechts ausgerichtet werden. Text kann außerdem als Blocksatz ausgerichtet werden:

Die linksbündige Ausrichtung ist bei den meisten Elementen das eingestellte Default-Verhalten. Eine Ausnahme ist beispielsweise das th-Element für Tabellenkopfzellen, die von den meisten Browsern per Default zentriert ausgerichtet werden.

Mit `text-align:left` erzwingen Sie in jedem Fall eine linksbündige Ausrichtung, mit `text-align:right` eine rechtsbündige Ausrichtung, mit `text-align:center` eine horizontal mittige Zentrierung des Inhalts und mit `text-align:justify` einen Blocksatz.



Abbildung 4.30: Wortabstandprobleme bei Blocksatz und geringer Elementbreite

Da die Browser für HTML keine automatische Silbentrennung anbieten, kann ein erzwungener Blocksatz besonders bei Elementen mit begrenzter Breite zu unschönen Wortzwischenraumlücken führen, wie die Abbildung zeigt. In dem abgebildeten Beispiel wurde definiert:

```
<p style="width:200px; text-align:justify">Lorem ipsum ... </p>
```

Durch die Angabe `width:200px` wird die Breite des p-Elements auf 200 Pixel begrenzt. Ähnliche Breitenverhältnisse ergeben sich aber auch häufig in Tabellenspalten.

Die CSS-Eigenschaft `text-align` bezieht sich stets nur auf den Inhalt eines Blockelements, nicht auf das Element selbst. Falls Sie das Element selbst ausrichten möchten, müssen Sie mit `margin-left` und `margin-right` arbeiten. Dazu ein Beispiel:

```
<p style="width:200px; text-align:right">
   Lorem ipsum ...
</p>
<p style="width:200px; text-align:right; margin-right:0px;
          margin-left:auto">
   Lorem ipsum ...
</p>
```

In dem Beispiel sind zwei Textabsätze notiert. Beide haben eine reduzierte Breite von 200 Pixel und der Inhalt beider Elemente wird rechtsbündig ausgerichtet. Das erste p-Element wird jedoch selbst linksbündig innerhalb der verfügbaren Gesamtbreite ausgerichtet, während das zweite p-Element rechtsbündig ausgerichtet wird. Die folgende Abbildung demonstriert das Ergebnis:

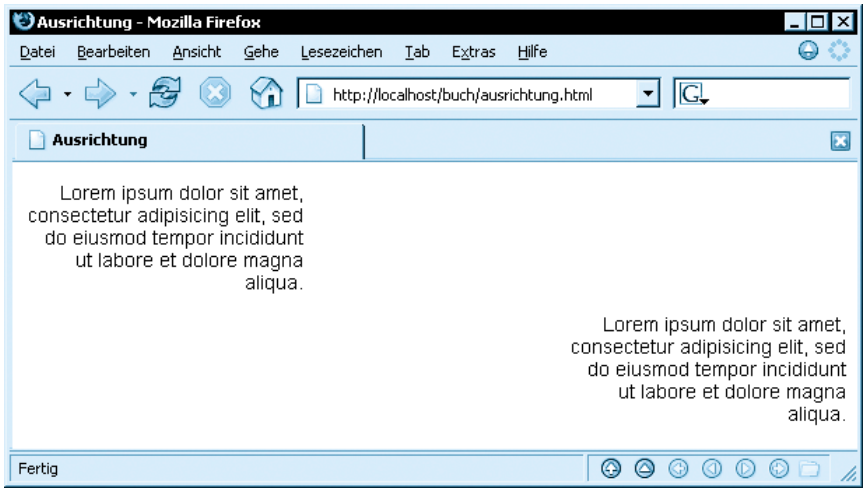


Abbildung 4.31: Ausrichtung von Elementinhalt und Element selbst

Die durch `text-align:right` bewirkte rechtsbündige Ausrichtung des Elementinhalts hält sich in jedem Fall an die mit `width` erzwungene Elementbreite. Die Ausrichtung des Elements selbst wird wie im zweiten p-Element zu sehen durch Angaben zu `margin-left` und `margin-right`, also zu den horizontalen Elementrändern, gesteuert. Die rechtsbündige Elementausrichtung wird durch `margin-right:0px` angewiesen. Doch erst im Zusammenspiel mit der Angabe `margin-left:auto` funktioniert die Ausrichtung tatsächlich wie gewünscht.

Einrückungen

Soll beispielsweise ein einzelner Textabsatz links eingerückt werden, können Sie dies ganz einfach mit `margin-left` erreichen. Weisen Sie einfach den gewünschten linken Rand zu, und das Element wird entsprechend eingerückt dargestellt.

Aus belletristischen Büchern ist auch die Einrückung der ersten Textzeile eines Absatzes bekannt. Für diesen Effekt steht die CSS-Eigenschaft `text-indent` zur Verfügung. Durch eine Angabe wie `text-indent:5mm` rücken Sie die erste Textzeile 5 Millimeter ein. Mit negativen Werten bewirken Sie übrigens eine Ausrückung der ersten Zeile gegenüber den weiteren Zeilen, also z. B. `text-indent:-5mm`.

4.6.8 CSS-Eigenschaften für Farbe und Form

Farben und Formen entstehen in CSS durch Rahmen und durch Gestaltung des Elementhintergrunds.

Rahmen und Innenabstände

Um einen einheitlichen Rahmen für alle vier Seiten eines Elements zu definieren, können Sie mit der CSS-Eigenschaft `border` arbeiten. Es ist aber auch möglich, Rahmen nur für einzelne Seiten zu definieren oder unterschiedlich aussehende Rahmen für einzelne Seiten. Dazu stehen die Eigenschaften `border-left` (Rahmen links), `border-right` (Rahmen rechts), `border-top` (Rahmen oben) und `border-bottom` (Rahmen unten) zur Verfügung.

Alle fünf Eigenschaften stellen Zusammenfassungen von einzelnen untergeordneten Eigenschaften zu Dicke, Farbe und Typ des Rahmens dar. Deshalb sind bei der Wertzuweisung bis zu drei Angaben möglich, die durch Leerzeichen zu trennen sind. Dazu ein Beispiel:

```
<div style="border:1px solid red">...
</div>
<div style="border-top:double rgb(192,192,255) 6px;
            border-bottom:solid rgb(255,192,192)">...
</div>
```

Im Beispiel werden zwei div-Bereiche definiert. Der erste Bereich erhält einen einfach durchgezogenen Rundumrahmen von 1 Pixel Breite (`border:1px solid red`). Beim zweiten Bereich bleiben die Elementseiten links und rechts rahmenlos. Für oben wird eine doppelte hellblaue Linie mit einer Gesamtbreite von 6 Pixel bestimmt (`border-top:double rgb(192,192,255) 6px`) und für unten eine einfache rosafarbene Linie ohne definierte Breite (`border-bottom:solid rgb(255,192,192)`).

Dem Beispiel ist erstens zu entnehmen, dass die Reihenfolge der Angaben bei der Wertzuweisung egal ist – `border:1px solid red` ist also genauso erlaubt wie `border:red solid 1px`. Zweitens geht aus ihm hervor, dass nicht alle drei Angaben zwingend sind. Fehlt eine Angabe, so verwendet der Browser einen Default-Wert.

Ein Rahmen wird vom Betrachter eigentlich nur als solcher empfunden, wenn er rund um das gesamte Element gezogen wird. Werden dagegen nur einzelne Rahmenangaben für

links, rechts, oben oder unten definiert, entsteht optisch eher der Eindruck von Linien. Genau das ist aber ein interessantes Stilmittel zur Auflockerung.

Damit Rahmen bzw. Linien richtig zur Geltung kommen, sind in der Praxis meist zusätzliche CSS-Angaben erforderlich. Vor allem der Innenabstand ist von Bedeutung, um zwischen Elementinhalt und Rahmen einen als angenehm empfundenen Raum zu schaffen. Anhand eines Beispiels wollen wir zeigen, wie das Zusammenspiel von CSS-Angaben zu Rahmenteilen, Innenabständen, Elementabständen und Einrückungen zu ansprechenden Ergebnissen beim Textlayout führen kann.

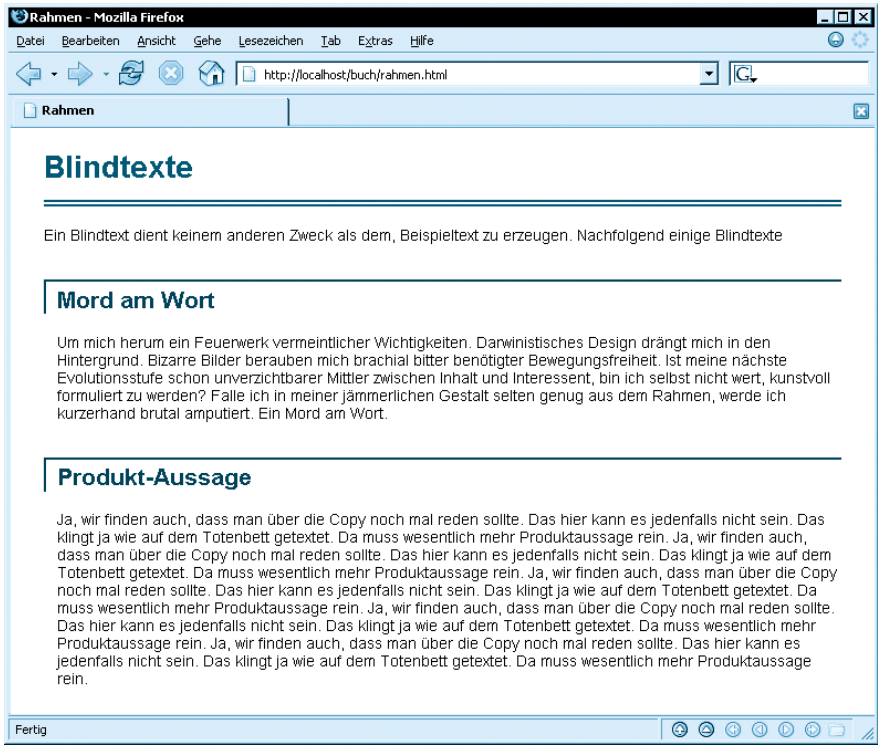


Abbildung 4.32: Textlayout mit Rahmenelementen, Abständen und Einrückungen

Der HTML-Quelltext zur Abbildung lautet:

Listing 4.6: Quelltext einer Seite mit Rahmenelementen, Abständen und Einrückungen

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Rahmen</title>
</head>
<body style="margin-left:30px; margin-right:30px;">
```

```
<h1 style="border-bottom:double green 6px; padding-bottom:12px;
color:green">Blindtexte</h1>

<p>Ein Blindtext dient keinem anderen Zweck als dem, Beispieltext zu erzeugen.
Nachfolgend einige Blindtexte</p>

<h2 style="border-top:solid teal 2px; border-left:solid teal 2px; padding-
left:10px; padding-top:4px; color:teal; margin-top:32px;">Mord am Wort</h2>

<p style="margin-left:12px;">Um mich herum ein Feuerwerk vermeintlicher
Wichtigkeiten.... </p>

<h2 style="border-top:solid teal 2px; border-left:solid teal 2px; padding-
left:10px; padding-top:4px; color:teal;  margin-top:32px;">Produkt-Aussage</h2>

<p style="margin-left:12px;">Ja, wir finden auch, dass man über die Copy noch
mal reden sollte. Das hier kann es jedenfalls nicht sein. Das klingt ja wie auf
dem Totenbett getextet... </p>

</body>
</html>
```

Die Überschriften erhalten in diesem Beispiel Schmucklinien zur optischen Auflockerung. Die h1-Überschrift erhält mit `border-bottom` unterhalb eine grüne und 6 Pixel breite Doppellinie. Die beiden h2-Überschriften erhalten jeweils oben und links (`border-top` und `border-left`) eine einfache, 2 Pixel breite Linie im Farbton »teal«.

Damit die Überschriftentexte nicht zu nah an den Linien kleben, werden jeweils Innenabstände notiert. Die h1-Überschrift erhält mit `padding-bottom` einen Innenabstand unten von 12 Pixel. Da diese Überschrift mit `border-bottom` eine Linie unten erhält, entsteht ein Abstand von 12 Pixel zwischen den tiefsten Unterlängen des Überschriftentextes und der Linie. Die h2-Überschriften erhalten analog zu den Definitionen für `border-left` und `border-top` geeignete Innenabstände mit `padding-left` (Innenabstand links) und `padding-top` (Innenabstand oben).

Die Überschriften bekommen im Beispiel ferner die gleiche Textfarbe (`color`) zugewiesen wie ihre Rahmen. Das unterstreicht optisch den Zusammenhang zwischen Überschriften und Linien.

Eine weitere Maßnahme zur optischen Auflockerung ist die Einrückung der Absätze mit den Blindtexten. Diese werden mit `margin-left:12px` so eingerückt, dass der Textanfang genau an der gleichen Linksposition beginnt wie der Textanfang der zugehörigen h2-Überschriften. Bei den h2-Überschriften errechnet sich die Linksposition des Textanfangs aus den Werten für `padding-left` + `border-left`, also 10 + 2 = 12 Pixel.

Damit die gesamte Seite noch ansprechender wirkt, wurden dem body-Element etwas großzügigere linke und rechte Ränder (jeweils 25 Pixel mit `margin-left` und `margin-right`) spendiert.

Die Wertzuweisungen an die Eigenschaften `border`, `border-left`, `border-top`, `border-bottom` und `border-right` setzen sich wie bereits erwähnt aus den drei Untereigenschaften für Dicke, Typ und Farbe des Rahmens zusammen. Diese Untereigenschaften sind auch mit separaten CSS-Eigenschaften ansprechbar: `border-width` (Rahmendicke), `border-style` (Rahmentyp) und `border-color` (Rahmenfarbe). In Kombination mit den vier Seiten ergeben sich dadurch auch Eigenschaften wie `border-left-width`, `border-top-style` oder `border-bottom-color`.

Beim Rahmentyp bietet CSS eine Reihe von vordefinierten Typen an. So bewirkt die Wertzuweisung `solid` einen durchgezogenen, einfachen Rahmen und `double` einen doppelten Rahmen. Weitere Typen lassen sich durch die Werte `dotted` (gepunktet) und `dashed` (gestrichelt) sowie durch Angaben zu 3D-Effekten wie `groove`, `ridge`, `inset` und `outset` erzwingen. Für den Effekt spielt die Rahmendicke natürlich eine entscheidende Rolle, und vor allem bei den 3D-Effektrahmen auch die Rahmenfarbe.

Hintergrundgestaltung

Jedes im Browser-Fenster sichtbare HTML-Element kann mithilfe von CSS eine eigene Hintergrundgestaltung erhalten – sowohl Block- als auch Inline-Elemente. Möglich sind dabei eine Hintergrundfarbe oder eine Hintergrundgrafik. Bei Hintergrundgrafiken kann es sich um ein so genanntes ***Wallpaper*** handeln, also ein durch Wiederholung entstehendes Muster, oder auch um eine wunschgemäß positionierte Einzelgrafik.

Die CSS-Eigenschaft `background-color` zur Definition einer Hintergrundfarbe haben wir bereits kennen gelernt. Erlaubt ist eine Farbangabe wie in ***Abschnitt 4.6.3*** beschrieben. Achten Sie bei Verwendung von Hintergrundfarben stets auf geeignete, d.h. gut kontrastierende Vordergrundfarben beim Text.

Farben machen Webseiten attraktiv. Aber man kann auch viel verkehrt machen im Umgang damit. Und noch mehr verkehrt machen kann man im Umgang mit Hintergrundgrafiken. In den Frühzeiten der grafischen Browser kamen die Wallpapers für Webseiten in Mode. Grauenhaft grobe Muster erwarteten den Besucher allerortens, auf denen der Text kaum noch lesbar war. Heute stößt man zwar nur noch selten auf solche Seiten, doch gerade Einsteiger lassen sich leicht dazu verleiten, grafisch zu dick aufzutragen.

Unser erstes Beispiel zum Einsatz von Hintergrundgrafiken zeigt, wie ein Wallpaper sinnvoll einsetzbar ist. Ein gern genutzter und ordentlich aussehender Effekt auf Webseiten ist das optische Hervorheben einer Navigationsleiste, indem diese über dem Seiteninhalt zu schweben scheint, da sie einen Schatten auf die Seite wirft. Die Grafik, mit deren Hilfe wir den entsprechenden Effekt bewerkstelligen, sieht so aus (siehe Abbildung 4.33).

Um den gewünschten Effekt zu erzielen, muss die Grafik den Style-Eigenschaften des body-Elements zugewiesen werden, und zwar so, dass sie senkrecht untereinander wiederholt wird, nicht aber waagerecht:

```
<body style="background-image:url(wallpaper.gif); background-repeat:repeat-y;">
```

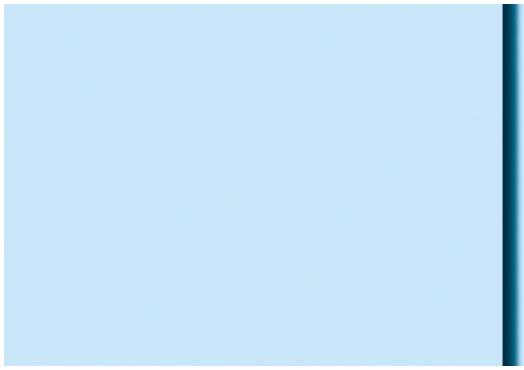


Abbildung 4.33: Wallpaper-Grafik für Navigationsleiste mit Schatteneffekt

Durch `background-image` wird eine Hintergrundgrafik für das betroffene Element eingebunden. Die Wertzuweisung hat das Schema `url(URI)`, wobei URI eine beliebige absolute Internetadresse einer Grafik oder eine lokal referenzierte Grafik sein kann. Im obigen Beispiel liegt die Grafikdatei, also *wallpaper.gif*, im gleichen Verzeichnis wie die HTML-Datei, in der sie eingebunden wird. Die Grafik hat übrigens eine Größe von nur 1,7 Kbyte Größe, ist also selbst bei schwachen Internetverbindungen schnell übertragen.

Die Art, wie die Hintergrundgrafik wiederholt werden soll, wird durch die CSS-Eigenschaft `background-repeat` festgelegt. Der Wert `repeat-y` bewirkt, dass die Grafik nur »eine Spalte lang« wiederholt wird, waagerecht dagegen nicht. Falls Sie eine Grafik nur oben »eine Zeile lang« wiederholen möchten, geben Sie `repeat-x` an. Mit `no-repeat` wird die Hintergrundgrafik nur einmalig angezeigt – ein Beispiel dazu folgt weiter unten. Wenn Sie gar nichts zu `background-repeat` angeben, wird die Voreinstellung `repeat` verwendet. Dabei wird die Hintergrundgrafik sowohl waagerecht als auch senkrecht so oft wiederholt, wie sie in die Ausdehnung des Elements hineinpasst (Tapetenmustereffekt).

Im Browser wird durch unser Beispiel folgender Effekt erzielt:

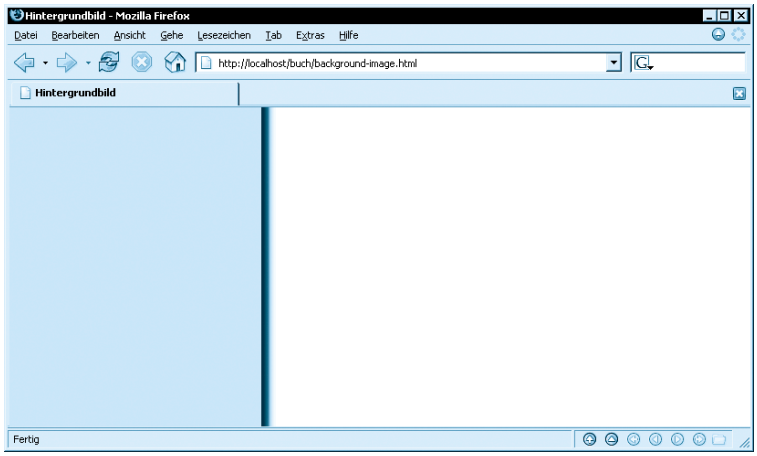


Abbildung 4.34: Wallpaper-Grafik im Browser

Der entstandene optische Bereich lässt sich hervorragend für eine Navigationsleiste verwenden. Der entsprechende HTML-Bereich dafür sollte jedoch genau positioniert werden (mehr dazu in ***Abschnitt 4.6.9***).

Ein zweites Beispiel zum Einsatz von Hintergrundgrafiken zeigt, wie Sie den Seiteninhalt bewusst über die Hintergrundgrafik legen können. Die Hintergrundgrafik ist im Beispiel ein Bild der Sonne als Feuerball. An allen Rändern ist die Grafik schwarz. Dadurch eignet sie sich zur Kombination mit einem schwarzen Seitenhintergrund.

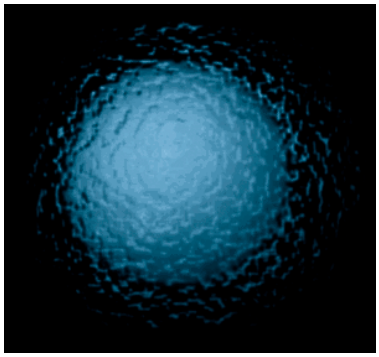


Abbildung 4.35: Sonnenbild für Hintergrundgrafik

Diese Hintergrundgrafik binden wir wieder ins `<body>`-Tag ein:

```
<body style="background-image:url(sonne.jpg); background-repeat:no-repeat;
background-color:black">
```

Die Hintergrundgrafik wird mit `background-image` eingebunden. Dass sie nur einmal angezeigt wird, wird durch `background-repeat:no-repeat` sichergestellt. Den Seitenhintergrund setzen Sie durch `background-color:black` auf Schwarz. Auf diesem Hintergrund platzieren wir nun eine passende, effektvolle Überschrift mit folgender Formatierung:

```
<h1 style="margin-top:100px; margin-left:130px;
           margin-right:40px; font-size:36px; color:white;
           border-top:rgb(239,82,0) dotted 4px;
           border-bottom:rgb(239,82,0) dotted 4px">
   Die Sonne - Licht und Leben
</h1>
```

Die Überschrift wird mithilfe von `margin-top` (Rand oben) und `margin-left` (Rand links) so positioniert, dass sie über der Sonnenmitte beginnt. Weiterhin erhält die Überschrift eine weiße Textfarbe (`color`), eine auffällige Schriftgröße von 36 Pixel (`font-size`) und zwei 4 Pixel dicke, gepunktete Rahmenlinien oberhalb und unterhalb (`border-top` bzw. `border-bottom`).

Wenn eine Hintergrundgrafik nur einmalig angezeigt wird, wie im Beispiel, so bedeutet dies: links oben an Position 0,0, gemessen von den Elementgrenzen. Mitunter ist es jedoch

sinnvoll, die genaue Position zu beeinflussen. Dazu dient die CSS-Eigenschaft `background-position`. Durch eine Angabe wie `background-position:30px 20px` erzwingen Sie eine Verschiebung um 30 Pixel zur linken Elementgrenze und 20 Pixel zur oberen Elementgrenze. Ferner stehen die relativen Wertzuweisungen `top`, `left`, `bottom`, `right` und `center` zur Verfügung. Je zwei sinnvolle Kombinationen davon ergänzen sich. So bewirkt die Angabe `background-position:top center` eine an der Elementbreite gemessen zentrierte und gemessen an der Elementhöhe obenbündige Ausrichtung der Hintergrundgrafik. Die Angabe `background-position:bottom right` richtet die Hintergrundgrafik rechts unten an den Elementgrenzen aus und `background-position:center center` positioniert die Hintergrundgrafik sowohl horizontal als auch vertikal mittig in der Elementbox. Im letzteren Fall würde übrigens auch eine einfache Angabe von `center` genügen. Wenn `background-position` nur ein Wert zugeordnet wird, gilt dieser sowohl für »von links« als auch für »von oben«.



Abbildung 4.36: Hintergrundbild und passender Vordergrund

Ein drittes Beispiel soll einen weiteren, in der Praxis häufig genutzten Effekt demonstrieren: ein Logo oder Schriftzug, der beim Scrollen von längeren Inhalten fix an seiner Stelle stehen bleibt. Früher wurden häufig Frames verwendet, nur um diesen Effekt zu erreichen. Glücklicherweise ist dies heute nicht mehr nötig. Folgendes Bild soll fix im Browser-Fenster stehen bleiben (siehe Abbildung 4.37).

Ein Komplett-Listing zeigt die weitere Vorgehensweise:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Hintergrundbild</title>
</head>
<body style="background-image:url(logo.jpg);
             background-repeat:no-repeat;
             background-attachment:fixed;
```

```
            background-position:10px 10px;
            background-color:rgb(251,240,204);">
    <div id="Seiteninhalt" style="margin-left:105px;
      margin-right:25px; margin-top:25px;">
       <h1 style="font-size:48px; font-weight:normal;
                  color:rgb(128,0,0);
                  font-family:'Brush Script MT',cursive">
          Herbstkollektion 2005
       </h1>
    </div>
</body>
</html>
```



Abbildung 4.37: Logo/Schriftzug für fixe Positionierung

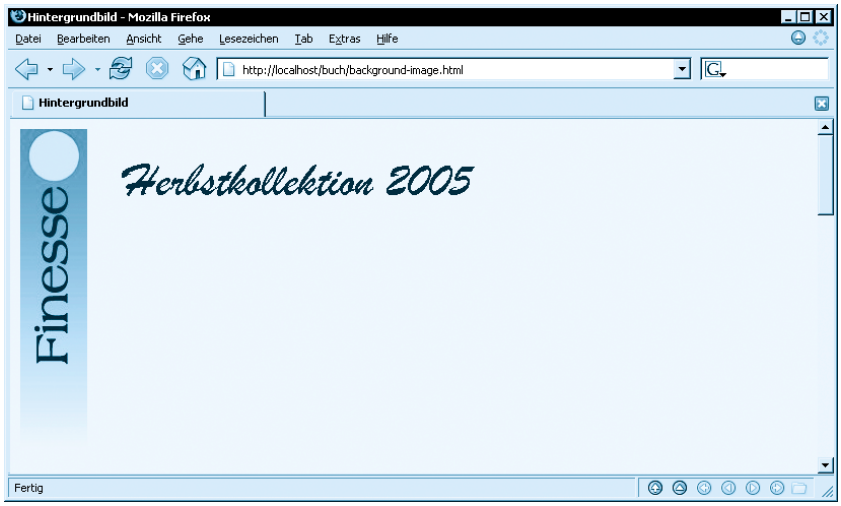


Abbildung 4.38: Hintergrundgrafik, die nicht mit gescrollt wird

Wie auch in den Beispielen zuvor wird die Hintergrundgrafik mit `background-image` ins body-Element eingebunden. Der Effekt, dass die Grafik beim Scrollen in längeren Seiteninhalten fix an ihrer Position stehen bleibt, wird durch die zusätzliche Angabe `background-`

`attachment:fixed` erreicht. Das Normalverhalten, also Scrollen mit dem Seiteninhalt, kann durch `background-attachment:scroll` erzwungen werden.

In unserem Beispiel haben wir die Grafik durch `background-position:10px 10px` etwas vom linken oberen Rand abgesetzt. Ferner definierten wir mit `background-color` eine Hintergrundfarbe für die Seite, die exakt der hellsten Farbe am unteren Rand der Grafik entspricht. So entsteht ein fließender Übergang, der gar nicht mehr erkennen lässt, wo genau die Grafik ihre untere Grenze hat.

Der Hintergrund eines HTML-Elements lässt sich also über Angaben zu den Einzeleigenschaften `background-color`, `background-image`, `background-position`, `background-repeat` und `background-attachment` steuern. All diese Angaben können jedoch auch in der Sammeleigenschaft `background` notiert werden. Dazu ein Beispiel:

```
<div style="background:url("background.png") gray 50% no-repeat fixed">...</div>
```

In diesem Beispiel wird für einen div-Bereich eine dunkelgraue (`gray`) Hintergrundfarbe festgelegt. Als Hintergrundgrafik soll die Datei ***background.png*** angezeigt werden. Die Position der Hintergrundfarbe wird mit `50%` angegeben. Das bedeutet: sowohl horizontal als auch vertikal zentriert, gemessen an den Ausmaßen der Elementbox. Die Hintergrundgrafik soll nur einmalig angezeigt werden (`no-repeat`) und sie soll nicht mitscrollen (`fixed`). Da solche Sammelangaben jedoch leicht unübersichtlich wirken, sind die Einzelangaben zumindest in Hinblick auf die Lesbarkeit empfehlenswerter.

4.6.9 CSS-Eigenschaften für die Positionierung von Elementen

Mit den bisher vorgestellten Möglichkeiten von CSS beeinflussen wir nicht den normalen Textfluss eines HTML-Dokuments, im CSS-Fachjargon als ***normal flow*** bezeichnet. In diesem Abschnitt werden wir nun unter anderem CSS-Eigenschaften kennen lernen, die den normalen Textfluss außer Kraft setzen oder manipulieren. Dadurch ergeben sich Möglichkeiten, die vor allem beim Design von Seitenlayouts mit CSS zum Tragen kommen.

Kontrolle von Breite und Höhe

Block-Elemente im normalen Textfluss nehmen so viel Breite ein wie möglich und so viel Höhe wie nötig. Dieses Default-Verhalten lässt sich auf mehrfache Weise manipulieren:

- Breite und Höhe können vorgegeben werden.
- Spielraum kann durch Angabe von Mindestbreite, Maximalbreite, Mindesthöhe und Maximalhöhe definiert werden.
- Für Inhalte, die größer sind als die Vorgaben zu Breite und/oder Höhe, lässt sich eine Konfliktlösung festlegen (z.B. Inhalte einfach abschneiden oder Inhalte scrollen).
- In Verbindung mit anderen Angaben lassen sich Elemente mit definierter Breite ausrichten.

Breite und Höhe eines Block-Elements können durch die CSS-Eigenschaften `width` (Breite) und `height` (Höhe) festgelegt werden. Ein einfaches Beispiel:

```
<p style="width:300px; height:150px; background-color:aqua"></p>
```

Das normale Elementverhalten wird dadurch aufgehoben. Selbst wenn das Element keinerlei Inhalt hat, so wie im Beispiel, wird eine Box mit den für Breite und Höhe angegebenen Ausmaßen dargestellt:

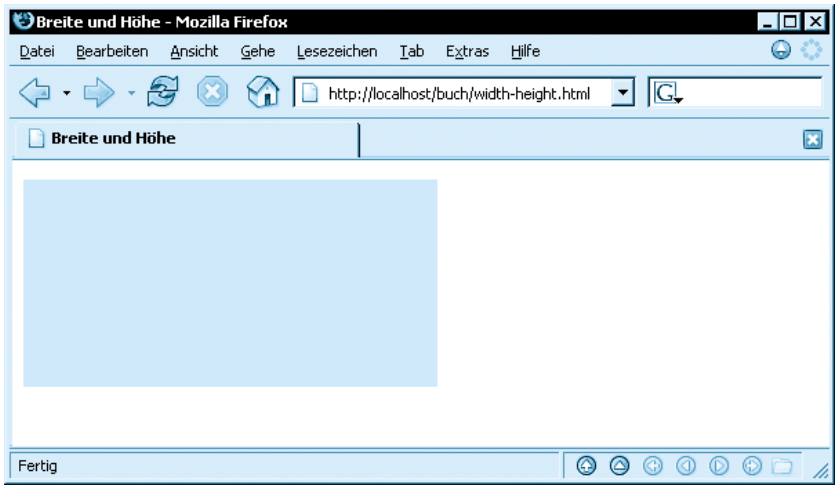


Abbildung 4.39: p-Element mit definierter Breite und Höhe

Die angegebene Hintergrundfarbe macht die Box des p-Elements sichtbar.

Füllt man eine solche Box mit Inhalt, werden die Angaben zu `width` und `height` jedoch nur so lange eingehalten, wie der Inhalt hineinpasst. Ist der Inhalt umfangreicher, wird die Box ausgedehnt. Reiner Text beispielsweise passt in der Breite durchaus in die Box, aber wenn er in der Länge mehr Raum benötigt, als Höhe angegeben ist, wird die Box in der Höhe einfach ausgedehnt und angepasst. Das Gleiche passiert mit der Breite, falls der Inhalt nicht so umgebrochen werden kann, dass er die Breite einhält. Das ist beispielsweise der Fall, wenn der Inhalt aus einer Grafik besteht, deren Breite größer ist, oder wenn der Inhalt eine Zeichenfolge ohne Whitespace-Zeichen enthält, die mehr Breite einnimmt.

Dies ist jedoch ebenfalls nur ein Default-Verhalten, das Sie ändern können. Dazu dient die Eigenschaft `overflow`.

Durch Angabe von `overflow:hidden` erreichen Sie, dass die Boxgrößen von `width` und `height` eingehalten werden und dass übergroßer Inhalt einfach abgeschnitten wird.

Für manche Fälle interessant ist jedoch auch das Angebot an den Anwender, übergroße Inhalte zu scrollen. Dazu dient die Angabe `overflow:scroll`.

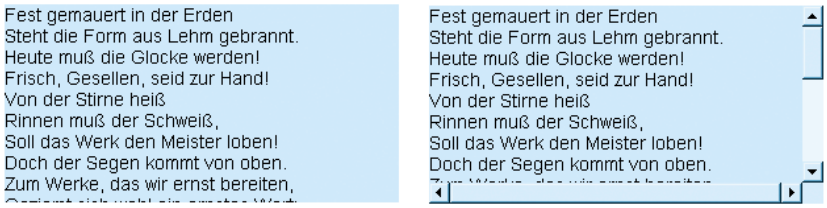


Abbildung 4.40: Beeinflussung übergroßer Inhalte durch die overflow-Eigenschaft

Die Angaben zu Breite und Höhe können auch prozentual sein. Interpretiert werden sie dann jedoch nur, wenn es ein Elternelement gibt, dessen absolut festgelegte oder errechnete Breite den Maßstab für die Prozentangabe bildet. Wenn Sie also beispielsweise einem p-Element im normalen Textfluss `width:70%` zuweisen, dann gilt die Breite des body-Elements, innerhalb dessen das p-Element vorkommt, als Bezug. Kommt das p-Element dagegen innerhalb eines div-Bereichs vor, dessen Breite mit `width:400px` festgelegt wurde, dann beziehen sich die 70% auf diese Breite.

Sie können auch Werte für `min-width` bzw. `max-width` und `min-height` bzw. `max-height` notieren. Wenn die angegebene oder verfügbare Breite größer ist als `min-width`, hat diese Eigenschaft gar keine Wirkung – das Element wird dann so breit wie angegeben oder möglich. Eine Wirkung hat die Angabe nur dann, wenn die angegebene oder verfügbare Breite kleiner ist. Analog verhält es sich bei `min-height`. Leider werden die Mindest- und Maximalangaben zu Breite und Höhe nicht von allen Browsern interpretiert. Von daher ist bei ihrer Verwendung Vorsicht angebracht.

Absolute, relative und fixe Positionierung

Durch Angaben zur Positionierung von Elementen heben Sie den normalen Textfluss wie folgt auf:

- *Relative Positionierung* positioniert eine Elementbox relativ zu der Position, die sie im normalen Textfluss hat.
- *Absolute Positionierung* positioniert eine Elementbox entweder relativ zu einem ebenfalls positionierten Elternelement, oder, falls es kein positioniertes Elternelement gibt, absolut im Dokument.
- *Fixe Positionierung* positioniert eine Elementbox in jedem Fall absolut im Dokument, und zwar so, dass das Element beim Scrollen nicht mitscrollt.

Zunächst ein Beispiel zur relativen Positionierung:

```
<p style="background-color:aqua">ABSATZ</p>
<p style="background-color:aqua">ABSATZ</p>
<p style="background-color:aqua; position:relative; left:100px;
top:8px">ABSATZ</p>
<p style="background-color:aqua">ABSATZ</p>
<p style="background-color:aqua">ABSATZ</p>
```

Zur besseren Visualisierung des Effekts, den die relative Positionierung bewirkt, werden fünf Textabsätze definiert. Allen wird eine Hintergrundfarbe zugewiesen, damit ihre Elementbox sichtbar ist. Der mittlere Absatz wird mit `position:relative` relativ positioniert. Wo genau er positioniert werden soll, wird im Beispiel durch die Eigenschaften `left` (links) und `top` (oben) festgelegt.

Die Angaben zu `left` und `top` bewirken in Verbindung mit der relativen Positionierung, dass die Elementbox im Beispiel 100 Pixel weiter rechts und 8 Pixel weiter unten beginnt als sie es normalerweise tun würde.

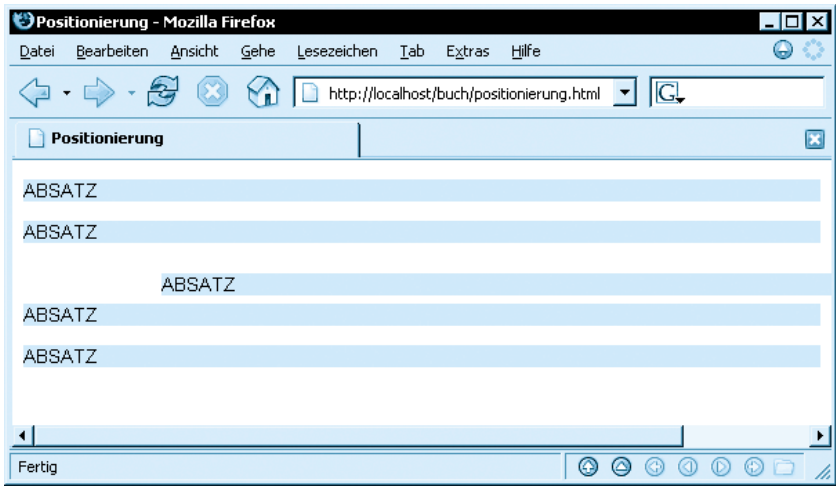


Abbildung 4.41: Relative Positionierung eines Textabsatzes

Im Ergebnis fällt auf, dass der Browser die Verschiebung des Elements nach unten bzw. rechts wie definiert ausführt. Die Elementbreite wird jedoch nicht angepasst. Das p-Element nimmt weiterhin die gesamte verfügbare Breite ein, die es im normalen Textfluss einnehmen würde. Das bedeutet im Klartext, dass der Anwender möglicherweise quer scrollen muss, um alle Inhalte des Elements zu sehen. Wenn Ihnen dieses Verhalten unerwünscht erscheint, dann könnten Sie es im Beispiel durch folgende Erweiterung korrigieren:

```
<p style="background-color:aqua; position:relative; left:100px; top:8px; margin-
right:100px">ABSATZ</p>
```

Durch `margin-right:100px` wird hier ein Gegengewicht zu `left:100px` geschaffen. Die Elementbreite verkürzt sich rechts um so viele Pixel, wie es links durch die relative Positionierung verschoben wird. Seine rechte Elementgrenze ist nun wieder bündig zu denen der übrigen Absätze.

Um die absolute Positionierung besser zu verstehen, greifen wir auf das Beispiel von Abbildung 4.34 ***auf Seite 138*** zurück. Dort wurde bereits darauf hingewiesen, dass sich die durch die Hintergrundgrafik entstandene linke Leiste hervorragend für eine Navigationsleiste eignet. Wir werden nun einige sichtbare Boxen in dieser Leiste positionieren. Auf die

Navigations-Links verzichten wir an dieser Stelle noch und notieren an ihrer Stelle nur die Linkbeschriftungen.

```
<body style="background-image:url(wallpaper.gif); background-repeat:repeat-y;">
   <div style="position:absolute; left:20px; top:20px;
               width:230px">
      <div style="position:absolute; left:0px; top:0px;
                  width:100%; height:20px; padding:2px;
                  background-color:olive; color:white;
                  border:white solid 1px; font-size:16px;
                  font-weight:bold; text-align:center">
         HOME</div>
      <div style="position:absolute; left:0px; top:30px;
                  width:100%; height:20px; padding:2px;
                  background-color:olive; color:white;
                  border:white solid 1px; font-size:16px;
                  font-weight:bold; text-align:center">
         Produkte</div>
      <div style="position:absolute; left:0px; top:60px;
                  width:100%; height:20px; padding:2px;
                  background-color:olive; color:white;
                  border:white solid 1px; font-size:16px;
                  font-weight:bold; text-align:center">
         Feedback</div>
   </div>
</body>
```

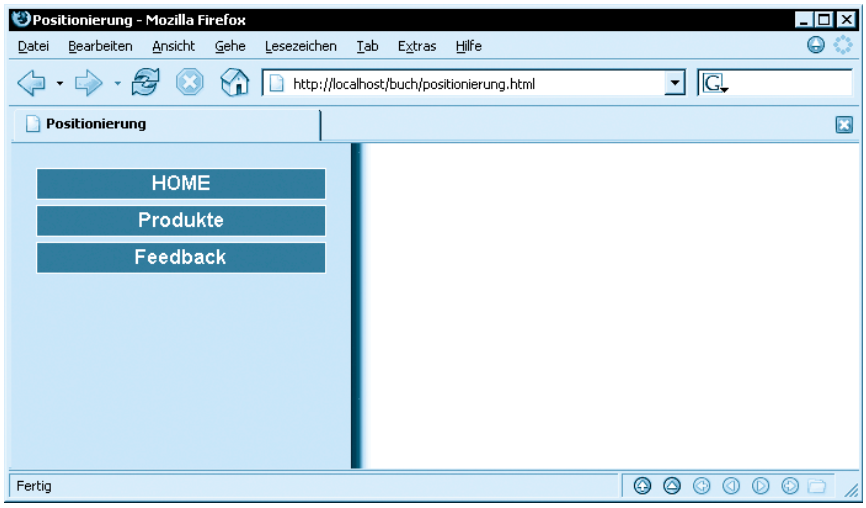


Abbildung 4.42: Absolut positionierte Elemente vor einer Hintergrundgrafik

Innerhalb des body-Bereichs werden insgesamt vier div-Bereiche definiert. Einer davon enthält die anderen drei als Elementinhalt. Alle Bereiche werden mit `position:absolute` absolut positioniert. Dabei kommt die Regel für absolutes Positionieren zum Tragen: Der Bezug für den äußeren div-Bereich ist das Dokument bzw. das Browserfenster. Die Angaben `left:20px` und `top:20px` bedeuten bei diesem Bereich also den Offset gegenüber der linken oberen Ecke des Anzeigefensters. Bei den drei inneren Bereichen ist es anders. Da der äußere Bereich selbst positioniert ist, stellt er für die inneren Bereiche den Bezugspunkt dar. Die Angaben `left:0px` und `top:0px` beim ersten inneren Bereich, der Box mit dem Textinhalt »HOME«, bewirken also, dass die linke obere Ecke dieses Elements beim Offset 20,20 beginnt. Denn das ist die linke obere Ecke des positionierten Elternelements, und der Offset 0,0 muss relativ zu diesem Offset betrachtet werden.

Die drei inneren Bereiche unterscheiden sich in ihren CSS-Definitionen nur durch den Wert bei `top` (jeweils 30 Pixel mehr pro Bereich). Dadurch werden die Bereiche sauber untereinander positioniert.

Der äußere div-Bereich gibt übrigens auch die Breite vor. Dort wird mit `width:230px` eine Breite festgelegt, die optimal auf das Hintergrundbild passt, das für die schattierte Fläche am linken Fensterrand verantwortlich ist. Die inneren Bereiche erhalten nur die Angabe `width:100%`, um diesen Bereich komplett auszufüllen. Man könnte meinen, dass diese Angabe überflüssig sei, weil Block-Elemente ja normalerweise die gesamte zur Verfügung stehende Breite einnehmen. Das gilt jedoch nur, wenn sich die Elemente im normalen Textfluss befinden. Positionierte Elemente haben in dieser Hinsicht keinen Bezugspunkt und werden deshalb nur so breit dargestellt, wie es ihr Inhalt erfordert – es sei denn, man definiert explizit eine Breite.

Noch einen Schritt weiter als die absolute Positionierung geht die fixe Positionierung. Dieses Feature ist so mächtig, dass es locker 90% aller Frameset-basierten Seiten im Web überflüssig macht. Denn meistens werden Frames nur genutzt, um so genannte ***non scrolling regions*** zu realisieren, z. B. für Firmenlogos oder Werbebanner, die beim Scrollen von längeren Inhalten im Browserfenster stehen bleiben. Leider wird die fixe Positionierung vom MS Internet Explorer 6.0 noch nicht unterstützt, weswegen die Akzeptanz bisher gering ist. Es existieren zwar Workarounds, um auch den Microsoft-Browser dazu zu bewegen, Bereiche fix zu positionieren, doch ist dies mit einiger Trickserei verbunden, was keine wirklich befriedigende Lösung ist.

Das folgende Beispiel definiert im oberen Anzeigefensterbereich eine waagerechte non scrolling region mit Logo:

```
<body style="background-color:rgb(251,240,204); margin:0">
   <div style="position:fixed; top:0px; left:0px; width:100%;
               height:66px; text-align:right;
               border-bottom:rgb(244,42,42) solid 2px;
               padding:0px; background-color:rgb(251,240,204);
               z-index:2"><img src="logo.jpg"
                  style="width:315px; height:66px; border:none"
                  alt="Logo">
   </div>
```

```
    <div style="position:absolute; top:85px; margin-left:10%;
                margin-right:10%; ; z-index:1">
       <h1 style="font-size:48px; font-weight:normal;
                  color:rgb(128,0,0);
                  font-family:'Brush Script MT',cursive">
          Herbstkollektion 2005
       </h1>
    </div>
 </body>
```



Abbildung 4.43: Fix positionierter Kopfbereich

Der Quelltext dieses Beispiels führt bei längeren Dokumentinhalten dazu, dass Browser, welche die Eigenschaft/Wert-Kombination `position:fixed` korrekt interpretieren, den Bereich mit Logo und roter Linie nicht mitscrollen. Der nach oben wegscrollende Text verschwindet unterhalb des Kopfbereichs. Die Abbildung im Beispiel zeigt, dass der Mozilla-Browser, der die fixed-Eigenschaft interpretiert, zwar den vertikalen Scrollbalken über die gesamte Fensterhöhe anzeigt. Dennoch bleibt beim Scrollen der Logobereich fest eingeblendet. Der MS Internet Explorer zeigt den Inhalt genauso an, scrollt jedoch den Kopfbereich mit.

Die Angaben `z-index:2` und `z-index:1` sorgen übrigens dafür, dass der untere Bereich beim Scrollen unter dem oberen verschwindet, weil er die niedrigere z-index-Nummer hat. Wir werden später noch genauer auf die z-index-Eigenschaft eingehen.

Umflusskontrolle

Ein optisch ansprechendes Gestaltungsmerkmal, bekannt aus Büchern und Zeitschriften, sind Grafiken, um die der Text herumfließt, oder auch in den Text eingeschobene und von ihm umflossene Überschriften, Infokästen oder Merksprüche. CSS stellt Mittel bereit, um solche Effekte auch auf Webseiten zu zaubern.

Beginnen wir mit einer Überschrift, die in den Text eingeschoben wird:

Der zugehörige Quelltext lautet:

```
<h1 style="width:150px; float:left; padding-top:0; margin-top:0;
           padding-bottom:0; margin-bottom:10px">
   Blindtext
</h1>
<p>
   Lorem ipsum ...
</p>
```

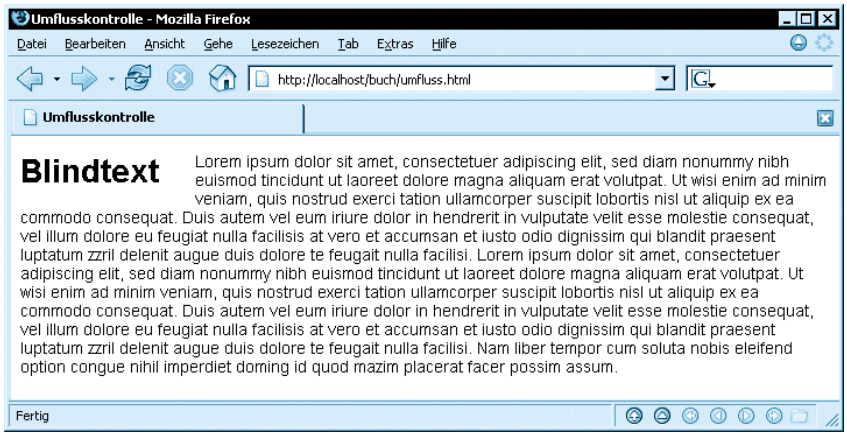


Abbildung 4.44: In den nachfolgenden Text eingeschobene Überschrift

Der Effekt kommt durch die Kombination der CSS-Eigenschaften `float` und `width` zustande. Durch `width` wird die Anzeigebreite des h1-Elements begrenzt. Mit `float:left` wird angewiesen, dass nachfolgende Elemente im Textfluss um das so definierte Element herumfließen. Die Angabe `float:right` wäre ebenfalls zulässig. In diesem Fall würde die Überschrift rechts außen ausgerichtet und der nachfolgende Text würde links darum fließen. Die übrigen CSS-Definitionen im Beispiel zu Abstand und Innenabstand sorgen für die Feinjustierung des Überschriftentextes.

Eigentlich ist in CSS 2.1 eine andere Angabe vorgesehen, um obigen Effekt zu erzielen, nämlich `display:run-in`. Doch leider interpretieren die Browser diese Wertzuweisung noch nicht, weshalb noch davon abzuraten ist.

Wichtig in Zusammenhang mit der float-Eigenschaft ist auch die Eigenschaft `clear`. Ein weiteres Beispiel soll dies demonstrieren:

```
<body style="background-color:rgb(251,240,204)">
   <h1 style="font-size:48px; font-weight:normal;
              color:rgb(128,0,0);
              font-family:'Brush Script MT',cursive">
      <img src="logo.jpg" style="width:66px; height:315px;
           border:none; float:left; margin-right:20px"
```

```
            alt="Logo">
         Herbst<br>Kollektion<br>2005
      </h1>
      <p style="clear:left">
         In diesem Herbst dominieren frohe und kräftige Farben.
      </p>
   </body>
```



Abbildung 4.45: Abbruch des Textumflusses und Fortsetzung unterhalb

In diesem Beispiel wird eine Grafik (img-Element) vom nachfolgend notierten Text umflossen. Im <img>-Tag der Grafik ist zu diesem Zweck die Eigenschaft `float:left` notiert. Da die Grafik eine gewisse Höhe hat, würde auch noch der im Anschluss an die h1-Überschrift notierte Textabsatz rechts um die Grafik fließen. Im einleitenden <p>-Tag des Absatzes sorgt jedoch `clear:left` dafür, dass die Wirkung von `float:left` aufgehoben wird. Für den Textabsatz gilt damit wieder der normale Textfluss, was bedeutet, dass er unterhalb der Grafik platziert wird.

Durch `clear:left` heben Sie eine `float:left`-Definition auf, durch `clear:right` eine `float:right`-Definition. Falls sowohl `float:left` als auch `float:right` »aktiv« sind, können Sie durch `clear:both` beide Umflussanweisungen aufheben.

Eine weitere wichtige CSS-Eigenschaft, die unter anderem zur Textflusskontrolle eingesetzt werden kann, ist `display`. Mithilfe dieser Eigenschaft können Sie einem Element explizit zuweisen, ob es sich als Block- oder als Inline-Element verhalten soll. Damit ist es möglich, die Block- und Inline-Defaults von HTML außer Kraft zu setzen.

Dem sichtbaren Text dieser Abbildung liegt folgender Quelltext zu Grunde:

```
<div>
<h1 style="display:inline; margin-right:10px;
           vertical-align:text-bottom">
   Blindtext:
</h1>
Lorem ipsum ...
</div>
```

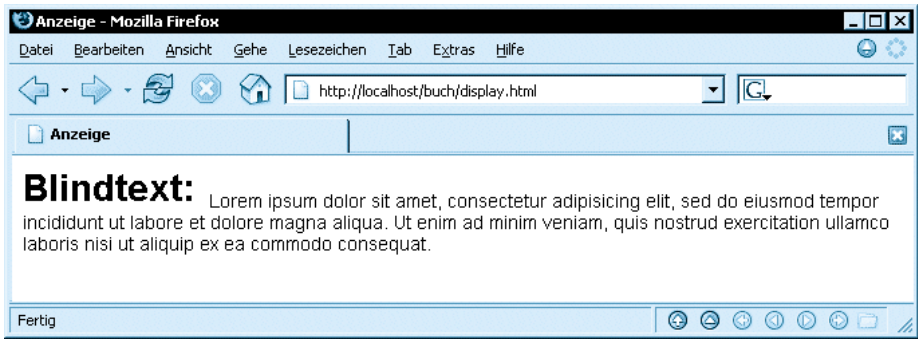


Abbildung 4.46: h1-Überschrift als Inline-Element

Das gesamte Konstrukt ist also in einen div-Bereich eingebettet, damit es sich in einem Block-Element befindet, was von striktem HTML gefordert wird. Aus der h1-Überschrift wird durch die Definition `display:inline` ein Inline-Element. Der nachfolgende Text beginnt einfach in der gleichen Zeile direkt hinter der rechten Elementgrenze der Überschrift. Damit der nachfolgende Text nicht unmittelbar an der Überschrift klebt, erhält diese im Beispiel mit `margin-right:10px` etwas Abstand rechts.

Bemerkenswert ist im Beispiel auch die Angabe `vertical-align:text-bottom`. Mit der vertical-align-Eigenschaft können Sie nebeneinander stehende und gegeneinander ausgerichtete Inhalte unterschiedlicher Höhe vertikal ausrichten. Würde diese Angabe im Beispiel fehlen, so würde die erste Zeile des nachfolgenden Textes zur zweiten Zeile mehr Zeilenabstand haben als die nachfolgenden Textzeilen untereinander, weil die Schriftgröße der h1-Überschrift den Zeilenabstand beeinflusst. Auf die Eigenschaft `vertical-align` werden wir in Zusammenhang mit der CSS-Formatierung von Tabelleninhalten noch genauer eingehen.

Arbeiten mit Layern, Sichtbarkeit und Transparenz

Dank absoluter Positionierung in Verbindung mit Angaben zu Breite und Höhe ist es möglich, Elementboxen frei anzuordnen. Selbstverständlich ist es auch möglich, zwei Elemente an der gleichen Stelle anzuordnen. Untersuchen wir, was dabei passiert:

```
<div style="position:absolute; top:20px; left:60px; width:400px;
            height:100px; border:solid 2px blue">
   <h1>Erster Bereich</h1>
```

```
</div>
<div style="position:absolute; top:20px; left:60px; width:400px;
            height:100px; border:solid 2px red">
   <h1>Zweiter Bereich</h1>
</div>
```

Es werden zwei div-Bereiche definiert. Beide erhalten identische Angaben zur absoluten Position. Sie unterscheiden sich lediglich in der Rahmenfarbe (der erste Bereich hat einen blauen Rahmen und der zweite einen roten) sowie im Elementinhalt (beide enthalten eine h1-Überschrift, jedoch mit unterschiedlichem Text).

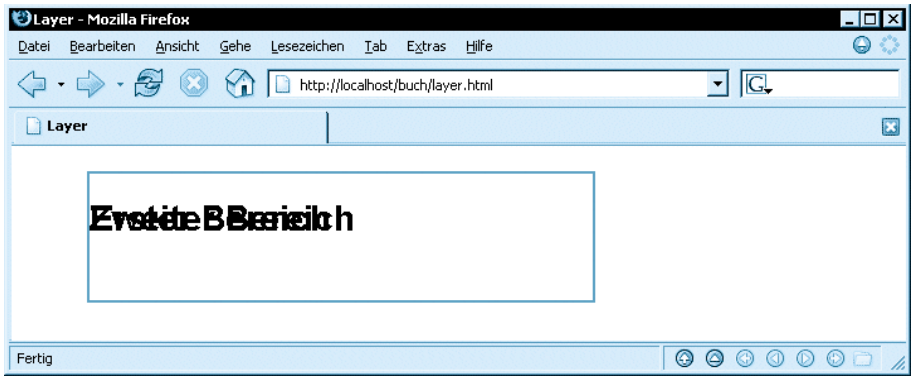


Abbildung 4.47: Zwei Bereiche an gleicher Position

Die Anzeige des Beispiels im Browser offenbart Folgendes:

Der zweite Bereich liegt über dem ersten, was dadurch sichtbar wird, dass die Rahmenfarbe Rot ist. Denn dies ist die Rahmenfarbe des zweiten Bereichs. Der Inhalt ist dagegen transparent, was daran erkennbar ist, dass beide Überschriftentexte sichtbar sind. Es sind also folgende Regeln erkennbar:

- Bei positionierten Elementboxen, die sich überlappen, ist die natürliche Schichtreihenfolge durch die Reihenfolge der Notation im Dokument vorgegeben. Ein hinter einem anderen Element notiertes Element wird über diesem angeordnet.
- Der Inhalt von Elementboxen ist transparent.

Mithilfe geeigneter CSS-Definitionen lässt sich dieses Default-Verhalten ändern. Die Schichtreihenfolge von Elementen können Sie durch die Eigenschaft `z-index` beeinflussen. Dazu bekommen alle Elemente, die sich überlappen, eine z-index-Eigenschaft zugewiesen. Als Wert wird jeweils eine frei wählbare Zahl angegeben. Das Element mit der niedrigsten Zahl wird dann in der untersten Schicht angeordnet, also von allen übrigen Elementen überlappt, und das Element mit dem höchsten z-index-Wert überlappt alle anderen Elemente. In unserem Beispiel können wir das einfach überprüfen, indem wir dem zweiten div-Bereich einen höheren z-index-Wert zuordnen als dem ersten:

```
<div style="position:absolute; top:20px; left:60px; width:400px;
            height:100px; border:solid 2px blue; z-index:2">
   <h1>Erster Bereich</h1>
</div>
<div style="position:absolute; top:20px; left:60px; width:400px;
            height:100px; border:solid 2px red; z-index:1">
   <h1>Zweiter Bereich</h1>
</div>
```

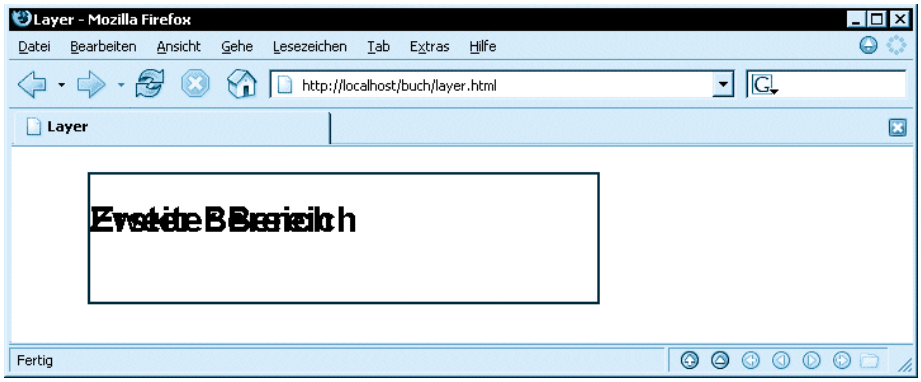


Abbildung 4.48: Schichtenkontrolle mit der z-index-Eigenschaft

Der sichtbare Layer ist nun das div-Element mit dem blauen Rahmen, obwohl dieses als Erstes notiert wurde. Grund ist, dass es mit `z-index:2` einen höheren Schichtwert erhalten hat als der zweite Bereich, dem `z-index:1` zugewiesen wurde.

Die Default-Transparenz von Elementen, erkennbar an den sichtbaren Überschrifttexten beider Bereiche, lässt sich am einfachsten durch eine definierte Hintergrundfarbe ausschalten:

```
<div style="position:absolute; top:20px; left:60px; width:400px;
            height:100px; border:solid 2px blue;
            background-color:white">
   <h1>Erster Bereich</h1>
</div>
<div style="position:absolute; top:20px; left:60px; width:400px;
            height:100px; border:solid 2px red;
            background-color:white">
   <h1>Zweiter Bereich</h1>
</div>
```

Beide Bereiche haben nun mit `background-color:white` eine definierte Hintergrundfarbe erhalten. Die z-index-Eigenschaft wurde entfernt.

Der zweite Bereich überlappt den ersten aufgrund der natürlichen Schichtreihenfolge. Da die Elementinhalte der Bereiche nun nicht mehr transparent sind, überdeckt der zweite Bereich den ersten vollständig.



Abbildung 4.49: Identisch positionierte, aber nicht transparente Bereiche

Wer das Arbeiten mit Layern aus Grafikprogrammen kennt, weiß, dass besonders edle Effekte durch eine skalierbare Transparenz sich überdeckender Inhalte entstehen. CSS bietet ein entsprechendes Feature erst ab dem zur Drucklegung dieses Buches noch nicht offiziellen Sprachstandard 3.0 an. In einigen Browsern ist es bereits implementiert:

```
<div style="position:absolute; top:20px; left:60px; width:400px;
            height:100px; border:solid 2px red;
            background-color:white; opacity:0.5">
   <h1>Zweiter Bereich</h1>
</div>
```

Das Beispiel zeigt den Quelltext des zweiten unserer beiden identisch positionierten Bereiche. Beide Bereiche sind aufgrund der definierten weißen Hintergrundfarbe nicht transparent. Durch die CSS-3.0-Eigenschaft `opacity` lässt sich jedoch eine Halbtransparenz definieren. Wie stark unterhalb liegende Inhalte durchscheinen, lässt sich durch die Wertzuweisung skalieren. Empfohlen wird hierbei, mit Werten von 0.0 (voll transparent) bis 1.0 (voll opak, also deckend) zu arbeiten.

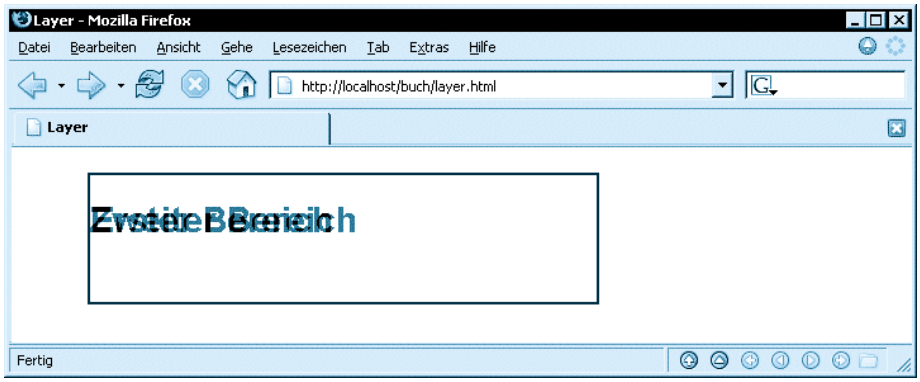


Abbildung 4.50: Halbtransparenter Bereich

Das Ergebnis der Zuweisung von `opacity:0.5` ist, dass der zweite Bereich den ersten nun zu 50% durchscheinen lässt. Auch die Rahmenfarben Rot und Blau mischen sich zu Lila.

Der MS Internet Explorer 6.0 interpretiert die opacity-Eigenschaft noch nicht, doch es gibt einen Workaround: die Microsoft-proprietäre CSS-Erweiterung der grafischen Filter. Auf unser Beispiel angewendet, könnten wir die CSS-Definition für den Internet Explorer folgendermaßen anpassen:

```
<div style="position:absolute; top:20px; left:60px; width:400px;
            height:100px; border:solid 2px red;
            background-color:white; opacity:0.5; filter:alpha(opacity=50)">
   <h1>Zweiter Bereich</h1>
</div>
```

Die CSS-Angabe `filter:alpha(opacity=50)` entspricht `opacity:0.5` und wird aber im Gegensatz zu Letzterer vom Internet Explorer interpretiert. Das Notieren beider Angaben bewirkt also, dass Browser, welche bereits mit `opacity` zurechtkommen, diese Eigenschaft interpretieren, während der Internet Explorer die filter-Eigenschaft interpretiert.

Das Arbeiten mit skalierbarer Teiltransparenz führt besonders bei Schrift auf halb durchscheinenden Grafiken zu edlen Effekten, wie sie beispielsweise aus zahlreichen Computerspielen mit transparenten Menüs bekannt sind. Behalten Sie jedoch bei solchen Effekten stets die Lesbarkeit des Textes im Auge. Wenn die Lesbarkeit dem Design geopfert wird, kann das Design noch so schick aussehen, es ist dennoch schlecht.

4.6.10 CSS-Eigenschaften für Listen und Tabellen

Listen und Tabellen haben einige spezifische Eigenschaften. So gehört zu den Variablen einer Aufzählungsliste das Bullet-Symbol oder die Art der Einrückung und bei nummerierten Listen die Art der Nummerierung. Bei Tabellen sind es spezielle Eigenschaften für Rahmen und Gitternetz oder die Art der Anzeige von leeren Zellen. Darüber hinaus haben aber auch anderweitig anwendbare Eigenschaften in Zusammenhang etwa mit Tabellen besondere Bedeutung, etwa Eigenschaften für Abstände oder horizontale und vertikale Ausrichtung.

Eigenschaften von Listen

Um das Aufzählungszeichen (Bullet-Symbol) einer Aufzählungsliste (ul-Element) festzulegen, stellt CSS die Eigenschaft `list-style-type` zur Verfügung. Bei nummerierten Listen (ol-Element) legt die gleiche Eigenschaft die Art der Nummerierung fest.

Beispiel für eine Aufzählungsliste:

```
<ul style="list-style-type:disc">
```

Beispiel für eine nummerierte Liste:

```
<ol style="list-style-type:upper-roman">
```

Für Aufzählungslisten sind die Wertzuweisungen `disc` (ausgefüllter Kreis ?), `circle` (nur umrandeter Kreis ?), `square` (ausgefülltes Quadrat ¦) und `none` (gar kein Aufzählungszeichen) erlaubt.

Die Nummerierungsart bei nummerierten Listen können Sie durch folgende Werte beeinflussen: `lower-roman` (i, ii, iii, iv usw.), `upper-roman` (I, II, III, IV usw.), `lower-alpha` (a, b, c, d usw.), `upper-alpha` (A, B, C, D usw.), `lower-latin` (a, b, c, d usw., jedoch auch mit Zeichen wie Umlauten, Accent-Buchstaben), `upper-latin` (A, B, C, D usw., inklusive Zeichen wie Umlauten, Accent-Buchstaben). Auch eine Alphanummerierung für andere Schriftkulturen kann definiert werden, durch `lower-greek` (Kleinbuchstaben des griechischen Alphabets), `georgian`, `armenian`, `hebrew`, `hiragana`, `katakana`, `hiragana-iroha`, `katakana-iroha`.

Grafische Browser stellen Aufzählungslisten und nummerierte Listen normalerweise so dar, dass das Aufzählungs- oder Nummerierungszeichen ausgerückt gegenüber mehrzeiligen Listenpunktinhalten dargestellt wird. Dieses Verhalten lässt sich durch Angabe von `list-style-position:inside` so ändern, dass die zweite und weitere Zeilen eines Listenpunkts unterhalb des Aufzählungspunkts oder der Nummerierung erscheinen.

Keine Lösung bietet CSS leider für das Default-Verhalten der Browser, Listen selbst eingerückt gegenüber dem normalen Fließtext darzustellen. Je nach Browser lässt sich das Verhalten zwar durch Angaben zu `margin-left` und `padding-left` beeinflussen, aber eine saubere Allgemeinlösung gibt es nicht. Das Gleiche gilt für den Abstand zwischen Aufzählungszeichen bzw. Nummerierung und Listenpunktinhalt.

Wenn Ihnen dagegen die magere Auswahl der bei `list-style-type` angebotenen Bullet-Symbole für Aufzählungslisten nicht zusagt, können Sie auch beliebige Grafiken als Bullet-Symbol verwenden. Dazu dient die Eigenschaft `list-style-image`. Ein Beispiel:

```
<ul style="list-style-image:url(blaues_dreieck.gif)">
```

Ähnlich wie bei `background-image` wird die gewünschte Grafik über das Schema `url(URI)` zugewiesen. URI kann eine beliebige Internetadresse oder eine lokal referenzierte Adresse sein. Im Beispiel befindet sich die Datei ***blaues_dreieck.gif*** im gleichen Verzeichnis. Die Grafik sollte eine passende Größe haben – die typische Icon-Größe von 16 × 16 Pixel ist eine praxisnahe Orientierungsgröße.

Rahmentyp bei Tabellen

Sichtbare Rahmen und Gitternetze von Tabellen in HTML werden von den meisten grafischen Browsern, wenn nichts anderes angegeben ist, so angezeigt, dass jede Zelle eine Umrahmung erhält. Dazu kommt der Außenrahmen. Webbrowser bilden diesen Sachverhalt so ab, dass Rahmen und Gitternetz als Doppellinien erscheinen oder in 3D-Form mit Licht- und Schattenseite. Beide Varianten wirken meist klobig. Durch die Angabe `border-collapse:collapse` im einleitenden `<table>`-Tag können Sie dafür sorgen, dass die Rahmen der Einzelzellen und die zwischen Zellen und Außenrahmen zusammenfallen.

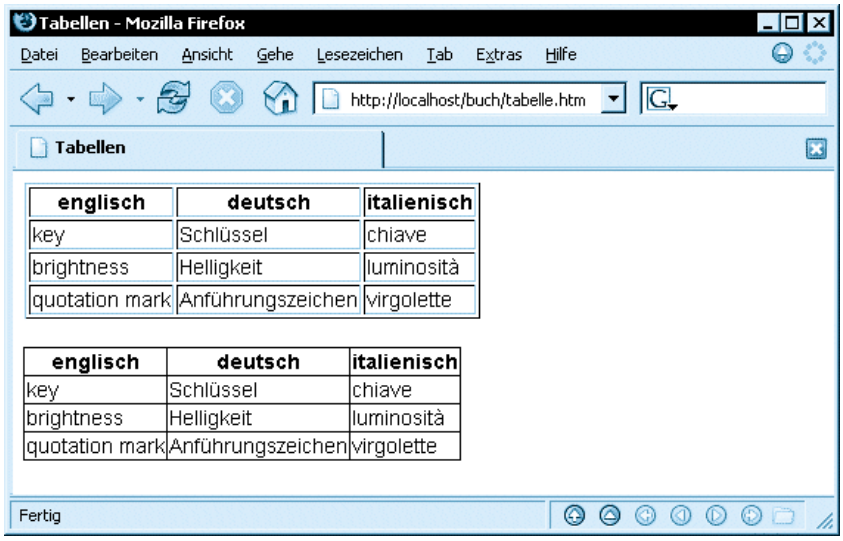


Abbildung 4.51: Tabelle ohne und mit zusammenfallenden Rahmen

Weitere Feinheiten zur Rahmengestaltung müssen Sie allerdings über die Eigenschaften `border`, `border-top`, `border-left`, `border-right` und `border-bottom` festlegen, und zwar sowohl für die Tabelle als auch für alle einzelnen Tabellenzellen. Da das Notieren von sich wiederholenden style-Attributen in etlichen Tabellenzellen jedoch nicht sehr zweckmäßig und in CSS viel eleganter über zentrale Formate lösbar ist, werden wir diese Möglichkeit im entsprechenden Zusammenhang behandeln (***Abschnitt 4.7***).

Breite von Tabellen und Spalten

Durch Angaben zur CSS-Eigenschaft `width` können Sie sowohl die Breite der gesamten Tabelle als auch einzelne Spaltenbreiten vorgeben. Dazu ein Beispiel:

```
<table border="1" style="width:500px;">
   <colgroup>
      <col style="width:80px">
      <col>
   </colgroup>
   <tr>
      <th>Platz</th>
      <th>Name</th>
   </tr><tr>
      <td>1.</td>
      <td>Michael Schumacher</td>
   </tr>
</table>
```

Eine Zuweisung von `width` im einleitenden `<table>`-Tag gibt eine gewünschte Breite für die gesamte Tabelle vor. Breiten für einzelne Spalten können Sie wahlweise in einem `<col>`-Tag oder im ersten `<th>`- oder `<td>`-Tag der Spalte vorgeben. Auch hierfür ist die

CSS-Eigenschaft `width` geeignet. Alternativ sind auch Definitionen mit `min-width` oder `max-width` möglich.

Diese Vorgaben sind jedoch nur Wunschvorgaben, die eingehalten werden, sofern es der Inhalt erlaubt. Ist der Inhalt so geartet, dass Spalten oder die gesamte Tabelle breiter werden als vorgegeben, geht die Kontrolle über die Breite verloren. CSS bietet jedoch die Möglichkeit das Tabellenlayout zu fixieren. Auch dazu ein Beispiel:

```
<table border="1" style="width:100%; table-layout:fixed;
                         border-collapse:collapse">
   <tr>
      <td style="width:15%; overflow:hidden; font-size:18px">
         1+2+3+4+5+6+7+8+9+10+11+12+13+14+15+16+17+18+19+20
      </td>
      <td style="width:30%; overflow:hidden; font-size:18px">
         1+2+3+4+5+6+7+8+9+10+11+12+13+14+15+16+17+18+19+20
      </td>
      <td style="width:55%; overflow:hidden; font-size:18px">
         1+2+3+4+5+6+7+8+9+10+11+12+13+14+15+16+17+18+19+20
      </td>
</tr>
</table>
```

In diesem Beispiel wird eine Gesamtbreite der Tabelle von 100% festgelegt. Steht die Tabelle direkt innerhalb von `<body> ... </body>`, bezieht sich diese Angabe auf die gesamte Dokumentbreite, also das Anzeigefenster abzüglich kleiner Default-Abstände vom Fensterrand. Auch für die einzelnen Spalten der Tabelle ist im Beispiel je eine Breite festgelegt: Die erste Spalte soll 15% Breite einnehmen, die zweite 30% und die dritte 55%.

Der Inhalt der Zellen ist jedoch problematisch. Da er keine Whitespace-Zeichen enthält, findet der Browser keine Stelle zum Umbrechen. Er würde die einzelnen Spalten und damit die gesamte Tabelle so weit ausdehnen, dass alle Inhalte angezeigt werden können. Durch die Angabe `table-layout:fixed` im einleitenden `<table>`-Tag wird dies jedoch verhindert. Die Angabe bewirkt, dass der Browser die vorgegebenen Breiten auf jeden Fall einhält. Was jedoch mit nicht passenden Spalteninhalten geschehen soll, müssen Sie ebenfalls angeben. Geben Sie nichts an, so hält der Browser die Breitenangaben ein, schiebt aber einen überbreiten Spalteninhalt einfach in die nächste Spalte, was zu hässlichen Überlappungen von Spalteninhalten kommt. In unserem Beispiel haben wir die Variante gewählt, durch Angabe von `overflow:hidden` in den einzelnen Tabellenzellen dafür zu sorgen, dass übergroße Inhalte abgeschnitten werden. Das Ergebnis sieht so aus:

Möglich wäre auch die Angabe `overflow:scroll`, die den Browser dazu veranlassen sollte, in Zellen mit übergroßem Inhalt Scrollleisten anzuzeigen. Doch leider funktioniert das gegenwärtig in kaum einem Browser.

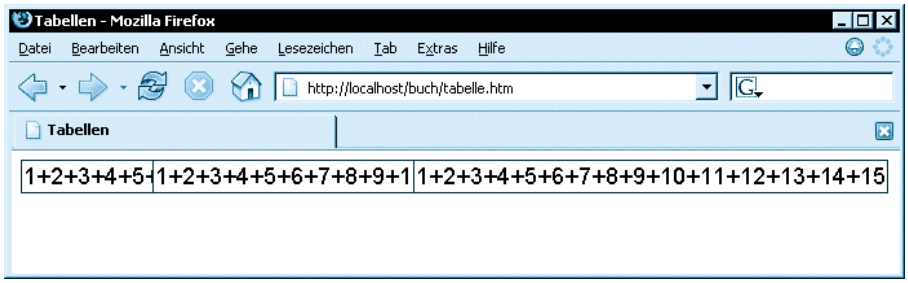


Abbildung 4.52: Fixes Tabellenlayout mit abgeschnittenen Spalteninhalten

Höhe und Ausrichtung von Tabelleninhalten

Inhalte von Tabellenzellen können unterschiedlich umfangreich sein und dadurch nicht immer die gesamte Breite der Spalte oder die gesamte Höhe der Zeile einnehmen. Die Spaltenbreite orientiert sich, sofern keine Breite vorgegeben wird, an dem breitesten Inhalt der Spalte und an den breitesten Inhalten anderer Spalten. Die Höhe orientiert sich an der Zelle mit dem vertikal umfangreichsten Inhalt innerhalb der Tabellenzeile.

Ebenso wie Breiten lassen sich natürlich auch Höhen explizit festlegen. Dazu steht die Eigenschaft `height` zur Verfügung. Auf das table-Element bezogen, gibt sie eine gewünschte Höhe für die gesamte Tabelle vor, und auf ein th- oder td-Element bezogen, gibt sie die gewünschte Höhe der betreffenden Tabellenzeile vor.

Anhand des Quelltextes einer vollständigen HTML-Datei zeigen wir nun einen in der Praxis häufig verlangten Effekt: Eine Tabelle soll den gesamten Raum des Anzeigefensters einnehmen und nur eine einzige Zelle enthalten. Deren Inhalt soll sowohl zentriert als auch vertikal mittig ausgerichtet werden, so dass der Inhalt genau in der Mitte des Anzeigefensters platziert wird:

Listing 4.7: Fensterfüllende Tabelle mit mittig zentriertem Inhalt

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html style="height:100%">
   <head>
      <meta http-equiv="content-type" content="text/html;
                        charset=ISO-8859-1">
      <title>Tabellen</title>
   </head>
   <body style="height:100%; margin:0; padding:0">
      <table style="width:100%; height:100%">
         <tr>
            <td style="text-align:center;
                       vertical-align:middle;">
               <div style="font-size:32px;
                           border-bottom:solid black 1px;
                           margin-left:20%; margin-right:20%;
                           padding-bottom:5px">
                  Einsam in der Mitte
```

```
                </div>
                <div style="font-size:20px; padding-top:5px">
                    Ein Film von Emanuel Piesepampel
                </div>
            </td>
        </tr>
    </table>
</body>
</html>
```

Das einleitende `<table>`-Tag bekommt die CSS-Definition `height:100%` zugewiesen. Das allein reicht jedoch nicht aus, da der Tabelleninhalt recht klein ist und das Elternelement, also das body-Element, nicht ausfüllt. Da sich aber die 100% auf die Höhe des Elternelements beziehen, bleibt der gewünschte Effekt aus. Deshalb wird dem body-Element ebenfalls die Definition `height:100%` zugewiesen. Doch auch das genügt noch nicht, da sich das body-Element, was die Höhe betrifft, nicht an den Ausdehnungen des Anzeigefensters orientiert. Deshalb wird auch noch dessen Elternelement, dem html-Element, explizit die Angabe `height:100%` zugewiesen. Erst dadurch wird die Tabelle tatsächlich auf 100% Höhe ausgedehnt.

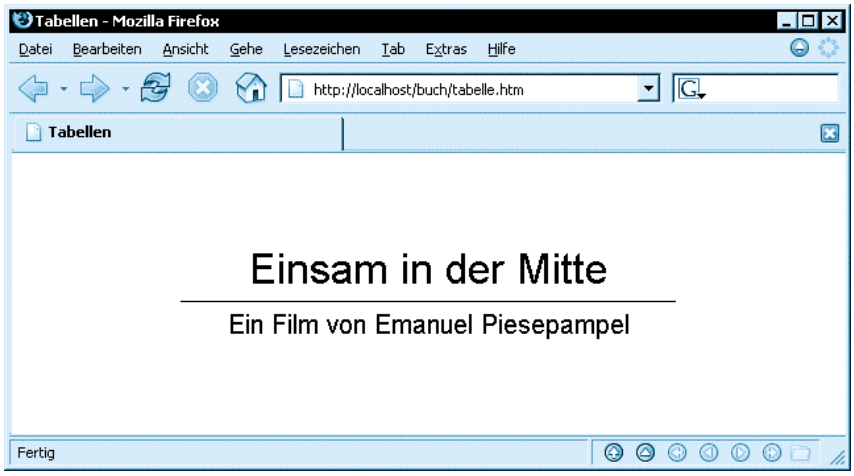


Abbildung 4.53: Fensterfüllende Tabelle mit mittig zentriertem Inhalt

Das nächste Problem entsteht jedoch dadurch, dass die Tabelle nun zwar auf 100% Höhe des Anzeigefensters ausgedehnt wird, der Browser den Inhalt des body-Elements jedoch per Default mit etwas Abstand zum Fensterrand versieht. Die Tabelle reicht dadurch in der Höhe ein klein wenig weiter als das Anzeigefenster, so dass ein vertikales Scrollen erforderlich wird, um den gesamten Tabelleninhalt zu sehen. Dieses Problem umgehen wir im Beispiel, indem wir dem body-Element `margin:0` und `padding:0` zuweisen. Diese Angaben entfernen alle Default-Abstände zum Fensterrand.

Da die Tabelle nur eine einzige Zelle besitzt, wird diese auf Grund der Breiten- und Höhenangaben sowohl in der Breite als auch in der Höhe über das gesamte Anzeigefens-

ter ausgedehnt. Um ihren Inhalt genau in die Mitte des Anzeigefensters zu bringen, richten wir ihn sowohl horizontal zentriert als auch vertikal mittig aus.

Die horizontale Ausrichtung von Tabellenzelleninhalten ist genau so wie das Ausrichten von anderen Block-Elementen mithilfe von `text-align` möglich. Tabelleninhalte können explizit links (`left`), rechts (`right`), zentriert (`center`) oder als Blocksatz (`justify`) ausgerichtet werden.

Für die vertikale Ausrichtung stellt CSS die Eigenschaft `vertical-align` zur Verfügung. Gerade bei Tabellenzellen kommt es häufig vor, dass diese vertikal obenbündig (`top`), mittig (`middle`) oder untenbündig (`bottom`) ausgerichtet werden sollen. Die vertical-align-Eigenschaft erlaubt daneben auch noch weitere Angaben, nämlich `baseline` (Ausrichtung an der Schriftgrundlinie des Elternelements), `sub` (Ausrichtung an der Position für tiefgestellten Text), `super` (Ausrichtung an der Position für hochgestellten Text), `text-top` (Ausrichtung an der Schriftoberkante des Elternelements) und `text-bottom` (Ausrichtung an der Schriftunterkante des Elternelements). Für Tabelleninhalte sind jedoch `top`, `middle` und `bottom` in der Regel ausreichend.

Durch die Angaben `text-align:center` und `vertical-align:middle` wird der Inhalt unserer einzigen, das Dokumentfenster voll ausfüllenden Tabellenzelle also genau in der Fenstermitte ausgerichtet. Der Zelleninhalt besteht aus zwei div-Bereichen, die durch entsprechende Formatierungen das gewünschte Schriftbild erreichen.

Abstände innerhalb von Zellen und Abstände zwischen Zellen

In vielen Fällen ist es optisch schöner, einen gewissen Abstand zwischen den Gitternetzlinien einer Tabelle und den Zellinhalten zu haben. Dazu wenden Sie einfach die für Innenabstand üblichen CSS-Eigenschaften `padding` (alle vier Seiten), `padding-left` (links), `padding-top`, `padding-right` (rechts) und `padding-bottom` (unten) auf th- bzw. td-Elemente an.

Ein trauriges Kapitel ist leider der Abstand zwischen Zellen. Dafür steht zwar die CSS-Eigenschaft `border-spacing` zur Verfügung, die genau das mit einer einzigen Angabe im `<table>`-Tag erledigt. Doch leider interpretiert der MS Internet Explorer diese Eigenschaft nicht. Da bleibt nichts anderes, als auf das ältere, aber nicht zum Strict-Standard gehörende HTML-Attribut `cellspacing` auszuweichen. In diesem Fall müssen Sie allerdings, wenn Sie konsequent sein wollen, den HTML-Dokumenttyp auf die Transitional-Variante ändern. Nachfolgend das Beispiel einer Tabelle, das auch im Internet Explorer 6.0 funktioniert:

```
<table cellspacing="10" style="background-color:teal; border-spacing:10px">
   <thead style="background-color:white;"><tr>
         <th>Äpfel</th><th>Birnen</th>
      </tr></thead>
   <tfoot style="background-color:aqua"><tr>
         <td>beliebter</td><td>weniger beliebt</td></tr>
   </tfoot>
   <tbody style="background-color:aqua">
      <tr>
```

```
            <td>12 Einheiten Vitamin C</td>
            <td>5 Einheiten Vitamin C</td>
         </tr><tr>
            <td>regt die Verdauung an</td>
            <td>wirkt entwässernd</td>
         </tr>
      </tbody>
   </table>
```

Im Beispiel wird ein 10 Pixel großer Abstand zwischen den Tabellenzellen definiert. CSS-gerecht wird dazu im style-Attribut des `<table>`-Tags die Angabe `border-spacing:10px` notiert. Um rückwärtskompatibel zum MS Internet Explorer zu bleiben, wird im einleitenden `<table>`-Tag außerdem das Attribut `cellspacing="10"` notiert, welches den gleichen Effekt hat. Die Angabe zum Zellabstand funktioniert logischerweise nicht, wenn gleichzeitig `border-collapse:collapse` angegeben wird, da dies ja die Zellrahmen zusammenfallen lässt, was nur bei einem Zellabstand von 0 möglich ist.

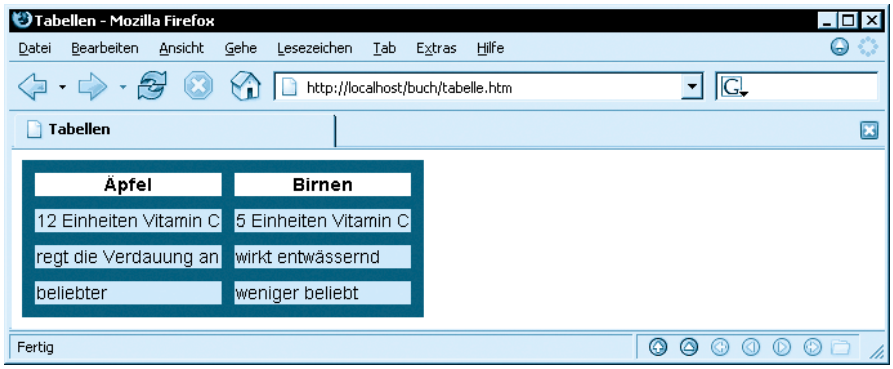


Abbildung 4.54: Farbige Tabelle mit Zellabstand

Wenn die Tabelle einen sichtbaren Rahmen hat, ist der definierte Zellenabstand im Gitternetz gut erkennbar. Im obigen Beispiel entschieden wir uns jedoch für eine andere Visualisierung. Die Tabelle hat keine sichtbaren Rahmen und Gitternetzlinien, doch dafür wurden Hintergrundfarben für die gesamte Tabelle und für einzelne Zellen definiert – mithilfe der bereits bekannten CSS-Eigenschaft `background-color`. Auf das table-Element angewendet, wird eine Hintergrundfarbe für die Tabelle bestimmt. Diese wird im Beispiel aber nur gut sichtbar, weil ein großzügiger Zellabstand definiert wurde. Für einzelne Tabellenzellen, welche die gleiche Hintergrundfarbe erhalten sollen, genügt es, `background-color` auf das zugehörige tr-Element oder noch allgemeiner auf das thead-, tfoot- oder tbody-Element anzuwenden.

4.7 Wiederverwendbare Formate mit CSS

Wenn Sie für Tabellen mit Hunderten oder Tausenden von Zellen einen Zellinnenabstand zwischen Gitternetzlinie und Zellinhalt definieren wollen, müssten Sie, wenn CSS in HTML nur über das style-Attribut eingebunden werden könnte, die padding-Angaben in jeder einzelnen Tabellenzelle wiederholen. Glücklicherweise ist das nicht nötig. Denn es besteht die Möglichkeit, CSS-Angaben zentral zu definieren und auf selektierbare Elemente anzuwenden.

In ***Abschnitt 4.6*** haben wir bewusst nur mit dem style-Attribut gearbeitet, weil dies die unmittelbarste und zunächst intuitivste Methode ist, HTML-Elemente mit CSS zu gestalten. Gedacht ist diese Methode jedoch eigentlich nur für Fälle, in denen bestimmte Elemente individuell formatiert werden sollen. In allen Fällen dagegen, wo die gleichen CSS-Definitionen auf mehr als ein Element anwendbar sind, sollten Sie auf jeden Fall auf zentrale Formatdefinitionen zurückgreifen, wie sie in diesem Abschnitt beschrieben werden. Zentral definierte, wiederverwendbare Formate erhöhen die Einheitlichkeit der Formate und tragen damit entscheidend zu einem professionellen Look bei.

4.7.1 Dokumentglobale und dokumentübergreifende Formate

CSS bietet zwei Varianten an, um Formate zentral zu definieren: im Dateikopf einzelner HTML-Seiten oder in einer separaten Datei, die dann in beliebig vielen HTML-Seiten eingebunden werden kann. Die erste Variante ist sinnvoller, wenn Sie die Formate nur für ein einzelnes Dokument definieren. Letztere Variante ist dagegen vorzuziehen, wenn Sie einheitliche Formate für eine komplette Site, bestehend aus mehreren bis etlichen HTML-Dateien, oder für dynamisch generierte HTML-Seiten (etwa mit PHP) definieren wollen. Die Syntax der zentralen Formatdefinitionen ist in beiden Fällen die gleiche.

Dokumentglobale CSS-Definitionen im style-Bereich des Dateikopfs

Im HTML-Kopfbereich, also zwischen `<head>` und `</head>`, können Sie einen Bereich für CSS-Formatdefinitionen notieren. Dazu ein erstes Beispiel:

```
<head>
   <title>Titel der Datei</title>
   <meta http-equiv="content-type"
         content="text/html; charset=ISO-8859-1">
   <style type="text/css">
      h1 { font-size:18px;
           font-family:Tahoma,Arial,sans-serif;
           color:blue;
           border-bottom:solid blue 18px;
           padding-bottom:10px;
      }
      p,div,li,th,td  {
           font-family:Tahoma,Arial,sans-serif;
      }
   </style>
</head>
```

Der Definitionsbereich wird in `<style>...</style>` eingeschlossen. Das einleitende `<style>`-Tag erhält das Attribut `type="text/css"`. Die Wertzuweisung ist der Mime-Type für CSS. Der Browser weiß dadurch, dass er den Inhalt des Elements mit seinem CSS-Parser auslesen muss.

Einige Autoren schließen den Inhalt des style-Bereichs zusätzlich in einen HTML-Kommentar (`<!-- ... -->`) ein, um ältere Browser davor zu bewahren, den Inhalt als Text anzuzeigen. Das Verhalten eines Browsers, der ein HTML-Element nicht kennt, sollte nämlich darin bestehen, das Markup des unbekannten Elements einfach zu ignorieren und darin enthaltenen Textinhalt einfach anzuzeigen. Doch solche Browser gibt es praktisch nicht mehr. Auch textbasierte Browser wie Lynx haben längst ausgereifte HTML-Parser, die auch Elemente wie `style` kennen und wissen, dass dessen Inhalt nicht angezeigt werden soll.

Auf die Definitionen innerhalb des style-Bereichs gehen wir später ein.

Dokumentübergreifende CSS-Definitionen in separaten CSS-Dateien

Der gleiche Inhalt, der innerhalb eines style-Elements im HTML-Kopfbereich stehen kann, kann auch – ohne jede umgebende HTML-Auszeichnung – in einer separaten Textdatei stehen. Die Datei könnte also beispielsweise folgenden Inhalt haben:

```
h1 { font-size:18px;
     font-family:Tahoma,Arial,sans-serif;
     color:blue;
     border-bottom:solid blue 18px;
     padding-bottom:10px;
}
p,li  {
   font-family:Tahoma,Arial,sans-serif;
}
```

Als Dateinamenerweiterung sollte die Datei *.css* erhalten. Eine solche Datei können Sie in HTML referenzieren, um so die darin definierten CSS-Formate für das HTML-Dokument zu übernehmen.

Es gibt zwei Möglichkeiten, die CSS-Datei zu referenzieren. Die am häufigsten verwendete ist die über das link-Element in HTML. Dazu ein Beispiel:

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

Das link-Element wird, wie wir bereits wissen, im Kopfbereich notiert, also zwischen `<head>` und `</head>`. Dem rel-Attribut wird in diesem Fall der Wert `stylesheet` zugewiesen. Als Mime-Type der verlinkten Ressource wird wie für CSS üblich `text/css` angegeben. Im href-Attribut schließlich wird die CSS-Datei referenziert. Der Wert kann ein beliebiger URI sein, also eine beliebige Internetadresse oder eine lokal referenzierte Quelle. Im obigen Beispiel liegt die Datei *formate.css* einfach im gleichen Verzeichnis wie die HTML-Datei, in der das link-Element notiert ist.

Daneben existiert die Möglichkeit, eine externe CSS-Datei innerhalb eines style-Bereichs zu referenzieren, und zwar mit einer CSS-internen Syntax. Diese Möglichkeit ist besonders dann zu empfehlen, wenn Sie sowohl externe CSS-Formate als auch dokumentspezifische Formate verwenden möchten. Ein Beispiel:

```
<style type="text/css">
   @import url(formate.css) all;
   p,li  {
      font-size:12px;
   }
</style>
```

In style-Bereichen können so genannte *rules* (zu Deutsch: Regeln) definiert werden, an dem führenden @-Zeichen zu erkennen. Eine dieser Regeln ist die @import-Regel. Über diese Regel lassen sich externe Stylesheet-Dateien in den style-Bereich einbinden. Die Syntax ist dabei ähnlich wie etwa bei der CSS-Eigenschaft `background-image`. Das Schema lautet auch hier `url(URI)`, wobei URI eine beliebige Internetadresse sein kann oder auch eine lokal referenzierte Quelle. Im Beispiel liegt die Datei *formate.css* im gleichen Verzeichnis wie die referenzierende HTML-Datei.

Hinter der Adressierung der Stylesheet-Datei folgt noch eine Angabe zum Medientyp. Wenn Sie nichts Genaueres spezifizieren möchten, geben Sie dort am besten `all` an – damit sind alle Medientypen gemeint. Auf medienspezifisches CSS gehen wir in Kapitel 6 noch näher ein.

Verhalten bei widersprüchlichen oder ergänzenden Definitionen

Wir haben nun drei »Orte« kennen gelernt, an denen CSS-Formate definiert werden können:

- Formatdefinitionen in einer externen CSS-Datei: geeignet für Formate, die für mehrere bis alle Seiten eines Projekts, einer Site gültig sein sollen.
- Formatdefinitionen im style-Element innerhalb der HTML-Kopfdaten: geeignet für Formate, die innerhalb des Dokuments mehrfach benötigt werden und nicht in einer separaten CSS-Datei definiert sind.
- Formatdefinitionen im style-Attribut eines HTML-Elements: geeignet für individuelle, von den Standarddefinitionen abweichende Formatierung und Gestaltung.

Es ist problemlos möglich, alle drei Orte von Formatdefinitionen zu mischen. Wichtig ist dabei aber, die folgenden Regeln zu kennen:

- Formatdefinitionen im style-Bereich in den HTML-Kopfdaten haben Vorrang vor Formatdefinitionen aus referenzierten externen CSS-Dateien.
- Formatdefinitionen in style-Attributen einzelner HTML-Elemente haben Vorrang vor Formatdefinitionen aus dem style-Bereich.

Je »näher« sich eine Formatdefinition am konkreten Zielelement befindet, desto mehr Vorrang hat sie also. Ein Beispiel soll dies demonstrieren. Eine separate CSS-Datei namens *formate.css* hat folgenden Inhalt:

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

Eine HTML-Datei hat unter anderem folgende Kopfdaten:

```
<link rel="stylesheet" type="text/css" href="formate.css">
<style type="text/css">
   h1 {
      color:green;
   }
</style>
```

Und im Dateikörper der HTML-Datei ist schließlich Folgendes notiert:

```
<h1 style="color:blue">Welche Farbe habe ich?</h1>
```

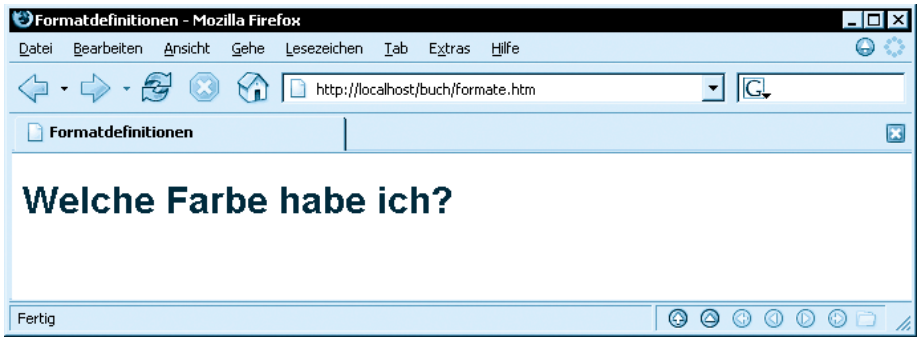


Abbildung 4.55: Das Ergebnis konkurrierender CSS-Formatdefinitionen

Die Antwort lautet Blau (`blue`). Würde das style-Attribut im h1-Element fehlen, so wäre die Antwort Grün (`green`), wegen der Definition im style-Bereich. Würde diese Definition ebenfalls fehlen, so wäre die Antwort Rot (`red`), wegen der Definition in der externen CSS-Datei.

Formateigenschaften aus entfernteren Definitionsorten können also durch solche an näheren Definitionsorten überschrieben werden. Sie können sich aber auch – und das dürfte wohl der häufigere Fall sein, einfach nur vererben und ergänzen. So lautet beispielsweise eine Anforderung, dass die Hintergrundfarbe einer Navigationsleiste verschieden sein kann, weil die Site in verschiedene Kategorien gegliedert ist, und eine Kategorie an der Farbe erkennbar sein soll. In einem solchen Fall bietet es sich an, die Angaben zur Positionierung der Navigationsleiste in einer separaten CSS-Datei zu notieren, da die Position immer gleich bleibt. Die Hintergrundfarbe kann dann innerhalb der einzelnen Dokumente definiert werden. Positionierung und Hintergrundfarbe ergänzen sich, oder CSS-gerechter

ausgedrückt: die Angaben zur Positionierung werden geerbt und die zur Hintergrundfarbe werden hinzugefügt.

4.7.2 Selektoren

Die Formatdefinitionen an sich haben an allen drei Definitionsorten das gleiche Syntaxschema: `CSS-Eigenschaft:Wertzuweisung`. Mehrere Formatdefinitionen werden durch Semikola voneinander getrennt.

Bei Formatdefinitionen innerhalb eines HTML-Elements, genauer, innerhalb von dessen style-Attribut, ist die Zuordnung zum Element eindeutig. Bei zentralen Definitionen im style-Bereich oder in einer externen CSS-Datei dagegen müssen Sie eine solche Zuordnung erst herstellen. Für jedes definierte Set von Formatdefinitionen müssen Sie deshalb einen ***Selektor*** notieren. Ein Selektor grenzt ein, welche Elemente von den zugehörigen Formatdefinitionen betroffen sein sollen. Die Gesamtsyntax ist dabei Folgende:

```
Selektor { CSS-Eigenschaft:Wert; CSS-Eigenschaft:Wert; ... }
```

Das Set an Formatdefinitionen wird in geschweifte Klammern eingeschlossen. Vor der öffnenden geschweiften Klammer wird der Selektor notiert. Leerraumzeichen sind überall erlaubt. Bei umfangreicheren CSS-Formatdefinitionen ist eine Notation wie die folgende am übersichtlichsten:

```
h1 {
   font-family:inherit;
   font-size:20px;
   font-weight:normal;
   color:rgb(195,35,35);
   letter-spacing:1px;
   margin-top:0px;
   margin-bottom:30px;
   padding-bottom:10px;
   border-bottom:rgb(145,35,35) 4px double
}
```

In diesem Beispiel ist `h1` der Selektor. Das bedeutet, dass die Formatdefinitionen auf alle h1-Elemente im HTML-Quelltext, also Überschriften 1. Ordnung, angewendet werden.

Bei der letzten Formatdefinition vor einer schließenden geschweiften Klammer kann das Semikolon entfallen.

Selektoren sind ein mächtiges Werkzeug, wie wir in den nachfolgenden Abschnitten sehen werden. Nicht nur »alle HTML-Elemente mit diesem Namen« können selektiert werden, sondern auch nur bestimmte Elemente, abhängig von verschiedenen Bedingungen.

4.7.3 Formatdefinitionen für Elemente

Den einfachsten Fall haben wir bereits kennen gelernt. Ein HTML-Element wie `h1`, `p`, `strong`, `code`, `div` oder `body` ist dabei der Selektor. Der nächste Fall besteht darin, HTML-Elemente mit gleichartigen Formatdefinitionen zu gruppieren. Ein Beispiel:

```
body, h1, h2, h3, h4, div, p, blockquote, th, td, li {
   font-family:Arial,sans-serif;
   color:black;
   background-color:white;
}
h1, h2, h3, h4 {
   color:rgb(195,35,35);
}
h1, h2 {
   letter-spacing:1px;
}
h1 {
   font-size:20px;
   font-weight:normal;
   margin-top:0px;
   margin-bottom:30px;
   padding-bottom:10px;
   border-bottom:rgb(195,35,35) 4px double
}
```

Bei einem Set von Formatdefinitionen, das für mehrere Elemente gelten soll, besteht der Selektor darin, die gewünschten Elemente durch Kommata zu trennen.

In der Praxis ist es sinnvoll, logisch so vorzugehen wie im obigen Code-Beispiel gezeigt: Zunächst werden wenige allgemeine Formatdefinitionen auf zahlreiche Elemente angewendet und am Ende werden zahlreiche spezielle Formatdefinitionen auf einzelne Elemente angewendet. Das Vererbungsprinzip von CSS sorgt dafür, dass die Formatdefinitionen sich ergänzen. Wenn Widersprüche auftauchen, wie im obigen Beispiel die Definitionen für `color` im ersten und im zweiten Set, dann überschreibt die später interpretierte Definition die frühere. Solche Widersprüche sind oft gar nicht als Widersprüche konzipiert. Es wird einfach zuerst ein Allgemeinfall definiert (im Beispiel: schwarze Textfarbe) und dann wird ein abweichender Fall für eine kleinere Selektion von Elementen definiert (im Beispiel: rote Textfarbe für Überschriften 1. bis 4. Ordnung).

Berücksichtigung der Elementumgebung

Ein Set von Formatdefinitionen ist also auf mehrere Elemente anwendbar, was die Anzahl der Anwendungsfälle erhöht. Oft ist aber auch der umgekehrte Fall erwünscht, nämlich eine kontrollierte Reduzierung der Anwendungsfälle. Eine Möglichkeit dazu sind Selektoren, die HTML-Elementverschachtelungen berücksichtigen. Dazu ein relevanter Beispielausschnitt aus einem HTML-Dokument:

```
<style type="text/css">
   h1 {
      color:rgb(195,35,35);
      font-family:Arial,sans-serif;
   }
   h1 code {
      color:rgb(35,35,195);
      font-family:Arial,sans-serif;
   }
</style>
<title>Formatdefinitionen</title>
</head>
<body>
   <h1>Die CSS-Eigenschaft <code>font-family</code></h1>
   <p>Die Eigenschaft <code>font-family</code> dient zur
    Bestimmung der Schriftart.</p>
</body>
```

In den zentralen Formatdefinitionen wird zunächst für h1-Elemente eine Schriftfarbe und eine Schriftart definiert. Anschließend wird für code-Elemente, die innerhalb von h1-Überschriften vorkommen, ebenfalls eine Schriftfarbe und eine Schriftart festgelegt.



Abbildung 4.56: Formatdefinition für Elementverschachtelung

Im HTML-Quelltext des Beispiels kommt das code-Element zweimal vor: in der h1-Überschrift und im nachfolgenden Textabsatz. Durch den Selektor `h1 code` wird erreicht, dass Textinhalte des code-Elements blau und in Arial oder einer anderen serifenlosen Schrift dargestellt werden, wenn (und nur wenn) das code-Element innerhalb einer h1-Überschrift vorkommt. Kommt das code-Element dagegen innerhalb eines anderen Elements vor, ist seine Darstellung abhängig von den Eigenschaften des anderen Elternelements und von den Browser-Voreinstellungen zur Darstellung des code-Elements an sich. Üblicherweise zeigen z. B. grafische Browser das code-Element in einer dicktengleichen Schrift an, wie auch in der Abbildung beim Wort »font-family« innerhalb des Textabsatzes zu sehen. Vererbt werden dem code-Element sowohl in der Überschrift als auch im Textabsatz Eigenschaften wie etwa die Schriftgröße. Durch den Selektor `h1 code` werden jedoch Schriftart und Farbe bei Vorkommen in einer h1-Überschrift explizit definiert.

Selektoren für einfache Elementverschachtelungen haben also folgendes Schema:

```
Element_1 Element_2 { Formatdefinitionen }
```

Dabei ist `Element_1` das Elternelement und `Element_2` das Element, für das die Formatdefinitionen gelten, wenn es innerhalb von `Element_1` vorkommt. `Element_1` und `Element_2` werden einfach durch ein oder mehrere Whitespace-Zeichen (z. B. ein Leerzeichen) voneinander getrennt.

Das Aufzählen mehrerer Elemente, getrennt durch Kommata, ist mit Verschachtelungsangaben beliebig kombinierbar. Eine Notation wie `h1,h2 code` bedeutet: Folgende Formatdefinitionen gelten für code-Elemente, die innerhalb von einer h1- oder einer h2-Überschrift vorkommen. Eine Notation wie `h1 code,kbd` bedeutet: Folgende Formatdefinitionen gelten für code- oder kbd-Elemente, die innerhalb einer h1-Überschrift vorkommen. Und eine Notation wie `h1,h2 code,kbd` bedeutet: Folgende Formatdefinitionen gelten für code- oder kbd-Elemente, wenn sie innerhalb von h1- oder h2-Überschriften vorkommen.

Neben der einfachen Verschachtelung besteht seit CSS 2.0 die Möglichkeit, auch komplexere Verschachtelungsregeln zu definieren. Die nachfolgende Tabelle listet alle Möglichkeiten der Verschachtelung auf.

Zeichen	Beispiel	Bedeutung
`[leer]`	`div em`	Formatdefinitionen gelten für em-Elemente, die innerhalb von div-Elementen vorkommen, egal in welcher Verschachtelungstiefe, also z. B. sowohl bei `<div><em></em></div>` als auch bei `<div><ul><li><em></em></li></ul></div>`.
`>`	`div > em`	Formatdefinitionen gelten für em-Elemente, die unmittelbar innerhalb von div-Elementen vorkommen, also nur bei Verschachtelungen wie `<div><em></em></div>`.
`*`	`div * em`	Formatdefinitionen gelten für em-Elemente, die innerhalb von div-Elementen vorkommen, aber nur, wenn mindestens noch eine Verschachtelungsebene dazwischen liegt, also z. B. bei `<div><p><em></em></p></div>`.
`+`	`div + p`	Formatdefinitionen gelten für p-Elemente, die unmittelbar nach einem div-Element vorkommen, also wenn die Struktur `<div></div><p></p>` gegeben ist.

Tabelle 4.13: Syntax bei Verschachtelungsangaben im Selektor

Die beiden Varianten mit den Zeichen > und * sind also dazu gedacht, um die Verschachtelungsregel genauer zu definieren. Die letztgenannte Möglichkeit mit dem Zeichen + ist genau genommen keine Verschachtelungsregel mehr. Diese Syntax ist jedoch sehr praktisch für alle Fälle, in denen aufeinander folgende Elemente sich in bestimmter Weise verhalten sollen. So lässt sich beispielsweise durch `h1 + p { margin-top:30px }` erreichen, dass ein Textabsatz einen Abstand von 30 Pixel oben erhält, aber nur, wenn er unmittelbar hinter einer Überschrift 1. Ordnung notiert ist.

4.7.4 Formatdefinitionen für Klassen und Einzelelemente

Die im Abschnitt zuvor behandelten Möglichkeiten, HTML-Elemente mit Selektoren zentral zu formatieren, gehen von Elementen aus, die keine weiteren identifizierenden Angaben enthalten als ihren HTML-Elementnamen. Es gibt jedoch zwei HTML-Attribute, die in allen Elementen erlaubt sind und es ermöglichen, ein Element entweder zu klassifizieren oder eindeutig zu identifizieren:

- Das class-Attribut erlaubt es, Elemente einem Klassennamen zuzuordnen.
- Das id-Attribut erlaubt es, einem Element einen dokumentweit eindeutigen Namen zu geben.

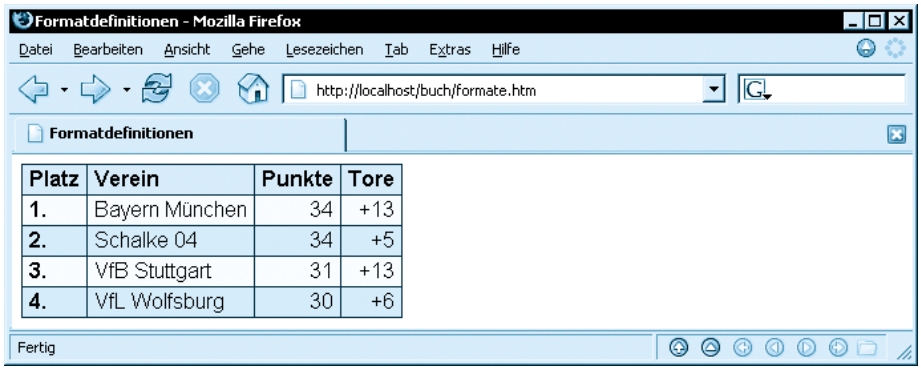
Anhand eines Komplettbeispiels, also des Quelltextes einer vollständigen HTML-Datei sollen diese Attribute und ihr Zusammenspiel mit CSS-Selektoren erläutert werden:

Listing 4.8: HTML mit class- und id-Attributen und entsprechende CSS-Selektoren

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<style type="text/css">
   body, table {
      background-color:white;
   }
   #Bundesliga {
      border-collapse:collapse;
   }
   #Bundesliga th, td {
      font-size:16px;
      padding-left:5px;
      padding-right:5px;
      padding-top:1px;
      padding-bottom:1px;
   }
   th.Bundesliga {
      text-align:left;
      background-color:rgb(221,221,221);
      color:black;
   }
   .gelb {
      background-color:rgb(255,255,153);
      color:black;
   }
   .orange {
      background-color:rgb(255,204,153);
      color:black;
   }
   .rechts {
```

```
      text-align:right;
   }
   .fett {
      font-weight:bold;
   }
</style>
<title>Formatdefinitionen</title>
</head>
<body>
<table border="1" id="Bundesliga">
<thead>
<tr>
   <th class="Bundesliga">Platz</th>
   <th class="Bundesliga">Verein</th>
   <th class="Bundesliga">Punkte</th>
   <th class="Bundesliga">Tore</th>
</tr>
</thead>
<tbody>
<tr class="gelb">
   <td class="fett">1.</td>
   <td>Bayern München</td>
   <td class="rechts">34</td>
   <td class="rechts">+13</td>
</tr>
<tr class="orange">
   <td class="fett">2.</td>
   <td>Schalke 04</td>
   <td class="rechts">34</td>
   <td class="rechts">+5</td>
</tr>
<tr class="gelb">
   <td class="fett">3.</td>
   <td>VfB Stuttgart</td>
   <td class="rechts">31</td>
   <td class="rechts">+13</td>
</tr>
<tr class="orange">
   <td class="fett">4.</td>
   <td>VfL Wolfsburg</td>
   <td class="rechts">30</td>
   <td class="rechts">+6</td>
</tr>
</tbody>
</table>
</body>
</html>
```

Das Beispiel enthält eine mit CSS formatierte Tabelle, die am Ende folgendermaßen aussieht:



Platz	Verein	Punkte	Tore
1.	Bayern München	34	+13
2.	Schalke 04	34	+5
3.	VfB Stuttgart	31	+13
4.	VfL Wolfsburg	30	+6

Abbildung 4.57: Mit Selektoren für class- und id-Attribute formatierte HTML-Tabelle

In einleitenden HTML-Tags sind die Attribute `class=` zum Zuweisen eines Klassennamens und `id=` zum Zuweisen eines dokumentweit eindeutigen Namens zulässig. Für die korrekte Anwendung müssen Sie als HTML-Autor selbst sorgen. Der einem id-Attribut zugewiesene Wert darf im gesamten Dokument wirklich nur einmal vorkommen. Wertzuweisungen an das class-Attribut sind dagegen eher sinnvoll, wenn sie mehrfach im Dokument vorkommen. Dabei ist es jedoch wichtig, dass der Klassenname in allen Fällen exakt gleich geschrieben ist (Groß-/Kleinschreibung berücksichtigen).

Halten Sie sich bei id- und Klassennamen an folgende Regeln:

- Das erste Zeichen muss ein Buchstabe A–Z bzw. a–z sein.
- Weitere Zeichen dürfen Buchstaben A–Z, a–z, Ziffern 0–9 sowie die Zeichen für Unterstrich (_), Bindestrich/Minuszeichen (-), Doppelpunkt (:) und Punkt (.) sein. Von der Verwendung von Bindestrich/Minuszeichen, Doppelpunkt und Punkt ist in der Praxis jedoch abzuraten, da die Namen dann bei Verwendung in Scriptsprachen wie JavaScript zu Fehlern führen können.
- Leerzeichen in Namen sind nicht erlaubt.

Klassen- und ID-Namen haben unterschiedliche Namensräume. Im obigen Beispiel gibt es beispielsweise ein `<table>`-Tag mit ID-Namen `Bundesliga`, gleichzeitig aber auch Kopfzellen `<th>` mit dem gleichen Klassennamen. Das führt zu keinen Konflikten.

Bei CSS-Selektoren werden Klassennamen durch einen Punkt (.) referenziert und id-Namen durch ein Gatterzeichen (#). Dabei sind folgende Notationen erlaubt:

- Mit `.Klassenname` referenzieren Sie alle HTML-Elemente, in deren einleitendem Tag `class="Klassenname"` notiert ist.
- Mit `Elementname.Klassenname` referenzieren Sie alle HTML-Elemente des Typs Elementname, in denen `class="Klassenname"` notiert ist, also z.B. mit `th.Bundesliga` alle Elemente `<th class="Bundesliga">`, nicht aber Elemente wie `<td class="Bundesliga">`.
- Mit `#IDName` referenzieren Sie ein bestimmtes Element mit eben diesem id-Namen.

- Mit `Elementname#IDName` referenzieren Sie ein bestimmtes Element mit eben diesem id-Namen, jedoch nur dann, wenn es ein Element des Typs `Elementname` ist, also z. B. mit `h1#Kopf` das Element `<h1 id="Kopf">`, nicht aber ein Element `<div id="Kopf">`. Zwar dürfen in einem Dokument ohnehin nicht beide Elemente mit dem gleichen id-Namen vorkommen, doch bei Formatdefinitionen in separaten CSS-Dateien, die in zahlreichen unterschiedlichen Dokumenten referenziert werden, kann es durchaus vorkommen, dass verschiedene Elemente in verschiedenen Dokumenten den gleichen id-Namen haben.

In unserem Beispiel wird für ein table-Element mit dem id-Namen `Bundesliga` bestimmt, dass Rahmen von Gitternetzlinien und Außenrahmen zusammenfallen sollen. Für alle Kopf- und Datenzellen innerhalb dieses table-Elements (`#Bundesliga th,td`) werden eine einheitliche Schriftgröße und Zellinnenabstände definiert. Wie an diesem Beispiel erkennbar, gelten für Klassen- und ID-Selektoren die gleichen Möglichkeiten zur Verschachtelung wie bei Selektoren von Elementen. Kopfzellen mit der Klasse `Bundesliga` (`th.Bundesliga`) werden linksbündig ausgerichtet, erhalten eine hellgraue Hintergrundfarbe und als Textfarbe wird Schwarz festgelegt. Es folgen vier reine Klassendefinitionen, nämlich für die Klassen `gelb`, `orange`, `rechts` und `fett`. Solche Klassen, die meist nur ein oder zwei Formateigenschaften beschreiben, sind in der Praxis häufig zu finden und sehr nützlich.

4.7.5 Attributbedingte Formatdefinitionen

Eine weitere Möglichkeit, mit einem Selektor nicht »alle Elemente« mit bestimmtem Elementnamen anzusprechen, sondern nur bestimmte, besteht darin, in den Elementen enthaltene Attribute und sogar Wertzuweisungen bei der Selektion zu berücksichtigen. Dazu müssen die Elemente natürlich mit entsprechenden Attributen ausgestattet sein. Ein weiteres vollständiges Listing eines HTML-Dokuments mit zentralen CSS-Definitionen soll diese Einsatzmöglichkeit demonstrieren:

Listing 4.9: CSS-Formatdefinitionen mit attributbedingten Selektoren

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text/html;
                  charset=ISO-8859-1">
<style type="text/css">
p {
   margin-left:25px;
}
p.Melanie {
   margin-left:10px;
   font-weight:bold;
   color:blue;
}
p.Ute {
  margin-left:10px;
```

```
    font-weight:bold;
    color:red;
  }
  cite[lang] {
    font-style:italic;
  }
  cite[lang=en] {
    color:blue;
  }
  cite[lang=de] {
    color:red;
  }
  </style>
  <title>Formatdefinitionen</title>
  </head>
  <body>
     <p class="Melanie">Melanie:</p>
     <p><cite lang="en">Excuse me, can you tell me the way
                        to the main station?</cite></p>
     <p class="Ute">Ute:</p>
     <p><cite lang="de">Mein Englisch ist schlecht - äh</cite>
        <cite lang="en">my english is bad!</cite></p>
     <p class="Melanie">Melanie:</p>
     <p><cite lang="en">Ah, sorry,</cite>
        <cite lang="de">Hauptbahnhof -</cite>
        <cite lang="en">where?</cite></p>
     <p class="Ute">Ute:</p>
     <p><cite lang="de">Ah - gleich da vorne!</cite>
        <cite lang="en">there!</cite></p>
  </body>
  </html>
```

Der HTML-Teil zwischen `<body>` und `</body>` im Beispiel besteht aus lauter Textabsätzen, die einen Dialog zwischen Melanie und Ute darstellen. Melanie ist angelsächsischer Herkunft, während Ute aus dem deutschsprachigen Raum stammt. Das, was beide jeweils sagen, ist durch cite-Elemente markiert. Dabei sind die englischen Passagen durch das dafür vorgesehene lang-Attribut, das ebenfalls in allen HTML-Elementen mit Textinhalt notiert werden kann, mit `lang="en"` gekennzeichnet. Alle deutschsprachigen Zitatfetzen sind mit `lang="de"` ausgezeichnet. Außerdem gibt es zwei Klassen, `Melanie` und `Ute`, die den Absätzen zugeordnet werden, in denen nur der Personenname steht.

Im style-Bereich werden zunächst Formatdefinitionen für Textabsätze allgemein und dann – einschränkend – für solche mit den Klassennamen `Melanie` und `Ute` notiert. Es folgen drei attributbedingte Selektoren: `cite[lang]`, `cite[lang=en]` und `cite[lang=de]`. `cite[lang]` betrifft alle HTML-Konstrukte `<cite lang="...">...</cite>`, `cite[lang=en]` alle Konstrukte `<cite lang="en">...</cite>` und `cite[lang=de]` alle `<cite lang="de">...</cite>`.

Wird in den eckigen Klammern nur der Attributname notiert, sind also alle Elemente mit diesem Attribut betroffen – unabhängig von dem Wert, der dem Attribut jeweils zugewiesen wird. Soll die Auswahl nur auf Elemente mit dem Attribut und einer bestimmten

Wertzuweisung eingeschränkt werden, wird in den eckigen Klammern das Attribut, ein =-Zeichen und der gewünschte Wert notiert.

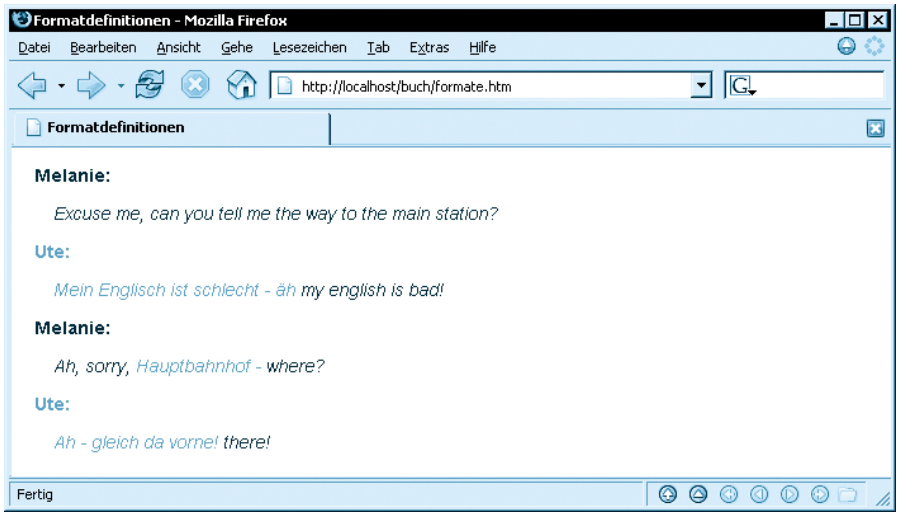


Abbildung 4.58: Attributbedingte CSS-Formatdefinitionen im Einsatz

In unserem Beispiel werden alle englischen Zitatpassagen ebenso wie der Name der englischsprachigen Melanie in Blau dargestellt und alle deutschsprachigen Zitatpassagen ebenso wie der Name der deutschsprachigen Ute in Rot.

So nützlich attributbedingte CSS-Selektoren sind – für die Praxis haben sie einen entscheidenden Nachteil: Der MS Internet Explorer interpretiert diese Syntax bis Version 6.0 noch nicht. Wenn Sie mit solchen Selektoren arbeiten, sollte das also nur unkritische Formate betreffen, bei denen es nicht tragisch ist, wenn sie nicht angezeigt werden können.

4.7.6 Formatdefinitionen für Pseudoelemente

Bisher haben wir Selektoren kennen gelernt, die sich auf konkrete HTML-Konstrukte beziehen. Pseudoelemente tun das nicht. Es handelt sich dabei vielmehr um Bestandteile, die zwar aus logischer oder typografischer Sicht dingfest zu machen sind, aber keine konkrete Entsprechung innerhalb von HTML haben.

Der bekannteste Vertreter für Pseudoelemente ist die Unterscheidung nach Hyperlinks zu bereits besuchten Zielen, zu noch nicht besuchten Zielen (abhängig von der aktuell gespeicherten History des Browsers), zu Links, deren Verweistext gerade mit der Maus überfahren wird, und zu angeklickten Links. Dieses Thema werden wir jedoch in Zusammenhang mit der Gestaltung von Hyperlinks in ***Abschnitt 4.8.9*** behandeln.

Ein anderer Fall sind aus dem Buch- und Zeitungssatz bekannte Hervorhebungen des ersten Buchstabens oder der ersten Zeile eines Absatzes oder einer Überschrift. Dazu ein Beispiel:

```
p.first:first-letter { font-size:300% }
```

In dieser style-Definition lautet der Selektor `p.first:first-letter`. Dabei stellt der Teil `p.first` einen Bezug zu p-Elementen mit dem Attribut `class="first"` her. Der folgende Doppelpunkt markiert, dass nun ein Pseudoelement folgt. CSS bietet eine Reihe solcher Pseudoelemente an. Eines davon ist `first-letter`. Es bedeutet das erste Zeichen eines Elements.

Oben notierter Selektor trifft in HTML also auf eine Notation wie diese zu:

```
<p class="first">Träumend lag ich wach.</p>
```

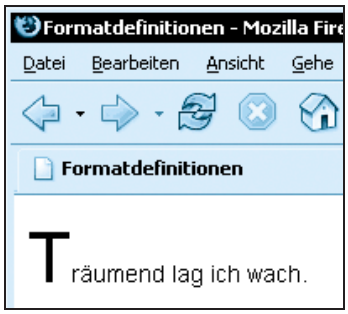


Abbildung 4.59: Hervorhebung des ersten Buchstabens mit Pseudoelement :first-letter

Weitere Pseudoelemente aus diesem Bereich, die CSS zur Verfügung stellt, sind `:first-line` (erste Zeile eines Fließtexts) und `:first-child` (erstes Kindelement eines Elements).



Abbildung 4.60: Vom Text umflossener Anfangsbuchstabe, über :first-letter definiert

Soll der erste Buchstabe übrigens so erscheinen, dass der von ihm eingeleitete Fließtextabsatz um ihn herumfließt, ist dies über zusätzliche Formatdefinitionen lösbar, vor allem zu `float`. Ein Beispiel:

```
p.first:first-letter {
   font-size:300%;
   float:left;
   margin-right:2px;
```

```
    margin-bottom:2px;
    margin-bottom:0px;
}
```

Ein weiterer Typ von Pseudoelementen in CSS ist für »generierten Inhalt« zuständig. Ein generierter Inhalt ist Text oder anderer Inhalt, der im Browser erscheint, aber in HTML nicht explizit notiert wird, so wie etwa die Bullet-Zeichen oder Nummern bei der Notation von `<li>...</li>`. CSS erlaubt es, für beliebige selektierbare Elemente davor oder dahinter automatisch bestimmbaren Inhalt einzufügen. Ein einfaches Beispiel:

```
span.Preis:before { content:"_ " }
```

Dieser Selektor bestimmt für span-Elemente mit dem Attribut `class="Preis"`, dass dem Inhalt des Elements der Wert »_ « vorangestellt wird. Der entsprechende HTML-Code könnte so aussehen:

```
<p><span class="Preis">39,90</span></p>
```

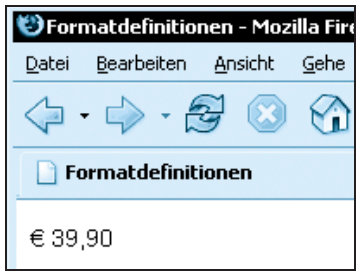


Abbildung 4.61: Über Pseudoelement generiertes Eurozeichen

Zur Verfügung stehen `:before` für »Inhalt davor« und `:after` für »Inhalt danach«. Innerhalb der geschweiften Klammern können Sie, eingeleitet durch `content:`, den gewünschten Inhalt bestimmen. Soll der Inhalt einfach aus Text bestehen, wird dieser in Anführungszeichen hinter `content:` notiert. Anstelle von Text können Sie aber auch z.B. eine Grafik angeben. Das Schema dazu lautet `content:url(URI)`, wobei URI eine Internetadresse oder eine lokal referenzierte Adresse einer Grafik ist.

Das Pseudoelement für »generierten Inhalt« bietet darüber hinaus noch eine Funktion an, die das automatische Nummerieren von Inhalten wie Überschriften erlaubt. Darauf gehen wir in ***Abschnitt 6.2*** genauer ein.

4.8 Hyperlinks

Bislang haben wir in diesem Kapitel typische Elemente zur Textstrukturierung und die Möglichkeiten zu deren Gestaltung kennen gelernt. Das »HT« in HTML steht aber für Hypertext. Ein Kern-Feature von HTML ist nämlich die konsequente Umsetzung des von HTML- und WWW-Vordenker Tim Berners Lee entwickelten Schemas zur Adressierung beliebiger Inhalte im Internet und in lokalen Host-Umgebungen.

Es ist nicht schwer, in HTML »einfach mal« ein paar Links zu notieren. Um Hyperlinks in allen Fällen korrekt einzusetzen, sind jedoch Kenntnisse über den Aufbau von URIs erforderlich. In diesem Abschnitt lernen Sie, wie Sie Quellen richtig adressieren, aber auch, welche zusätzliche Möglichkeiten Sie dem Anwender in Zusammenhang mit Hyperlinks anbieten können und wie Sie Links optisch ansprechend gestalten.

4.8.1 URIs und Links in HTML

Möglicherweise ist Ihnen das Akronym URL (*Uniform Resource Locator* – zu Deutsch: einheitlicher Quellenorter) geläufiger als URI (*Universal Resource Identifier* – zu Deutsch: universelle Quellenbezeichnung). Noch unbekannter dürfte URN (*Uniform Resource Name* – zu Deutsch: einheitlicher Quellenname) sein. Um zu verstehen, was ein URI ist, muss man jedoch alle drei Akronyme im Zusammenhang sehen.

Ein URN sieht aus wie eine typische »Internetadresse«, also z. B. *http://www.example.org/von-mir-persoenlich-erfunden/*. Es handelt sich jedoch nicht um eine existierende Adresse, sondern um einen »universell eindeutigen Namen«, der das Schema der Internetadressierung benutzt, weil sich damit leicht universell eindeutige Namen erstellen lassen.

Ein URL ist eine existierende Internetadresse wie *http://www.mut.de/*. Insofern ist es verständlich, dass vielfach von URLs oder URL-Adressen die Rede ist.

URI ist der Überbegriff für URL und URN. Im Falle existierender Internetadressen bedeutet URI und URL also durchaus das Gleiche. Wegen der URNs jedoch, die vor allem in der XML-Welt eine gewisse Bedeutung haben, spricht die HTML-Spezifikation vorzugsweise von URIs.

Für den Aufbau von URIs gilt folgendes Format:

```
<Schema>://[<Benutzer>[:<Passwort>]@]<Server>[:<Port>]/<Pfad,...> ?<Anfrage>
```

Die nachfolgende Tabelle schlüsselt die Bestandteile dieses Formats auf:

Bestandteil	Bedeutung	Beispiele
Schema	Bezeichnet ein Übertragungsprotokoll oder einen Internetdienst. Bei Übertragungsprotokollen steht am Ende meist `://`, bei Diensten ohne Protokollangabe oft nur ein `:`.	`http://` (HTTP-Übertragungsprotokoll) `ftp://` (FTP-Übertragungsprotokoll) `mailto:` (Internetdienst »E-Mail«) `news:` (Internetdienst »Newsgroups«) `telnet://` (Telnet-Übertragungsprotokoll)
Benutzer	Enthält einen Benutzernamen für einen persönlichen Zugang.	`http://orion:geheim@localhost/` In diesem Beispiel ist `orion` der Benutzername und `geheim` das Passwort, um z. B. direkt eine durch ***.htaccess*** geschützte Seite aufzurufen. Bitte jedoch niemals öffentlich verwenden, da das Passwort im Klartext übertragen und leicht ausspionierbar ist!

Tabelle 4.14: Bestandteile eines URI

Bestandteil	Bedeutung	Beispiele
Passwort	Enthält das Passwort für einen persönlichen Zugang. Von der Verwendung ist dringend abzuraten!	
Server	IP-Adresse oder Hostname (bei Übertragungsprotokollen, die auf TCP oder UDP basieren).	`http://localhost` (Servername ist in diesem Fall »localhost«) `ftp://192.168.101.45` (Server ist in diesem Fall der durch die IP-Adresse angegebene Rechner)
Port	Portnummer, an der der Server »lauscht«, sofern es sich nicht um die Standardportnummer des Übertragungsprotokolls handelt.	`http://localhost:8085` (der Webserver »lauscht« hier nicht wie üblich an Port 80, sondern an Port 8085)
Pfad	Pfadname der adressierten Quelle aus Sicht des Servers.	`http://localhost/` (adressierte Quelle ist hier /, also die Default-Datei im Wurzelverzeichnis für Webdokumente, das im Webserver konfigurierbar ist) `http://localhost/data/impressum.htm` (adressierte Quelle ist hier ***/data/impressum.htm***)
Anfrage	Parameter, die der Quelle übergeben werden (bei HTTP sind dies GET-Parameter).	`http://localhost/main.php?view=std` (das PHP-Script bekommt hier einen Parameter namens »view« mit dem Wert »std« übergeben).

Tabelle 4.14: Bestandteile eines URI (Fortsetzung)

Links zu weiteren Informationen rund um URIs:

Wikipedia-Artikel zu URIs:
http://de.wikipedia.org/wiki/URI
RFC 2396 zu URIs:
http://www.faqs.org/rfcs/rfc2396.html
W3C-Übersichtsseite zur Adressierung mit URIs:
http://www.w3.org/Addressing/

HTML-Notation von Hyperlinks

Wenn Sie in HTML Ressourcen auf einem anderen Server im Internet adressieren wollen, müssen Sie auf jeden Fall einen hinreichend vollständigen URI angeben. Entsprechende Links in HTML könnten etwa so aussehen wie im folgenden Quelltextausschnitt:

```
<h1>Links für HTML-Kenner</h1>
<ul>
<li><a href="http://www.w3.org/TR/xhtml11/">XHTML 1.1 Recommendation</a></li>
<li><a href="http://www.w3.org/TR/xhtml1/">XHTML 1.0 Recommendation</a></li>
<li><a href="http://www.w3.org/TR/html401/">HTML 4.01 Recommendation</a></li>
</ul>
```

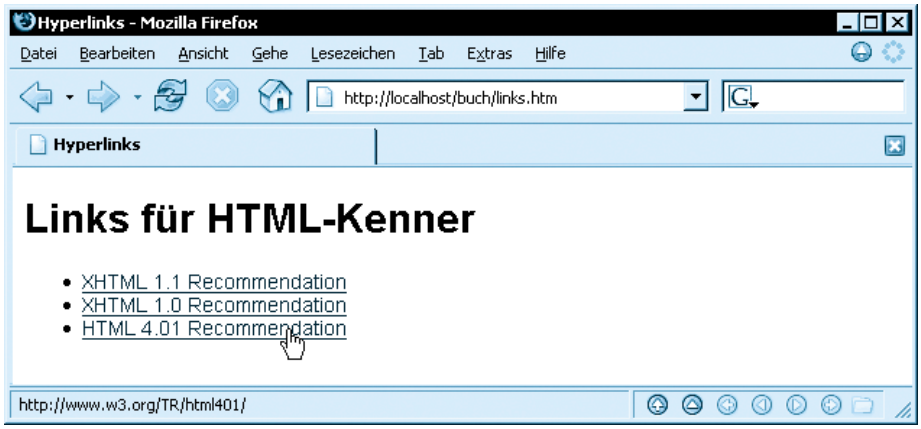


Abbildung 4.62: Einfache Links im Browser

HTML-Links haben immer den gleichen Aufbau:

```
<a href="Zieladresse">Anklickbarer Verweisinhalt</a>
```

Das a-Element ist für Links zuständig. Es kann in Block- und anderen Inline-Elementen vorkommen und selbst neben Text auch andere Inline-Elemente (mit Ausnahme anderer a-Elemente) enthalten. Erlaubt sind also z. B. HTML-Konstrukte wie diese:

```
<h2>Spezial-Domains wie <a href="http://www.example.org/"> example.org</a></h2
Wikipedia hat auf alles eine <a href=" http://de.wikipedia.org/wiki/
Antwort"><em>Antwort</em></a>
<a href="http://de.news.yahoo.com/"><img src="http://eur.i1.yimg.com/
eur.yimg.com/i/de/ne/nws6b.gif" alt="Yahoo Nachrichten"></a>
```

Im ersten Beispiel steckt der Link in einer h2-Überschrift. Entsprechend dem Vererbungsprinzip übernimmt der Link bei der Darstellung z. B. die Schriftgröße der Überschrift.

Im zweiten Beispiel enthält der Linktext selbst ein anderes Inline-Element (`em`). Das Element wird wie üblich dargestellt, übernimmt jedoch die Darstellungsform für Hyperlinks.

Zu den Inline-Elementen gehört auch das img-Element, das zum Referenzieren von Grafiken in HTML dient. Als Elementinhalt eines Hyperlinks notiert, wird die Grafik selbst zum anklickbaren Link. Den Rahmen, den der Browser um eine auf diese Weise anklickbare Grafik zieht, bekommen Sie weg, indem Sie im `<img>`-Tag den Rahmen unterdrücken – am besten über die CSS-Syntax `style="border:none"`.

Das Konstrukt `<a...><img...></a>` ist in der Praxis übrigens extrem verbreitet – so basieren z. B. anklickbare Werbebanner darauf oder Logos, die bei Anklicken z. B. zur Startseite der Webpräsenz führen.

Links auf dynamische Inhalte

Hinter zahlreichen Webseiten steckt kein statischer Inhalt, wie z. B. eine fest abgespeicherte HTML-Datei, sondern ein Script oder Programm (z. B. ein PHP-Script), das variable HTML-Inhalte erzeugt und an den aufrufenden Browser zurücksendet. Dazu erwartet ein solches Script meist entsprechenden Input. In vielen Fällen steht dieser Input direkt im URI, und zwar in den so genannten GET-Parametern, eingeleitet durch ein Fragezeichen hinter der Pfadangabe des Scripts. In einem HTML-Link können Sie als Verweisziel auch solche URIs angeben. Beispiel:

```
<a href=" http://www.google.de/
search?hl=de&q=Meteoriten&btnG=Suche&meta=">Google-Suche nach Meteoriten</a>
```

Am einfachsten bringen Sie einen solchen URI in den eigenen HTML-Quelltext, indem Sie im Browser die Zielsite aufrufen (im Beispiel also Google) und dort den gewünschten Zustand herstellen (im Beispiel also nach `Meteoriten` suchen). Wenn die Seite mit Suchtreffern angezeigt wird, kopieren Sie den Inhalt der Adresszeile des Browsers in die Zwischenablage. Den kopierten Inhalt fügen Sie dann im Editor als Zuweisung an das href-Attribut ein.

Testen Sie Links auf solche URIs jedoch aus und überprüfen Sie, ob Sie tatsächlich die gewünschten Inhalte erhalten. Denn in vielen dynamischen URIs werden auch Parameter wie Session-IDs mitgeschleppt, die sich nicht für einen Link eignen. Probieren Sie gegebenenfalls auch mal, solche Parameter-Wert-Kombinationen wegzulassen.

Linktypen und Darstellung von Links

Es mag erstaunen, dass alle Links in HTML über das gleiche HTML-Element nach dem gleichen Schema notiert werden. Denn auf modernen Webseiten gibt es offensichtlich ganz unterschiedliche Typen von Links, z. B. anklickbare Werbebanner oder Schaltflächen in einer Site-internen Navigationsleiste und Hyperlinks im normalen Fließtext.

Der Vorteil dieser Lösung ist auf jeden Fall, dass Sie nur ein Verlinkungsschema kennen müssen, das auf alle Fälle passt. Der Nachteil ist, dass Sie die unterschiedliche Bedeutung der Links nur durch optisch unterschiedliche Gestaltung oder Platzierung auf der Webseite kenntlich machen können.

Wir werden in diesem Abschnitt zunächst noch näher auf die möglichen Fälle und Besonderheiten bei der Adressierung von Verweiszielen eingehen. In ***Abschnitt 4.8.9*** werden Sie dann mit den Möglichkeiten der optischen Gestaltung von Links vertraut gemacht. Weil speziell Navigationsleisten auf modernen Websites große Bedeutung haben, wird dieses Thema in ***Abschnitt 5.2*** außerdem noch gesondert behandelt.

Hinweise zu Links auf fremde Quellen

Durch Hyperlinks in Verbindung mit dem Internet und dem Adressierungsschema der URIs ist im Prinzip alles mit allem anbieterübergreifend vernetzbar. Deshalb sind Hyper-

links in HTML gewissermaßen der Schlüssel zur Wunschvorstellung der Informationsgesellschaft, die immer mehr zum Paradigma unseres Zeitalters wird.

Doch leider ist die Realität nicht so einfach. Ein Link ist schnell gesetzt, kann jedoch ebenso schnell zivil- oder gar strafrechtliche Folgen haben. Außerdem gibt es gewisse Anstandsregeln. Die folgenden Hinweise sollten Sie auf jeden Fall beachten:

- ***Kostenpflichtige Links:*** Manche Anbieter erlauben Links nur gegen Entgelt, z. B. einige Anbieter von Karten- oder Straßenplanmaterial. Wenn Sie Links auf solche URIs ohne Kenntnis des Anbieters setzen (z. B. weil Sie auf diese Weise Ihren Besuchern eine Anfahrtsskizze präsentieren wollen), ist das genauso wie Schwarzfahren mit der U-Bahn oder Parken auf einem kostenpflichtigen Parkplatz ohne Parkschein. Das Vergehen kann vom Anbieter durch Einschalten eines Anwalts oder gerichtlich verfolgt werden.
- ***Links auf illegale Inhalte:*** Im Internet tummeln sich etliche Angebote mit Quellen, die zwar als URI adressierbar sind, deren Bereitstellung jedoch eine Straftat darstellt. Das betrifft z. B. illegale pornografische Inhalte wie Kinderpornos oder Downloadangebote von Vollversionen registrierungspflichtiger Software. Wenn Sie Links zu solchen Quellen setzen, können Sie im Rahmen der Mitstörerhaftung bei Entdecken juristisch genauso belangt werden wie der Anbieter selbst. Dabei können je nach Schwere des Falls mehrjährige Haftstrafen drohen!
- ***Deeplinks:*** Einige Anbieter wünschen keine direkten Links auf Unterseiten ihres Angebots, sondern nur auf die Startseite. Manchmal steckt technisches Unvermögen dahinter (z. B. weil der Anbieter Frames einsetzt und direkt adressierte Unterseiten plötzlich ohne Navigation, Werbebanner und Logo angezeigt werden). In anderen Fällen ist es dem Anbieter einfach unangenehm, wenn ein Besucher zuerst sein »Kellerregal« zu sehen bekommt und nicht das protzige Eingangsportal. Es gibt zwar kein Gesetz, das Deeplinks verbietet, doch ein Anbieter, der explizit keine solchen Links wünscht, könnte auf jeden Fall einen Anwalt einschalten und ein Gerichtsverfahren anstrengen. Bei Deeplinks auf fremde Webangebote empfiehlt es sich deshalb, dort nach publizierten Geschäftsbedingungen (AGB), nach einem Impressum oder nach einer anderen Seite mit rechtlichen Hinweisen zu suchen, um sich über eventuell unerwünschte Deeplinks zu informieren. Der Rest ist eigenes Risiko. Es gibt sogar gute Gründe, sich über ein Deeplink-Verbot hinwegzusetzen, nämlich die Zivilcourage, die nötig ist, damit die Linkfreiheit im Netz nicht den Zwangsvorstellungen von einzelnen Anbietern geopfert wird.
- ***Links zu kleineren Sites:*** Wenn Sie selbst eine von Besuchern sehr stark frequentierte Seite anbieten und darauf einen Link auf ein fremdes Angebot setzen, ist damit zu rechnen, dass durch den Link viele Besucher zu dem fremden Angebot surfen. Die meisten Anbieter müssen bei ihren Hosting-Providern jedoch für Traffic ab einer bestimmten Grenze mehr bezahlen – das Prinzip ist ähnlich wie beim Stromverbrauch. Durch Ihren Link können Sie dem fremden Anbieter also zusätzliche und in der Höhe unkontrollierbare Kosten verursachen, indem Sie gewissermaßen seine Lichter auf seine eigenen Kosten dauernd ein- und ausschalten. Wer ein Webangebot unterhält,

muss zwar stets damit rechnen, dass Besucherzahlen und Traffic auch mal stark anschwellen können. Doch es gehört zum guten Ton, sich in entsprechenden Fällen vor dem Setzen eines Links mit dem Fremdanbieter zu verständigen und dessen Einwilligung einzuholen.

Wenn Sie diese Punkte beachten, brauchen Sie jedoch auch nicht übervorsichtig zu sein. Sie brauchen beispielsweise keine Mail mit Lesebestätigung und Wichtig-Flag an Google zu schicken, wenn Sie beabsichtigen, auf ein Suchergebnis von deren Suchmaschine zu verlinken.

Link zum Thema Links auf fremde Quellen:

Telepolis-Artikel »Linkhaftung: Gesetzgeberische Untätigkeit schafft endlich Klarheit«:
http://www.heise.de/tp/r4/artikel/12/12721/1.html

4.8.2 Links auf lokale Quellen

Unter einer »lokalen Quelle« verstehen wir Links, deren Ziel ohne Angabe eines Protokolls und Servers adressierbar ist. Aus dem Format eines URIs werden in diesem Fall nur der Pfad und gegebenenfalls der Anfrage-Teil benötigt.

Relative Adressierung

Der einfachste Fall ist der, dass sich das Verweisziel im gleichen Verzeichnis befindet wie das HTML-Dokument mit dem Verweis. Angenommen, der Verweis befindet sich in einer Datei namens *index.html* und führt zu einer Datei namens *impressum.html* im gleichen Verzeichnis. Im HTML-Link genügt dann folgende Adressierung:

```
<a href="impressum.html">Impressum</a>
```

Wenn sich das Verweisziel in einem Unter- oder einem Unterunterverzeichnis relativ zum verlinkenden Dokument befindet, sind Angaben dieser Form möglich:

```
<a href="team/julia.html">Julia Anders</a>
<a href="projekte/ws2004/uebersicht.html">Projektübersicht Wintersemester 2004/
2005</a>
```

Das Trennzeichen für Verzeichnisse in Pfadangaben ist dabei stets ein einfacher Schrägstrich (/). Das gilt auch dann, falls Sie unter MS Windows arbeiten, wo das entsprechende Trennzeichen normalerweise der Backslash (\) ist.

Falls sich das Verweisziel eine Verzeichnisebene oberhalb befindet oder in einem Verzeichnis unterhalb einer übergeordneten Verzeichnisebene, sind Adressierungen der folgenden Art möglich:

```
<a href="../impressum.html">Impressum</a>
<a href="../../projekte/ws2004/uebersicht.html">Projektübersicht Wintersemester
2004/2005</a>
```

Eine Verzeichnisebene oberhalb wird durch die Zeichenfolge `../` notiert. Eine Angabe wie im zweiten Beispiel bedeutet: zwei Verzeichnisebenen höher und dort im Unterverzeichnis *projekte* das Unterverzeichnis *ws2004* und dort die Datei *uebersicht.html*.

Das aktuelle Verzeichnis kann übrigens durch die Zeichenfolge `./` explizit angesprochen werden. Eine href-Wertzuweisung wie `./impressum.html` ist identisch mit `impressum.html`.

Absolute Adressierung

Alle zuvor angeführten Beispiele verwenden eine relative Adressierung ihres Verweisziels. Ebenso möglich ist jedoch eine absolute Adressierung wie in den nachfolgenden Beispielen:

```
<a href="/team/julia.html">Julia Anders</a>
<a href="/impressum.html">Impressum</a>
```

Eine absolute Adressierung beginnt mit einem Schrägstrich. Dahinter folgt der vollständige Pfad zur gewünschten Quelle. Unklar bleibt dabei allerdings der Bezugspunkt. Auf einem Windows-PC bedeutet eine solche Adressierung: ausgehend vom Wurzelverzeichnis des aktuellen Laufwerks. Auf einem Unix-basierten Betriebssystem bedeutet die Adressierung: ausgehend vom Wurzelverzeichnis des aktuellen Dateisystems (darunter können auch externe Gerätelaufwerke, Netzwerkpfade usw. fallen, sofern sie entsprechend gemountet wurden). Befindet sich das HTML-Dokument mit dem Link dagegen in einer HTTP-Umgebung, erkennbar daran, dass die Adresszeile im Browser, der das Dokument anzeigt, mit *http://...* beginnt, so bedeutet die absolute Adressierung noch etwas anderes. Der Bezugspunkt ist in diesem Fall das im Webserver konfigurierbare *document-root*-Verzeichnis (Wurzelverzeichnis für Webseiten), und zwar das document-root-Verzeichnis des »implizierten« Hosts. Das kann z.B. auch ein virtueller Host sein, eine Subdomain usw.

Bei absoluter Adressierung sollten Sie deshalb wissen, was Sie tun. Falls Sie z.B. in Ihrer lokalen Entwicklungsumgebung einen Webserver fahren und gegebenenfalls die Virtual-Host-Konfiguration mit derjenigen der Produktivumgebung auf dem öffentlichen Webserver abgleichen, werden absolute Pfadangaben bei lokalen HTTP-Aufrufen ebenso aufgelöst wie auf dem öffentlichen Webserver. Falls Sie ohne lokale HTTP-Umgebung Seiten erstellen, die auf einen öffentlichen Webserver hochgeladen werden sollen, ist es besser, mit relativer Adressierung zu arbeiten. Und wenn Sie Seiten etwa für eine HTML-basierte CD/DVD-ROM-Produktion erstellen, ist relative Adressierung oberste Pflicht bei der Linkadressierung.

4.8.3 Anker und Links zu Ankern

Für Seiten mit längerem Inhalt stellt HTML ein nützliches Feature bereit: Links zu dateiinternen Ankern. Viele Webseiten enthalten beispielsweise neben der Seitenüberschrift eine ganze Reihe von Unterüberschriften, weil es der umfangreiche Inhalt erfordert. Dabei ist es für Leser angenehm, am Beginn der Seite gleich anklickbare Links auf die Unterüberschriften angeboten zu bekommen. Die Links helfen bei der Orientierung, was auf der aktuellen Seite zu finden ist, und ermöglichen außerdem den gezielten Sprung zu einer bestimmten Unterüberschrift.

Zunächst das Beispiel eines Links zu einem Anker innerhalb des aktuellen HTML-Dokuments, also innerhalb der aktuell im Browser angezeigten Webseite:

```
<a href="#spieltag_12">Ergebnisse des 12. Spieltags</a>
```

Anker werden durch das Gatterzeichen (#) adressiert. Unmittelbar hinter dem Gatterzeichen folgt der Name des Ankers, zu dem bei Anklicken des Links gesprungen werden soll. Innerhalb des Dokuments muss natürlich ein entsprechender Anker definiert sein. Zum Definieren von Ankern wird ebenfalls das a-Element benutzt, jedoch mit einem name-Attribut anstelle des href-Attributs. Die HTML-Umgebung des Verweisziels aus dem obigen Beispiel könnte so aussehen:

```
<h3><a name="spieltag_12"> Ergebnisse des 12. Spieltags</a></h3>
```

Der Anker wird also innerhalb einer Überschrift platziert. Das schließende `</a>` könnte auch weiter vorne notiert werden, sogar unmittelbar hinter dem Start-Tag. Doch ein solcher »leerer« Anker wird von diversen Browsern aus unerfindlichen Gründen nicht erkannt, weshalb empfohlen wird, Anker-Elemente stets mit Inhalt zu versehen.

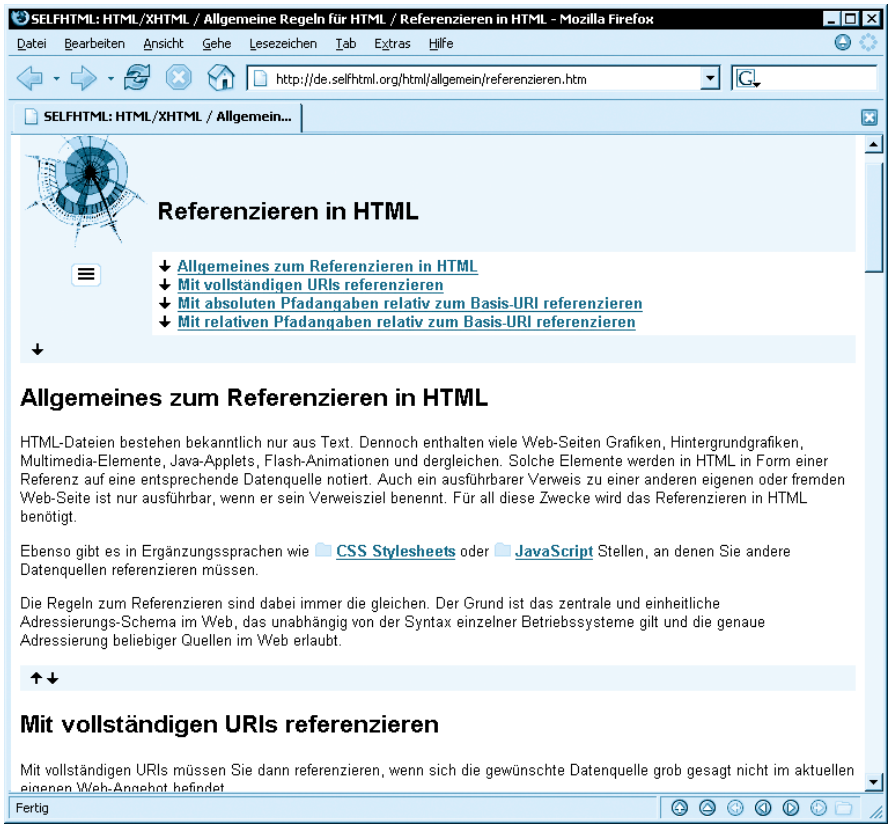


Abbildung 4.63: Links auf Anker innerhalb der Seite (hier in der Dokumentation SELFHTML)

Ankernamen können Sie frei vergeben. Folgendes ist jedoch zu beachten:

- Ankernamen müssen dokumentweit eindeutig sein.
- Das name-Attribut teilt mit dem id-Attribut den gleichen Namensraum. Falls Sie also Elemente mit id-Namen versehen, müssen alle id-Namen und Wertzuweisungen an das name-Attribut unterschiedlich sein.
- Groß-/Kleinschreibung wird unterschieden. Die Namen *WS_2004* und *ws_2004* sind unterschiedlich.
- Das name-Attribut darf auch HTML-Entities wie `ü` enthalten.
- Das erste Zeichen muss ein Buchstabe A–Z oder a–z sein.
- Nachfolgende Zeichen dürfen Buchstaben, Ziffern, der Unterstrich, der Bindestrich, der Doppelpunkt oder der Punkt sein. Im Hinblick auf andere Verwendungen, etwa mit JavaScript, ist jedoch dringend zu empfehlen, kein anderes Sonderzeichen als den Unterstrich zu verwenden.

Es ist übrigens erlaubt, beide Attribute `name=` und `href=` in einem a-Element zu notieren. Damit ist das Element zugleich ein Hyperlink und auch ein Anker, der anderen Hyperlinks als Verweisziel dienen kann.

Es ist auch möglich, Anker in anderen Dokumenten anzuspringen. Dabei ist es unerheblich, wie das Dokument selbst adressiert wird, ob mit einer vollständigen URI-Angabe oder lokal relativ oder absolut. Am Ende wird einfach das Gatterzeichen, gefolgt vom Ankernamen, notiert. Ein Beispiel:

```
<a href="http://www.w3.org/TR/html401/#minitoc">Inhalt der HTML 4.01
Spezifikation</a>
```

4.8.4 Links auf Default-Dateinamen

In HTTP-Umgebungen ist es möglich, bei Links auf bestimmte Dateien den Dateinamen einfach wegzulassen. Wenn beispielsweise bei Eingabe von *http://www.w3.org/* in der Adresszeile des Browsers eine Seite aufgerufen wird, so liegt dies daran, dass in der Konfiguration des Webservers von *w3.org* bestimmte Dateinamen, wie z.B. *index.htm*, *index.html*, *welcome.html* oder *contents.html* als Default-Dateinamen eines Verzeichnisses für Webdokumente eingerichtet wurden. In HTTP-Umgebungen sind also auch Adressierungen wie `href="/"` oder `href="/projekte/2004/"` möglich. Wenn dort eine Datei mit gültigem Default-Dateinamen existiert, sendet der Webserver automatisch diese Datei an den aufrufenden Browser.

Die Adressierung von Default-Dateinamen ist durchaus kombinierbar mit einem Anfrageteil oder mit Ankeradressierung. So entstehen Adressierungen wie `href="/?y=2004&m=12"` oder `href="/projekte/#2004"`.

In ***Kapitel 9*** werden wir das Thema Default-Dateinamen noch näher behandeln, dort in Zusammenhang mit der Konfiguration eines Webservers.

4.8.5 Links auf beliebige Inhaltstypen

Die meisten HTML-Links führen sicherlich zu anderen HTML-basierten Seiten. Es gibt aber keine Vorschrift dafür, dass ein Verweisziel von diesem Dateityp sein muss. Sie können ebenso gut Links auf PDF-Dokumente, Excel-Tabellen, ZIP-Archive, Grafik- oder Multimediadateien setzen. Beispiele:

```
<a href="../jahresbericht-2005.pdf">Jahresbericht (PDF)</a>
<a href="http://www.meta-text.net/selfhtml81.zip">Download SELFHTML 8.1</a>
<a href="/bilder/sonnenuntergang-2.tif">Sonnenuntergang (1200 x 800 Pixel)</a>
```

Der Link ist also nicht das Problem. Das Problem besteht allein darin, was beim Anwender passiert, wenn er auf einen solchen Link klickt. Dazu wird der zugehörige Mime-Type bestimmt. Wird das Linkziel über einen Webserver angefordert (HTTP-Umgebung), entscheidet die Kommunikation zwischen Webserver und Browser darüber, welches der korrekte Mime-Type ist. Existiert keine HTTP-Umgebung, entscheidet der Browser allein, und zwar auf Grund ermittelbarer Daten wie der Dateiendung. Außerdem besteht noch die Möglichkeit, als HTML-Autor die Entscheidung zu beeinflussen, indem man im Link den gewünschten Mime-Type explizit angibt. Dazu kann im `<a>`-Tag das Attribut `type=` notiert werden. Die obigen Beispiele nochmals, doch diesmal mit expliziter Typangabe:

```
<a href="http://www.meta-text.net/selfhtml81.zip" type="application/
zip">Download SELFHTML 8.1</a>
<a href="/bilder/sonnenuntergang-2.tif" type="image/tiff">Sonnenuntergang (1200
x 800 Pixel)</a>
```

Eine Liste gültiger Mime-Typen stellt die Organisation IANA (Internet Assigned Numbers Authority) unter *http://www.iana.org/assignments/media-types/* zur Verfügung.

In der Konfiguration eines Browsers ist eine Liste mit ihm bekannten Mime-Typen gespeichert. Für die einzelnen Mime-Typen ist auch gespeichert, wie der Browser mit Inhalten dieses Typs verfahren soll. Außerdem ist eingestellt, was der Browser mit Daten tun soll, deren Mime-Type er nicht kennt. Da diese Einstellungen vom Endanwender ganz oder teilweise bearbeitbar sind oder – wie beim MS Internet Explorer der Fall – eng mit den Dateiverknüpfungsinformationen des Betriebssystems verzahnt sind, ist nicht vorhersehbar, wie der Browser reagiert.

Hat ein Anwender beispielsweise MS Office oder OpenOffice installiert, könnte sein Browser, falls er ein Dokument mit dem Mime-Type ***application/msexcel*** (also eine Excel-Datei) erhält, direkt die betreffenden Office-Programme (MS Excel bzw. OpenOffice Calc) mit den heruntergeladenen Daten aufrufen, eleganterweise sogar über Schnittstellen wie OLE direkt im Browserfenster. Ist bei einem Anwender jedoch kein Officepaket installiert, das mit Dateien dieses Typs etwas anfangen kann, wird der Browser dem Anwender vermutlich in einem Dialogfenster anbieten, das Dokument herunterzuladen und als Datei unter einem frei wählbaren Namen abzuspeichern.

Bei anderen Dateitypen wie etwa Archivdateien (ZIP, gzip, RAR, auch selbstentpackend als EXE) wünscht sich ein Linkanbieter vielleicht, dass der Browser bei Anklicken eines

entsprechenden Links einen Download-Dialog anbietet. Doch auch das ist nicht sicher. Es kann durchaus sein, dass sich bei einem Anwender z. B. gleich ein externes Programm wie WinZip öffnet. Eine Kontrolle hierüber haben Sie als »Linksetzer« nicht, und das ist auch gut so.

4.8.6 Sonderzeichen in URI-Angaben

URIs dürfen eigentlich nur Zeichen aus dem ASCII-Zeichensatz enthalten. Wenn Sie beispielsweise eine Datei *münchen.html* genannt haben und einen Link darauf setzen, dürfen Sie den Dateinamen nicht so verlinken, wie er lautet.

Falsch ist also:

```
<a href="http://meinserver/münchen.html">München</a>
```

Richtig ist dagegen:

```
<a href="http://meinserver/m%FCnchen.html">München</a>
```

Bei allen Zeichen, die einen Unicode-Zeichenwert höher als 127 haben, sollten Sie in URI-Angaben so verfahren. Betroffene Zeichen werden durch ein Prozentzeichen eingeleitet, gefolgt vom Zeichenwert in Hexadezimalschreibweise. Nachfolgende Tabelle listet die höherwertigen Buchstaben für den Zeichensatz ISO-8859-1 auf:

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö		Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	Ï
F	ð	ñ	ò	ó	ô	õ	ö		ø	ù	ú	û	ü	ý	þ	ÿ

Tabelle 4.15: Hexadezimalwerte von Buchstaben aus dem Zeichensatz ISO-8859-1

Suchen Sie in dieser Tabelle einfach den gewünschten Buchstaben. Sein Hexadezimalwert ergibt sich aus dem Kopfbuchstaben der Zeile und dem der Kopfzeichen der Spalte. Für den Buchstaben Ö ergibt sich also beispielsweise D6 und in einem URI müsste dieser Buchstabe in der Form `%D6` notiert werden.

Doch nicht alle Zeichen des ASCII-Zeichensatzes dürfen in URIs innerhalb von Server-, Verzeichnis- oder Dateinamen oder im Anfrageteil (GET-String) eingegeben werden. URI-eigene Sonderzeichen und Leerraumzeichen müssen ebenfalls hexadezimal maskiert werden. Die folgende Tabelle listet die betroffenen Zeichen und ihre korrekte Notation auf:

Zeichen	Notation	Zeichen	Notation
?	%3F	Leerzeichen	%20
=	%3D	Wagenrücklauf	%0D

Tabelle 4.16: Maskierungspflichtige ASCII-Zeichen in URIs

Zeichen	Notation	Zeichen	Notation
%	%25 oder %%	Zeilenvorschub	%0A
:	%3B	Tabulator	%09
/	%2F	&	%26

Tabelle 4.16: Maskierungspflichtige ASCII-Zeichen in URIs (Fortsetzung)

4.8.7 Features für E-Mail-Verweise

Links auf E-Mail-Adressen erfreuen sich großer Beliebtheit. Kein Wunder: Ein Klick und der Anwender kann z. B. dem Anbieter einer Website sofort eine Mail an eine von diesem bestimmte Mail-Adresse schreiben. In HTML sieht ein einfacher Link dieser Art so aus:

```
<p>E-Mail an <a href="mailto:s6m4@yahoo.de">Stefan Münz, s6m4@yahoo.de</a></p>
```

Die Wertzuweisung an das href-Attribut besteht in diesem Fall aus dem Schema `mailto:`, gefolgt von der E-Mail-Adresse.

Eine Garantie darauf, dass solche Links tatsächlich den gewünschten Effekt haben, gibt es jedoch nicht. Der Browser muss wissen, welche lokale Software die Default-Software zum Managen von E-Mails ist. Außerdem muss ihm bekannt sein, wie er das Mail-Programm so aufrufen kann, dass sich darin sofort ein Fenster zum Schreiben einer neuen Mail öffnet. Am einfachsten haben es hierbei natürlich so genannte Browser-Suiten wie die Mozilla-Suite oder die Opera-Suite, die über ein eigenes Mail-Modul verfügen. Bei separaten Mail-Programmen, besonders dann, wenn sie nicht Outlook oder Outlook Express heißen, ist die Gefahr, dass E-Mail-Verweise nicht funktionieren, dagegen deutlich größer.

Angesichts der genannten Gründe ist es zweckmäßig, im Verweistext eines E-Mail-Links stets die Mail-Adresse noch mal explizit anzugeben. So können auch Anwender, bei denen der Link nicht funktioniert, den angegebenen Mail-Kontakt nutzen.

In der oben notierten Form öffnet sich also beim Anwender nach Anklicken des Links ein Fenster zum Eingeben einer neuen E-Mail. Das Adressatenfeld ist vorbelegt mit der im Link angegebenen Mail-Adresse. Da ist es verständlich, dass schon früh der Wunsch aufkam, die Felder für das Mail-Subject, Kopienempfänger oder sogar den Mail-Inhalt mit Text vorzubelegen. Eine entsprechende Syntax im erlaubten Rahmen der URIs hat sich dabei etabliert. Die meisten modernen Browser interpretieren die entsprechende Syntax. Dennoch ist keinesfalls garantiert, dass die Schnittstellenkommunikation mit dem Mail-Programm so klappt, dass alle Felder wie gewünscht vorbelegt werden.

Ein einfaches Beispiel mit Vorbelegung des subject-Felds:

```
<a href="mailto:s6m4@yahoo.de?subject=Feedback%20zum%20Buch">Mail an Stefan
Münz, s6m4@yahoo.de</a>
```

Die optionalen Feldvorbelegungen werden innerhalb des mailto-URIs als Anfrageteil, also getrennt durch ein Fragezeichen, hinter der Mail-Adresse notiert. Wie in GET-Strings üblich, wird jedes gewünschte Feld wie ein Parameter behandelt. Parameter folgen dem

Schema `Name=Wert`. Mehrere Parameter-Wert-Kombinationen werden durch ein &-Zeichen voneinander getrennt: `Name=Wert&Name=Wert&Name=Wert`. Ein Beispiel für mehrere Angaben zur Feldvorbelegung:

```
<a
href="mailto:s6m4@yahoo.de?cc=stefan.muenz@selfhtml.org&subject=Feedback%20zum%2
0Buch">Mail an Stefan Münz, s6m4@yahoo.de</a>
```

In diesem Beispiel werden das subject-Feld (Parameter `subject`) und das cc-Feld (Parameter `cc`) mit entsprechenden Inhalten vorbelegt. Folgende Parameter werden in der Regel erkannt:

- `cc` (sichtbarer Kopienempfänger)
- `bcc` (unsichtbarer Kopienempfänger)
- `subject` (Betreff)
- `body` (Nachrichtentext)

Beachten Sie, dass Sie Sonderzeichen maskieren wie in ***Abschnitt 4.8.6*** beschrieben. Wenn Sie den Nachrichtentext vorbelegen möchten, können Sie darin Zeilenumbrüche durch die Sonderzeichenmaskierung `%0A` erzeugen, z. B.:

```
<a
href="mailto:s6m4@yahoo.de?cc=stefan.muenz@selfhtml.org&subject=Feedback%20zum%2
0Buch&body=Lieber%20Autor,%0A%0AIch%20habe%Ihr%20Buch%20gelesen">Mail an Stefan
Münz, s6m4@yahoo.de</a>
```

4.8.8 Hyperlink-Optimierung

HTML stellt verschiedene Attribute zur Verfügung, deren Aufgabe darin besteht, die Funktionalität von Links zu verbessern.

Zunächst einmal können Sie in einem a-Element die beiden Attribute `rel=` (logischer Bezug) und `rev=` (logischer Rückbezug) notieren. Dabei sind die gleichen Wertzuweisungen möglich wie in Zusammenhang mit dem link-Element, das wir in ***Abschnitt 0*** behandelt haben. Ein Beispiel:

```
<a href="../index.htm" rel="chapter">Kapitel 7</a>
```

Mögliche Angaben für das rel- und das rev-Attribut sind z. B. `contents` (Inhaltsverzeichnis), `chapter` (Kapitel), `section` (Abschnitt), `subsection` (Unterabschnitt), `index` (Stichwortverzeichnis), `next` (Seite vor), `prev` (Seite zurück) und `start` (Startseite).

Welche Auswirkungen eine solche Notation im Browser hat, bleibt der Browser-Software überlassen. Firefox beispielsweise behandelt rel- und rev-Angaben in Hyperlinks genauso wie notierte link-Elemente, d.h., er übernimmt die Bezugsangaben in die als Plug-In erhältliche Navigationsleiste für logische Linkbezüge.

Eine andere Hilfe – sowohl für den Browser als auch für den Anwender – sind nähere Angaben zum Verweisziel. Dazu werden die Attribute `hreflang=` und `charset=` angeboten. Bei `hreflang=` können Sie ein Sprachenkürzel gemäß ISO 3166-1 angeben, also z. B. `de` für deutschsprachig, `en` für englischsprachig oder `ar` für arabischsprachig. Bei `charset` können Sie dem Browser bereits mitteilen, welchen Zeichensatz das Verweisziel benutzt, z. B. ISO-8859-1 oder einen anderen Zeichensatz gemäß der Liste der IANA-Organisation. Ein Beispiel:

```
Der arabische Sender <a href="http://www.aljazeera.net/" hreflang="ar"
charset="windows-1256">Al Dschasira</a>
```

Die Angabe zum charset-Attribut deckt sich mit der entsprechenden Angabe aus dem HTML-Quelltext der verlinkten Seite:

```
<meta http-equiv="Content-Type" content="text/html;
      charset=windows-1256">
```

Beim hreflang-Attribut wird das Sprachenkürzel für »arabisch« angegeben.

Welchen Nutzen ein Browser aus diesen Informationen zieht, bleibt ebenfalls ihm überlassen. Bei den heutigen Browsern haben diese Angaben jedenfalls keine sichtbare Wirkung, obwohl solche Dinge wie das Anzeigen eines Flaggensymbols beim Überfahren des Links mit der Maus denkbar wäre.

Eindeutig auf den Anwender zielen dagegen die möglichen Attribute `tabindex=` und `accesskey=` ab. Beide Attribute sollten, wenn verwendet, bei allen oder zumindest allen wichtigen Links einer Seite notiert werden, da ihre Werte einen logischen Zusammenhang bilden.

Drückt ein Anwender beim Anzeigen der Seite wiederholt die Tabulatortaste, kann er auf diese Weise nacheinander auf die in der Seite enthaltenen Links positionieren. Mit der Enter-Taste kann er den Verweis dann »anklicken«. Das Feature ist für Anwender gedacht, die keine Maus benutzen können oder wollen. Normalerweise wird zunächst auf den ersten Link der Seite positioniert, dann auf den zweiten usw. Durch Angabe des tabindex-Attributs in den Links einer Seite können Sie die Reihenfolge selbst bestimmen. Bei `tabindex=` können Sie als Wert Zahlen zwischen 0 und 32767 angeben. Auf den Link mit der niedrigsten tabindex-Nummer wird zuerst positioniert, dann auf den mit der zweitniedrigsten usw.

Das accesskey-Attribut erlaubt dagegen das direkte »Anklicken«, also Ausführen eines Links über einen Hotkey. Wenn Sie beispielsweise `accesskey="c"` notieren, dann bieten die meisten Browser eine entsprechende Hotkey-Funktionalität an, um den Link auszuführen. Als Wert sollten also Buchstaben oder auch Ziffern zugewiesen werden.

Die Umsetzung im Browser kann jedoch unterschiedlich aussehen, wie die folgende Tabelle belegt:

Plattform / Browser	Umsetzung des accesskey-Attributs
Windows / Internet Explorer	Alt + accesskey + Enter
Windows / Mozilla	Alt + accesskey
Windows / Opera	⇧ + Esc + accesskey
Macintosh / Internet Explorer:	Ctrl + accesskey + Enter
Macintosh / Safari	Ctrl + accesskey
Macintosh / Mozilla	Ctrl + accesskey
Macintosh / Opera	⇧ + Esc + accesskey

Tabelle 4.17: Umsetzung des accesskey-Attributs in verschiedenen Umgebungen

Leider kennen die meisten Anwender die entsprechenden Tastaturkombinationen bei ihren Browsern gar nicht. Jene, die sie kennen und nutzen, werden dem Entwickler der Webseiten jedoch die Verwendbarkeit bei wichtigen Links danken.

4.8.9 Optische Gestaltung von Hyperlinks mit CSS

In den meisten Browsern kann der Anwender die Farbe für Links zu bereits besuchten Seiten und zu noch nicht besuchten Seiten einstellen, und oft auch, ob Links unterstrichen dargestellt werden sollen oder ob sie beim Überfahren oder Anklicken mit der Maus durch optische Änderung reagieren sollen. Per Voreinstellung werden Links zu noch nicht besuchten Seiten in grafischen Browsern meist blau und unterstrichen angezeigt und Links zu bereits besuchten Seiten lila oder dunkelblau und unterstrichen.

Da es sich dabei um »Zustände« handelt und nicht um HTML-Elemente, ist der CSS-Zugriff darauf durch so genannte Pseudoelemente möglich. Speziell für die Gestaltung von Hyperlinks in ihren verschiedenen Zuständen stehen die Pseudoformate `:link`, `:visited`, `:hover` und `:active` zur Verfügung. Ihren Einsatz zeigt das folgende Beispiel. Im zentralen style-Bereich im Dateikopf oder in einer separaten, referenzierten CSS-Datei stehen folgende Definitionen:

```
a:link {
   text-decoration:none; font-weight:bold; color:#E00000;
}
a:visited {
   text-decoration:none; font-weight:bold; color:#E08080;
}
a:hover {
   text-decoration:none; font-weight:bold;
   background-color:#FFFFA0;
}
a:active {
```

```
    text-decoration:none; font-weight:bold;
    background-color:#A0FFFF;
}
```

Der Selektor `a:link` bedeutet »alle a-Elemente (also Hyperlinks) zu noch nicht besuchten Zielen«, `a:visited` »alle Links zu bereits besuchten Zielen«, `a:hover` »Links, auf die der Anwender positioniert« und `a:active` »Links, die der Anwender anklickt«. Bei der Gestaltung, also der Wahl der CSS-Formateigenschaften, können Sie Ihrer Phantasie freien Lauf lassen. Im obigen Beispiel haben wir festgelegt, dass die Links nicht unterstrichen dargestellt werden (`text-decoration:none`), dass sie stattdessen fett dargestellt werden (`font-weight:bold`), dass Links zu noch nicht besuchten Zielen in kräftigem Rot erscheinen (`color:#E00000`), während Links zu besuchten Zielen blassrot angezeigt werden (`color:#E08080`). Links, auf die der Anwender gerade positioniert, egal ob mit der Maus oder mit der Tastatur, erhalten eine hellgelbe Hintergrundfarbe (`background-color:#FFFFA0`) und beim Anklicken eine hellblaue (`background-color:#A0FFFF`).



Abbildung 4.64: Mit CSS gestaltete Links

Vor allem das Nichtunterstreichen von Links (`text-decoration:none`) ist ein immer wieder nachgefragtes Feature.

Mit etwas aufwändigeren Angaben können Sie auch komplette Navigationsleisten inklusive Mouseover-Effekt allein mit HTML und CSS erstellen, d.h. ohne Rückgriff auf Grafiken und JavaScript, wie es früher üblich war. Ein Beispiel dazu liefern wir in ***Abschnitt 5.2***.

4.9 Grafik, Flash und Multimedia

HTML ist als Auszeichnungssprache für Text mit Hypertext-Funktionalität konzipiert. Es bietet keine eigene Auszeichnungsmöglichkeit für grafische Elemente an. Grafiken können ebenso wie beliebige andere multimediale Elemente über entsprechende Schnittstellen-Elemente referenziert werden. Moderne grafische Browser integrieren die referenzierten Inhalte optisch an der Stelle, an der das jeweilige Element notiert ist.

Dieses Konzept drückt allerdings auch eine Gewichtung aus: Grafik und Multimedia sind im W3C-Konzept von HTML optionale, aber keine prägende Features. Deshalb haben immer noch zahlreiche Webdesigner, die eine grafische Ausbildung haben und »grafisch denken«, ihre Probleme mit dem textorientierten Strukturgedanken von HTML. Die Folge sind meist Webseiten, die nur wenig, meist lieblos hingerotzten und fehlerhaften oder von WYSIWYG-Editoren generierten HTML-Code enthalten – und dieser besteht vorwiegend aus Grafik- und Flash-Referenzen. Denn das, was solche Webdesigner sich unter der »Message« oder dem »Feeling« vorstellen, die von Webseiten transportiert werden sollten, ist für sie nur über Grafiken und animierte Effekte realisierbar.

Über die Philosophie von HTML und die Berechtigung des Wunschs nach rein grafischen Webseiten lässt sich vermutlich endlos streiten. Wer sich jedoch auf HTML einlässt, sollte die text- bzw. hypertextorientierte Grundausrichtung dieser Markup-Sprache akzeptieren. Wer komplett grafisch denken will, dem bleibt die Möglichkeit, anstelle von HTML-Seiten gleich Flash-Movies auf seinen Webspace zu laden und gegebenenfalls den Webserver so zu konfigurieren, dass er z.B. auch Dateinamen wie *index.swf* als Default-Dateinamen erkennt und swf-Dateien mit einem entsprechenden HTTP-Header ausliefert. Der aufrufende Browser bekommt dann beim Aufruf nichts anderes als ein Flashmovie gesendet, welches er – so er denn über ein geeignetes Flash-Plug-In verfügt – im gesamten Anzeigefenster darstellen kann. Es spricht nichts gegen eine solche Radikallösung. Auch Tim Berners Lee, der Gründervater des Web, hat stets betont, dass das Web nicht auf bestimmte Dateiformate fixiert sei. HTML ist einfach eine Standardsprache für Webseiten, doch wenn Design-Vorstellungen zu stark mit deren Konzepten konfligieren, ist ein Komplettverzicht auf HTML allemal besser und sauberer als die zahlreichen Sprachvergewaltigungen, die man im Web findet.

4.9.1 Geeignete Grafikformate

Es gibt in HTML keine Vorschrift dafür, welches Format eine eingebundene Grafik haben sollte. Darüber, welche Formate angesagt sind, haben letztlich die grafischen Browser entschieden, und zwar einfach dadurch, welche Formate sie in der Lage sind, ohne Aufruf eines Fremdprogramms direkt anzuzeigen.

Dabei haben sich in den 90er Jahren die beiden Pixel- oder Rastergrafikformate GIF und JPEG etabliert. Beide Formate zeichneten sich dadurch aus, dass sie gut komprimieren konnten und dass sie – zumindest scheinbar – frei verwendbar waren. Beim GIF-Format erwies sich Letzteres allerdings als Trugschluss. Ende der 90er Jahre, als die ganze Welt das Ende der 60er Jahre entwickelte und zunächst vom Online-Dienst CompuServe benutzte, unscheinbare GIF-Format für Millionen von Webgrafiken nutzte, startete der Lizenzinhaber des von GIF benutzten Kompressionsalgorithmus plötzlich eine Kampagne, um die Pfründe abzustecken. Software, die in der Lage war, GIF-Grafiken anzuzeigen oder zu bearbeiten, sollte fortan Lizenzgebühren an den Lizenzgeber zahlen.

Aus diesem Schlamassel ging ein neues, lizenzfreies und offen dokumentiertes Format hervor, das vor allem als Alternative zum GIF-Format entwickelt wurde, zum Teil aber auch mit dem JPEG-Format konkurriert: das PNG-Format. Von der »Szene« im Netz wird

dieses neue Format, welches ebenso wie HTML und CSS vom W3-Konsortium spezifiziert wird, als das neue Universal-Grafikformat gepriesen, und vor allem das GIF-Format gilt in solchen Kreisen mittlerweile als Igitt-Format.

Um sich ein objektiveres Urteil zu bilden, sollte sich jeder Webdesigner mit den Grundeigenschaften und Unterschieden der drei genannten Formate vertraut machen. Die nachfolgende Tabelle listet wichtige Faktoren und Unterschiede auf:

	PNG	JPEG	GIF (89er-Format)
Kompressionsart	verlustfrei	verlustbehaftet	verlustfrei
Kompressionsalgorithmus	zlib-Technologie (lizenzfrei)	JPEG ist ein eigener Kompressionsalgorithmus	lzw-Technologie (lizenziert)
Interlacing	ja, optional	ja, optional (»progressive«)	ja, optional
Transparenz	ja, echte Transparenz	nein	ja, jedoch nur einfarbige Transparenz
Animationsfähigkeit	nein	nein	ja
Max. Farbtiefe	bis zu 48 Bit (281 Trillionen unterschiedliche Farben), herunterskalierbar bis auf 8 Bit	24 Bit (16,7 Mio. unterschiedliche Farben pro Bild)	8 Bit (256 unterschiedliche Farben pro Bild)
Truecolor-Unterstützung	ja	ja, immer	nein
Paletten-Unterstützung	ja, optional	nein	ja, immer
Graustufen-Unterstützung	ja, bis zu einer Farbtiefe von 16 Bit (65536 Graustufen-Palette)	ja, mit einer Farbtiefe von 8 Bit (256 Graustufen)	ja, mit der Farbtiefe von 8 Bit (256 Graustufen)

Tabelle 4.18: Wichtige Unterschiede zwischen den Formaten PNG, JPEG und GIF

Die meisten heute verbreiteten Grafikeditoren und Bildbearbeitungsprogramme für Pixelgrafiken unterstützen beim Abspeichern alle drei Formate. Achten Sie bei der Wahl eines Grafikprogramms aber auch darauf, welche Optionen es beim Abspeichern in diesen Formaten anbietet. Denn viele letztlich ausschlaggebende Faktoren wie die Kompressionsoptionen, Interlacing und andere werden nicht während der Bildbearbeitung, sondern erst beim Abspeichern eingestellt.

Speichern Sie wichtige Grafiken auf jeden Fall auch in einem geeigneten Originalformat. Wenn Sie beispielsweise mit Photoshop oder Paint Shop Pro arbeiten, sollten Sie z. B. Grafiken, die auf der Arbeit mit Layern basieren, stets auch im programmeigenen Format als Sicherungskopie speichern, damit Sie die Grafik zu einem späteren Zeitpunkt wieder optimal bearbeiten können.

4.9.2 Bildbearbeitung in Hinblick auf Webseiten

Dies ist kein Buch über Bildbearbeitung – das Thema kann allein ein Buch gleichen Umfangs füllen. Dennoch sollte jeder, der Websites erstellt und darin Grafiken einbindet, ein paar grundsätzliche Begriffe aus der Bildbearbeitung kennen, die vor allem in Hinblick auf Webdesign von Bedeutung sind.

Vektor- und Raster- bzw. Pixelgrafiken

Die Computergrafik kennt zwei grundsätzliche Modelle, um Grafiken zu beschreiben. Die eine teilt den »canvas«, also die Bildfläche, in lauter Bildpunkte ein und beschreibt, welcher Bildpunkt welche Farbe haben soll. In der Drucktechnik sind dabei Maße wie »dots per inch«, also Bildpunkte pro Inch, verbreitet und die Bildfläche wird als Raster aus Bildpunkten repräsentiert. Daher die Bezeichnung »Rastergrafik«. Die Längen- und Breitenausmaße einer solchen Grafik ergeben sich aus der Punktdichte und der Anzahl vorhandener Bildpunkte. Bei Grafiken, deren Zweck vor allem die Anzeige am Bildschirm ist, entfällt das Thema Punktdichte. Denn hier ist das Maß der Bildpunkte ein Pixel. Bei solchen Grafiken hat sich daher die Bezeichnung »Pixelgrafik« durchgesetzt. Dennoch verfolgen Pixel- und Rastergrafiken das gleiche Beschreibungsmodell.

Anders die Vektorgrafik – deren Beschreibung besteht aus Eigenschaften von Elementen wie Punkt, Linie, Rechteck, Kreis, Ellipse, Polygon und Text. Solche Elemente können Eigenschaften wie Farbe, Farbverlauf, Länge, Höhe, Dicke, Radius, Eckkoordinaten, Schriftart, Transparenzwert usw. besitzen. Aus der Beschreibung kann dann die Darstellung der Grafik »zur Laufzeit« und abhängig vom Ausgabemedium umgesetzt werden.

Raster- und Pixelgrafik eignet sich vorzugsweise für Fotos, während Vektorgrafik optimal für Illustrationen geeignet ist. Es gibt allerdings viele Zwischenbereiche. Firmenlogos, schriftzug-dominierte Werbebanner usw. werden heute als Pixelgrafik in HTML referenziert, obwohl sie vom Typ her meist besser als Vektorgrafik beschrieben werden könnten.

Bekannte Vektorgrafikprogramme sind CorelDRAW oder Adobe Illustrator. Führende Raster-/Pixelprogramme sind Adobe Photoshop oder Jasc Paintshop Pro. Einige Programme, wie Ulead PhotoImpact, bieten im Editiermodus auch Mischformen an.

Farbmodelle, Farbtiefe, Truecolor, Graustufen und Paletten

In den Frühzeiten der Computerbildschirme war Farbe ein Luxus. Seit einigen Jahren jedoch gehören Truecolor-fähige Bildschirme und Grafikkarten zur Grundausstattung jedes PCs. »Truecolor« ist dabei ein irreführender Begriff. Intern erhält auch bei Truecolor-Bildern jeder Bildpunkt eine bestimmte Farbe und die Anzahl möglicher Farben ist begrenzt. Die Grenze ist jedoch so hoch angesiedelt, z. B. bei 16,7 Millionen möglichen Farben, dass man nicht mehr von einer »Palette« bestimmter Farben sprechen kann. Die Auswahl einer Farbe erfolgt stattdessen direkt über die Dialogumsetzung eines Farbmodells.

Ein Farbmodell lernten Sie in Zusammenhang mit dem Definieren von Farben in CSS bereits kennen: das *RGB-Modell*. Dabei wird eine Farbe aus einer additiven Mischung der Grundfarben Rot, Grün und Blau bestimmt. Dieses Modell ist allerdings nicht unumstrit-

ten. Vor allem Grafiker, die sich mit Druckgrafik auskennen, sind mit anderen Modellen vertrauter, etwa mit dem *CMYK-Modell*. Dabei werden Farben als subtraktive Mischung der so genannten Prozessfarben Zyan, Magenta, Gelb und Schwarz definiert. Gute Grafikprogramme bieten beide Modelle zur Bestimmung von Farben an und sind auch in der Lage, Grafiken vom einen Farbmodell ins andere zu konvertieren.

Mit Paletten wird gearbeitet, wenn die Farbtiefe sehr begrenzt ist. Die Farbtiefe errechnet sich daraus, wie viele Byte zum Speichern oder Darstellen des Farbwerts eines Bildpunkts spendiert werden. Bei einem Byte pro Bildpunkt lassen sich 256 verschiedene Farben speichern, da ein Byte, bestehend aus 8 Bit, nicht mehr als $2^8 = 256$ Zustände einnehmen kann. Das GIF-Format kennt nur diese Form der Speicherung. Daher kann eine GIF-Grafik immer nur aus maximal 256 unterschiedlichen Farben bestehen. Welchen Farbwert jede dieser 256 Farben hat, ist pro Bild frei festlegbar. Eben deshalb spricht man von der Palette, die ein Bild verwendet.

Graustufen sind ein Spezialfall, der Schwarzweißbilder beschreibt. Graustufen sind wie der Name schon sagt definierte Grauwerte irgendwo zwischen Schwarz und Weiß. Die Anzahl der Graustufen kann in einer Graustufenpalette dargestellt werden. Die meisten Pixel-Grafikformate unterstützen Graustufen im Rahmen des gewählten Farbmodells und der verfügbaren Farbtiefe. Mit dem RGB-Modell lassen sich letztlich nur 256 Graustufen beschreiben, da bei Graustufen alle drei Farbwerte gleich sein müssen und pro Farbkanal maximal 256 Werte möglich sind.

Transparenz, Layer-Technik und Alpha-Filter

Die Eigenschaft des GIF-Formats, eine Palettenfarbe als transparent zu definieren, erfreut sich seit den Anfängen des Webdesigns großer Beliebtheit. Dieses Feature erlaubt es, Grafiken auf unterschiedlichen farbigen Flächen oder Wallpapers zu platzieren, ohne die Grafik selbst anpassen zu müssen.

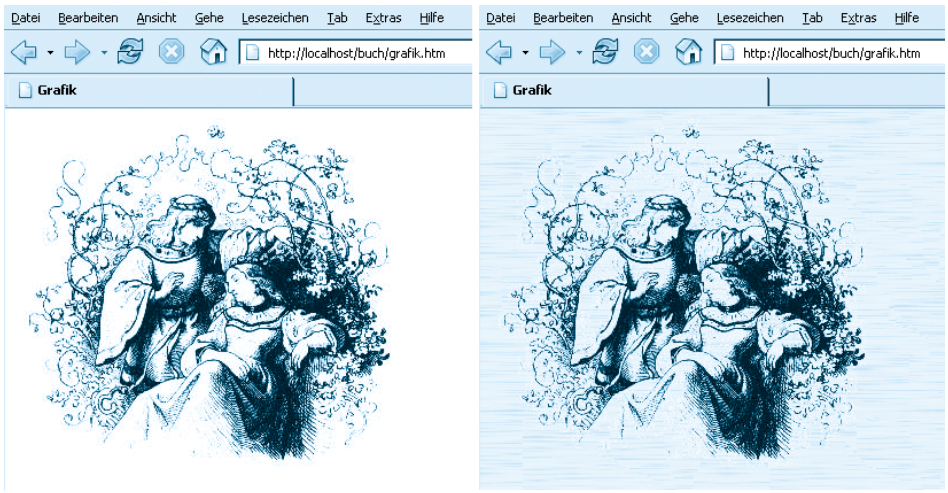


Abbildung 4.65: GIF-Grafik mit Einfarben-Transparenz auf verschiedenen Hintergründen

Die Transparenzfunktion des GIF-Formats ist eigentlich nur ein Spezialfall von Transparenz, die jedoch gerade für Webseiten recht interessant ist. Im abgebildeten Beispiel benutzt die Grafik eine 256-Graustufenpalette. Die Farbe, welche die Hintergrundfarbe des Bilds darstellt, wurde als transparent definiert. Wie die Abbildung zeigt, kann die Grafik dank dieser Transparenz auf verschiedenen Untergründen angezeigt werden.

Anti-Aliasing

Manche Grafiken im Web stellen nichts weiter als einen Schriftzug dar, der ebenso gut mithilfe von CSS realisiert werden könnte. Doch abgesehen von dem Problem der individuellen Schriftarten, die bei Anwendern nicht vorausgesetzt werden können, gibt es noch einen anderen Grund: Grafiken ermöglichen das Arbeiten mit dem Anti-Aliasing-Effekt. Dadurch entstehen abgerundete, weiche, angenehme Kanten, was auffällige Schriftzüge professioneller und eleganter wirken lässt.

Abbildung 4.66: Schriftzug mit Anti-Aliasing (unten: Ausschnittvergrößerung)

Beim Anti-Aliasing werden bei schrägen Strichen oder Rundungen die Treppeneffekte abgeschwächt, indem abgeschwächte Farbtöne hinzugefügt werden. In der Normalansicht wirken schräge Kanten dadurch sauberer.

Dithering

Wenn in einer Grafik die Farbtiefe reduziert wird, muss ein Algorithmus entscheiden, wie die Reduktion arbeiten soll. Das kann beispielsweise auch dann der Fall sein, wenn Sie auf einer Webseite eine Truecolor-Grafik einbinden, diese Webseite jedoch auf einem System angezeigt werden soll, das nur 256 Farben gleichzeitig anzeigen kann. Das könnten ältere Rechner und Bildschirme sein, aber auch modernere, webfähige Kleingeräte wie Handhelds oder Tablet-PCs. Der Webbrowser wird in solchen Fällen in der Regel den so genannten Dithering-Algorithmus benutzen. Dabei werden feine Farbverläufe durch Pixelmuster ersetzt.

Interlacing

Grafiken, die eine gewisse Übertragungs- und Ladezeit benötigen, können durch die Interlacing-Technik schneller angezeigt werden. Dabei wird die Grafikanzeige schichtweise aufgebaut. Für Webseitenbesucher entsteht dadurch bei grafiklastigen Seiten ein subjektiv schnellerer Seitenaufbau. Da alle drei Webgrafikformate das Interlacing unterstützen, ist sehr zu empfehlen, beim Abspeichern von Grafiken diese Option einzuschalten.

Mehrbildertechnik/Animation

Manche Grafikformate erlauben es, mehrere Bilder in einer Grafik zu speichern. So auch das GIF-Format. Dessen Besonderheit besteht jedoch zusätzlich darin, dass auf einer Zeitschiene festgelegt werden kann, wie diese Einzelbilder angezeigt werden sollen. Auf diese Weise werden einfache Animationen möglich, auch in Form von Endlosschleifen. Als die Browser begannen, diese Spezialität zu unterstützen, sorgten ganze Heerscharen verspielter Webdesigner dafür, dass man alsbald kaum noch eine Seite aufrufen konnte, auf der es nicht wie wild blinkte und wackelte.

Mittlerweile sind die so genannten Zappelgrafiken in professionellen Kreisen verpönt. Zum Einsatz kommen sie vor allem noch bei Werbebannern. Es gibt spezielle Grafikprogramme zum Bearbeiten animierter GIF-Grafiken. Einige populäre Grafikprogramme enthalten auch Assistenten für solche Grafiken.

Webgerechte Grafiken

Empfehlungen für webgerechte Grafiken gibt es viele. Einige Hardliner gehen so weit zu fordern, dass webgerechte Grafiken maximal 216 Palettenfarben nutzen, und zwar nur solche, die als websichere Farben gelten. Mit solch drastischen Einschränkungen allerdings sind etwa Fotos kaum befriedigend darstellbar. Folgende Kriterien sollte eine für den Einsatz im Web gedachte Grafik jedoch erfüllen:

- Die Dateigröße der Grafik sollte so klein wie möglich sein.
- Die Grafik sollte sich so schnell wie möglich aufbauen.
- Die Grafik sollte auch auf kleineren Bildschirmen ohne viel Scrollen zu sehen sein.
- Die Grafik sollte den Inhalt einer Webseite ergänzen, nicht ersetzen.
- Die Grafik sollte mit dem Webseitenlayout harmonieren.

Testen Sie beim Speichern einer Grafik verschiedene Qualitätsstufen (Farbtiefe, Art der Farbreduktion, Kompressionsfaktor usw.) und achten Sie auf die dabei entstehenden Dateigrößen. Am Ende steht meist ein Kompromiss aus Grafikqualität und Dateigröße. Grafiken mit mehr als 100 Kbyte sind auch im DSL-Zeitalter nur akzeptabel, wenn sie inhaltstragend sind, z.B. Fotos. Reine Dekorationsgrafiken, Logos, Werbebanner und dergleichen sollten auf jeden Fall deutlich kleiner sein.

Link zum Thema Bildbearbeitung:

Digitale Bildbearbeitung:
http://www.ralphaltmann.de/bildbearbeitung/
Bildbearbeitung elektronisch/digital:
http://www.fotocommunity.de/info/Bildbearbeitung_elektronisch/digital

4.9.3 Grafikreferenzen

Grundsätzlich wird eine Grafik in HTML wie folgt eingebunden:

```
<img src="bild.png" alt="Bildkurzbeschreibung">
```

Das img-Element, ein alleinstehendes Element ohne End-Tag (XML-Notation: `<img.../>`), dient zur Referenzierung einer Grafik. Im obligatorischen src-Attribut wird der URI der gewünschten Grafik angegeben. Es kann sich um eine absolute Internetadresse oder um eine lokal adressierte Grafikdatei handeln.

Das alt-Attribut ist ebenso obligatorisch wie das src-Attribut. Da Grafiken aus verschiedensten Gründen nicht angezeigt werden können – z.B. weil der Anwender die Grafikanzeige in seinem Browser aus Performance-Gründen deaktiviert hat oder weil er die Übertragung der Daten zur Webseite abgebrochen hat, sobald der gewünschte Text lesbar war, oder weil er einen Browser in einer Textmodus-Umgebung wie einer Unix-Shell benutzt. In all diesen Fällen wird anstelle der Grafik der Alternativtext angezeigt, den Sie dem alt-Attribut zuweisen.

Das alt-Attribut ist nicht zu verwechseln mit dem title-Attribut, obwohl manche Browser das alt-Attribut beim Überfahren der Grafik mit der Maus ebenso wie das title-Attribut als Tooltipp-Fenster einblenden. Um einen solchen Effekt gewollt in allen modernen Browsern zu erzielen, sollten Sie das title-Attribut zusätzlich zum alt-Attribut notieren.

In der Regel sollten Sie dem Browser auch noch mehr Daten zur Grafik mitteilen. Dazu gehören vor allem Angaben zu Höhe, Breite und Rahmen, aber auch zum Abstand zu Nachbarelementen, zur Ausrichtung usw. All diese Eigenschaften können Sie mithilfe von CSS zuweisen. Ein Beispiel:

```
<img src="estland.jpg" style="width:300px; height:240px;
     border:none; float:left; margin-right:20px;
     margin-bottom:20px" alt="Typisches Motiv in Estland">
```

Weil die Angabe von Breite und Höhe eine besonders wichtige Information für den Browser darstellt – er kann dann den Raum für die Grafik beim Seitenaufbau bereits reservieren, noch bevor die Daten der Grafik geladen sind –, sind auch im Strict-Standard die beiden HTML-Attribute `width=` und `height=` im img-Element erlaubt. Als Wert wird die Pixelgröße ohne Maßeinheit zugewiesen. Da jedoch meistens noch andere Eigenschaften zu notieren sind, die CSS erfordern, ist es zweckmäßig, auch die Angaben zu Breite und Höhe mithilfe der dafür vorgesehenen CSS-Eigenschaften `width` und `height` zu notieren.



Abbildung 4.67: Grafik mit CSS-Angaben

Das Beispiel weiter oben zeigt, wie eine Grafik mithilfe von CSS im Textfluss ansprechend positioniert wird. Die Angabe `float:left` bewirkt, dass nachfolgender Inhalt rechts um die Grafik herumfließt. Damit der Inhalt nicht unmittelbar an der Grafik klebt, wird mithilfe der Eigenschaften `margin-right` und `margin-bottom` für die betroffenen Seiten ein Sicherheitsabstand von je 20 Pixel definiert. Ein Rahmen wird mit `border:none` explizit unterdrückt.

Das img-Element zählt zu den Inline-Elementen. Eine Grafik wird also, wenn Sie nichts anderes festlegen, mitten im umgebenden Fließtext angezeigt. Nützlich ist dies vor allem bei Kleingrafiken mit Icon- oder Symbolcharakter. Viele Websites benutzen solche Grafiken beispielsweise, um Hyperlinks optisch zu typisieren. Ein Beispiel:

```
<p>
   Sie können den Jahresbericht <img src="doc.gif"
   style="width:14px; height:12px; border:none"
   alt="Webseite"><a href="jb/index.htm">als Webseite</a>
   oder als <img src="pdf.gif" style="width:14px; height:12px;
   border:none" alt="PDF"><a href="jb/jb.pdf"
   type="application/pdf">als PDF-Dokument</a> einsehen.
</p>
```

Um eine Grafik aus dem Inline-Kontext zu lösen, müssen Sie das img-Element entweder in ein Block-Element einbinden, z.B. durch `<div><img...></div>`, oder dem Element mittels CSS explizit Block-Charakter zuweisen (`<img ... style="display:block">`).

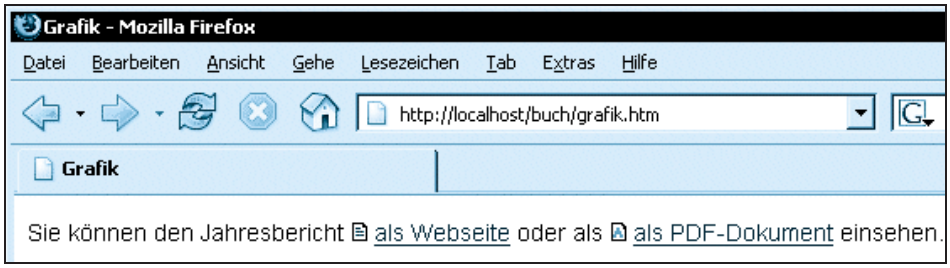


Abbildung 4.68: Kleine Symbolgrafiken im Text

Das img-Element kann natürlich auch in Tabellenzellen, in absolut positionierten Elementen usw. vorkommen. Dies erlaubt zusätzliche Möglichkeiten der Anordnung der Grafik und der Zuordnung von Grafik und anderen Inhalten.

4.9.4 Image-Maps (verweis-sensitive Grafiken)

Anklickbare Grafiken sind auf fast jeder Webseite zu finden. Dies sind einfach Grafiken, die in einen Hyperlink eingebettet und dort anstelle eines Verweistextes notiert werden. Das Schema lautet `<a href=....><img></a>`. Wichtig ist in diesem Fall zu wissen, dass Browser dem img-Element in diesem Fall, sofern Sie nichts anderes angeben, einen Rahmen in der Farbe für Verweise verpassen. Durch `<img ... style="border:none">` können Sie den Rahmen unterdrücken. Bedenken Sie jedoch, dass der Anwender dadurch möglicherweise übersieht, dass die Grafik ein Link ist.

Imagemaps oder verweis-sensitive Grafiken gehen noch einen Schritt weiter als anklickbare Grafiken. Ein Imagemap ist eine Grafik mit anklickbaren Bereichen, wobei jeder Bereich sein eigenes Linkziel haben kann.

Ein Beispiel soll dies demonstrieren. Dabei soll ein vollständig grafisches Menü die Navigation auf einer Startseite ermöglichen. Die Grafik sieht folgendermaßen aus:

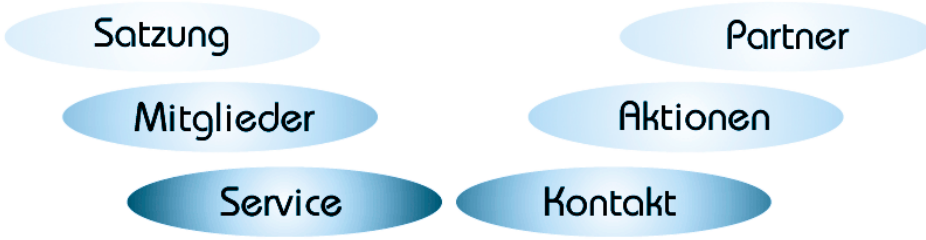


Abbildung 4.69: Grafik für grafische Navigation

Aus den sechs beschrifteten Ellipsen sollen bei der Anzeige im Browser sechs anklickbare Schaltflächen werden, die beim Anklicken zu entsprechenden Verweiszielen führen. Der HTML-Code dazu sieht folgendermaßen aus:

```
<map name="navigation">
   <area shape="rect" coords="5,6,274,66"
         href="/satzung/" alt="Satzung" title="Satzung">
   <area shape="rect" coords="55,72,325,135"
         href="/mitglieder/" alt="Mitglieder"
         title="Mitglieder">
   <area shape="rect" coords="113,146,380,206"
         href="/service/" alt="Service" title="Service">
   <area shape="rect" coords="530,6,800,66"
         href="/partner/" alt="Partner" title="Partner">
   <area shape="rect" coords="451,72,724,135"
         href="/aktionen/" alt="Aktionen" title="Aktionen">
   <area shape="rect" coords="395,146,661,206"
         href="/kontakt/" alt="Kontakt" title="Kontakt">
</map>
<p>
   <img src="navigation.jpg" style="width:807px; height:213px;
   border:none" alt="Navigation" usemap="#navigation">
</p>
```

Die Grafik selbst wird wie im unteren Teil des Quelltextausschnitts zu sehen wie üblich über das img-Element referenziert. Der Bezug zum Imagemap wird über das zusätzliche Attribut `usemap=` hergestellt. Dem Attribut wird der URI einer zugehörigen Map-Definition im gleichen Dokument zugewiesen. Der Wert wird also genauso notiert wie ein Link zu einem Anker innerhalb des Dokuments, mit führendem Gatterzeichen (#), gefolgt vom Ankernamen.

Der Anker, zu dem der Bezug führt, ist das map-Element, markiert durch `<map> ... </map>`. Dieses Element kann irgendwo zwischen `<body>` und `</body>` notiert werden. In der Regel wird es jedoch an zentraler Stelle notiert, z. B. gleich hinter dem einleitenden `<body>`-Tag. Das map-Element dient als Behälter für Definitionen von verweis-sensitiven Flächen.

Das obligatorische name-Attribut im einleitenden `<map>`-Tag erzeugt den Anker, der bei `usemap=` anzugeben ist.

Die Definitionen der verweis-sensitiven Flächen werden durch `<area>`-Tags markiert. Diese haben in HTML kein End-Tag, müssen in XHTML also in der Form `<area ... />` notiert werden.

Die verweis-sensitiven Flächen werden in diesem Tag durch die Attribute `shape=` und `coords=` festgelegt. Das shape-Attribut bezeichnet die Art der Fläche und das coords-Attribut die Pixelkoordinaten der Punkte, mit deren Hilfe sich die Fläche beschreiben lässt. Als Flächenformen stehen `shape="rect"` (Rechteck), `shape="circle"` (Kreis) und `shape="poly"` (Polygon, beliebiges Vieleck) zur Verfügung. Das Schema der Wertzuweisung an das coords-Attribut ist von der Angabe zum shape-Attribut abhängig.

shape=	**Schema für** coords=	**Erläuterung**
rect	x1,y1,x2,y2	x1 = linke obere Ecke, Pixel von links y1 = linke obere Ecke, Pixel von oben x2 = rechte untere Ecke, Pixel von links y2 = rechte untere Ecke, Pixel von oben
circle	x,y,r	x = Mittelpunkt, Pixel von links y = Mittelpunkt, Pixel von oben r = Radius in Pixel
poly	x1,y1,x2,y2,... xn,yn	x = Pixel einer Ecke von links y = Pixel einer Ecke von oben Von der letzten definierten Ecke müssen Sie sich eine Linie zur ersten definierten Ecke hinzudenken. Diese schließt das Polygon.

Tabelle 4.19: Schemata für area-Definitionen verweis-sensitiver Grafiken

Obwohl die zum Anklicken vorgesehenen Flächen in unserem Beispiel elliptisch sind, verwenden wir der Einfachheit halber `shape="rect"` zur Definition der Verweisflächen. Die Flächenform »Kreis« ist jedenfalls ungeeignet. Mithilfe der Flächenform »Polygon« und einer ganzen Kette von Eckpunkten könnten wir die Ellipsen zwar einigermaßen genau beschreiben, doch für den gewünschten Zweck genügt es, gedachte Rechtecke um die Ellipsen als verweis-sensitiv zu kennzeichnen.

Im href-Attribut wird das Linkziel angegeben. Es kann sich um einen beliebigen URI handeln, also um absolute Internetadressen oder um lokal referenzierte Quellen. Ebenso wie im img-Element ist im area-Element die Angabe eines alt-Attributs Pflicht. Der Grund leuchtet ein: Wenn die Grafik nicht angezeigt werden kann, wären auch die Links überhaupt nicht visualisierbar. Das alt-Attribut ermöglicht es dem Browser, anstelle der fehlenden grafischen Links Textlinks mit den alt-Texten als Verweistexten anzuzeigen. Leider machen die meisten Browser keinen Gebrauch davon.

Das title-Attribut haben wir in unserem Beispiel spendiert, damit dem Anwender beim Überfahren der verweis-sensitiven Flächen zusätzlich ein Tooltipp-Fenster angezeigt wird.

4.9.5 Flash-Einbindung in HTML

Keine andere Technologie hat das Web in den letzten Jahren optisch nachhaltiger geprägt als der Siegeszug von Flash. Egal ob für Online-Spiele, animierte Illustrationen, schicke Site-Navigation oder Startseiten-Trailer eingesetzt: Flash hat in all diesen und anderen Bereichen konkurrierende Techniken wie Java-Applets oder auch DHTML verdrängt. Grund genug, die standardkonforme Einbettung eines Flash-Movies in HTML zu behandeln.

Flash-Dateien werden in HTML über das object-Element eingebunden. Ein Beispiel:

```
<object type="application/x-shockwave-flash"
        data="/flash/intro.swf"
        style="width:740px; height:380px">
   <param name="movie" value="/flash/intro.swf">
```

```
    <param name="quality" value="high">
    Sie benötigen einen Flash-Player als Browser-Plugin.
    Ein solches Plugin bekommen Sie auf
    http://www.macromedia.com/software/flashplayer/
</object>
```

Möglicherweise sind Ihnen bereits wesentlich komplexere Quelltextkonstrukte zum Einbetten von Flash begegnet. Diese greifen beispielsweise zusätzlich auf das embed-Element zurück, ein nicht zum HTML-Standard gehörendes Element, das speziell Netscape 4.x bedienen soll. Das ganze HTML-Dokument wird dadurch syntaktisch invalid. Oder Sie haben im einleitenden `<object>`-Tag Attribute wie `classid=` und `codebase=` mit zum Teil merkwürdigen Wertzuweisungen gesehen.

Das oben vorgestellte, standardkonforme Quelltextkonstrukt ist jedoch in heutiger Zeit völlig ausreichend und funktioniert mit allen verbreiteten Browsern. Im einleitenden <object>-Tag sind dabei zwei Attribute wichtig: `type=` muss notiert werden, um den Browser über eine Mime-Type-Angabe darüber zu informieren, welche Art von Daten das Objekt enthält. Der Mime-Type für Flash lautet `application/x-shockwave-flash`. Das Attribut `data=` ist ebenfalls wichtig und muss notiert werden, obwohl die Dateiquelle des Flash-Movies in einem `<param>`-Tag innerhalb des object-Elements nochmals referenziert wird. Anzugeben ist ein URI, also eine vollständige Internetadresse, oder eine lokale Referenz auf die swf-Datei oder die Datenquelle, welche den swf-Code sendet.

Ähnlich wie beim img-Element sind beim object-Element die HTML-Attribute `width=` und `height=` erlaubt, um dem Browser die Pixelausmaße des einzubettenden Objekts vorab mitzuteilen. Diese können aber wie im Beispielcode gezeigt ebenso in CSS-Form angegeben werden.

Als Elementinhalt von `<object> ... </object>` können unter anderem alleinstehende `<param>`-Tags notiert werden (bei XHTML beachten: `<param .../>`). Diese haben die obligatorischen Attribute `name=` und `value=`. Sie definieren Parameter, die an das aufgerufene Objekt übergeben werden. Bei Flash-Movies sollten auf jeden Fall die beiden im Beispiel gezeigten Parameter (`name="movie"` und `name="quality"`) notiert werden. Beim movie-Parameter sollte als Wert nochmals der URI des Flash-Movies angegeben werden. Beim quality-Parameter empfiehlt sich die Standard-Wertzuweisung `high`.

Das object-Element kann als Elementinhalt außerdem Text und andere Block- oder Inline-Elemente enthalten. Diese werden angezeigt, falls das Objekt nicht angezeigt werden kann. Im obigen Beispiel haben wir etwas Text notiert, der einen Anwender, bei dem das Flash-Movie nicht angezeigt werden kann, über die Möglichkeit aufklärt, wie er ein Plug-In erwerben kann.

Ohne Flash-Plug-In können Flash-Movies nicht im Browser angezeigt werden. Die meisten Browser verfügen heute jedoch in der Standardinstallation bereits über ein entsprechendes Plug-In.

Zum Erstellen von Flash-Movies benötigen Sie eine Flash-Autorensoftware. Das Produkt, welches Flash-Erfinder Macromedia selbst anbietet, ist bestens geeignet, aber auch nicht unbedingt preiswert. Es gibt jedoch preiswerte oder kostenlose Drittanbieter-Alternativen.

Flash ohne HTML

Da Flash vektororientiert ist und sich damit problemlos skalieren lässt, kann es beispielsweise für Produktpräsentationen, Spiele und Ähnliches auch sinnvoll sein, das Flash-Movie selbst anstelle eines HTML-Dokuments mit eingebettetem Flash-Movie aufzurufen. In einer vorgeschalteten Seite verlinken Sie dazu einfach die swf-Datei anstelle eines entsprechenden HTML-Dokuments, beispielsweise so:

```
<a href="produkt.swf" type="application/x-shockwave-flash">Produktpräsentation</a>
(Flash erforderlich!)
```

Rückverweise zur übrigen Site sind innerhalb von Flash möglich.

4.9.6 Java-Applets in HTML

Java-Applets sind mittlerweile seltener im Web als vor dem Aufkommen von Flash. Dafür werden Java-Applets häufiger für seriösere und anspruchsvollere Zwecke genutzt, wie etwa Online-Banking, Online-Brokering und Online-Learning.

HTML stellte zunächst ein separates applet-Element zum Einbinden von Java-Applets zur Verfügung. Heute gehört dieses Element noch zur Transitional-Variante von HTML 4.01. In der Strict-Variante ist es nicht mehr vorhanden und auch nicht mehr nötig. Das universeller konzipierte object-Element bietet stattdessen alles, was zum Einbinden von Java-Applets erforderlich ist. Da auch alle modernen Browser mit dieser Methode zurechtkommen, darf das applet-Element mittlerweile mit gutem Gewissen der Vergessenheit anheim fallen.

```
<object classid="java:Beispiel.class"
        codetype="application/java-vm"
        style="width:600px; height:400px">
   Java erforderlich! Die Java-Runtime-Umgebung können Sie
   von http://java.sun.com/ downloaden.
</object>
```

Das object-Element kennen Sie bereits vom Abschnitt über die Einbindung von Flash. Im Fall von Java-Applets müssen im einleitenden `<object>`-Tag jedoch andere Attribute notiert werden. Erforderlich sind die Attribute `classid=` und `codetype=`. Beim classid-Attribut weisen Sie als Wert die Zeichenfolge `java:`, gefolgt vom Namen der class-Datei des Java-Applets zu. Dabei darf nur der reine Dateiname angegeben werden, keine Internetadresse oder Adressierung eines anderen Verzeichnisses. Sollte sich das Applet nicht im gleichen Verzeichnis befinden wie das HTML-Dokument oder auf einem ganz anderen Hostrechner liegen, benötigen sie zusätzlich das Attribut `codebase=`. Diesem Attribut weisen Sie als Wert den URI des Verzeichnisses zu, in dem die class-Datei liegt, also beispielsweise `codebase="../classes/"`.

Beim codetype-Attribut geben Sie den Mime-Type für den Objektcode des Java-Applets an. Dieser Mime-Type lautet standardmäßig `application/java-vm`.

Parameter, die dem Java-Applet beim Start übergeben werden sollen, können über `<param>`-Tags notiert werden. Einem Applet, das z.B. einen Wert zu einem Parameter namens `init_value` erwartet, kann etwa Folgendes mitgeteilt werden:

```
<object...>
   <param name="init_value" value="3">
</object>
```

Link zu Java:

Startseite zur Java-Technologie:
http://java.sun.com/
Tutorial zum Erstellen von Java-Applets:
http://java.sun.com/docs/books/tutorial/applet/
Wikipedia-Artikel zu Java und Java-Applet:
http://de.wikipedia.org/wiki/Java_%28Programmiersprache%29
http://de.wikipedia.org/wiki/Java-Applet

4.9.7 SVG-Vektorgrafiken in HTML

Leider unterstützen selbst nach modernsten Kriterien entworfene Browser wie Firefox bis heute von Haus aus ausschließlich Pixelgrafikformate. Dabei ist Vektorgrafik für Anwendungsfälle wie Illustrationen, technische Zeichnungen, Logos, kunstvolle Schriftzüge usw. eindeutig besser geeignet als Pixelgrafik. In Verbindung mit Scriptsprachen ist auch die Darstellung von Animation und bewegten Abläufen möglich.

SVG (*Scalable Vektor Graphics*) ist ein in den letzten Jahren eigens für den Einsatz im Web entwickeltes Vektorgrafikformat. Die vektorgrafischen Informationen werden in XML-Form gespeichert. SVG ist also ein XML-Derivat. Es wird ebenso wie HTML und CSS vom W3-Konsortium als offen dokumentierter Standard spezifiziert und weiterentwickelt. Führende Vektorgrafikprogramme wie CorelDRAW oder Adobe Illustrator sind längst in der Lage, Daten als SVG-Grafik zu exportieren.

Auch wenn SVG-Unterstützung der Browser noch lausig ist, so ist doch zu erwarten, dass sich die Situation noch verbessern wird, was sicherlich zu einer weiteren Verbreitung von SVG beitragen wird. In HTML können Sie eine SVG-Datei wie auch Flash und Java über das object-Element einbinden. Ein Beispiel:

```
<object type="image/svg+xml" data="beispiel.svg"
        style="width:350px; height:176px">
   SVG-Viewer erforderlich! Auf
   http://www.w3.org/Graphics/SVG/SVG-Implementations
   erhalten Sie Informationen über SVG-Implementierungen.
</object>
```

Die SVG-Quelle wird im data-Attribut als URI referenziert. Das type-Attribut sollte unbedingt notiert werden und den Mime-Type `image/svg+xml` als Wert zugewiesen bekommen.

Wenn Sie XHTML statt HTML verwenden, sind Sie bei SVG übrigens nicht auf die Referenzierung angewiesen. In diesem Fall können Sie über die Modularisierungsphilosophie von XML den SVG-Code direkt in den Quelltext integrieren. Dazu wird das einleitende `<html>`-Tag wie folgt erweitert:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg">
```

Es wird also zusätzlich zum Namensraum von XHTML derjenige von SVG deklariert. Der XML-Code des SVG-Dokuments kann nun an der gewünschten Stelle im Dokument wie im folgenden Minibeispiel integriert werden:

```
<svg:svg height="100px" width="100px">
    <svg:rect x="0" y="0" width="100" height="100" fill="red" />
</svg:svg>
```

Bei jedem SVG-Tag muss dazu das Präfix `svg:` notiert werden, da HTML als Default-Namensraum definiert wurde. In der Browser-Praxis ist dies jedoch noch weitgehend Zukunftsmusik, da für eine korrekte Interpretation der XML-Parser des Browsers benötigt wird.

Link zum SVG-Format:

W3C-Recommendation zu SVG:
http://www.w3.org/TR/SVG/
SVG-Tutorial:
http://svg.tutorial.aptico.de/
Mozilla SVG Project:
http://www.mozilla.org/projects/svg/
Adobe SVG Zone:
http://www.adobe.com/svg/

4.9.8 Eingebettete Objekte aller Art

In den Abschnitten über Flash, Java und SVG haben wir gesehen, dass die Einbindung der Ressourcen stets über das universelle object-Element in HTML erfolgt. Dieses Element ist so universell, dass sich damit jede beliebige Ressource einbinden lässt. So ist es auch schon absehbar, dass das img-Element in neueren XHTML-Versionen verschwindet. Grafiken werden dann ebenfalls über das object-Element referenziert. Das sieht beispielsweise so aus:

```
<object type="image/png" data="bild.png"
        style="width:354px; height:314px"></object>
```

Auch eine nackte Textdatei lässt sich über dieses Element einbinden. Ein Beispiel:

```
<object type="text/plain" data="e-mail-thomas-003.txt"
        style="width:80%; height:300px;
               overflow:scroll"></object>
```

Wichtig sind im Zweifelsfall stets die Attribute `data=` zum Referenzieren der Quelle und `type=` zur Angabe des Mime-Type. Mit dem style-Attribut lässt sich der Anzeigebereich anpassen. Im Beispiel der referenzierten Textdatei haben wir eine Anzeigegröße vorgegeben und mit `overflow:scroll` sichergestellt, dass der Text, sofern er mehr Platz einnimmt, vom Anwender gescrollt werden kann.

Typische, über das object-Element eingebundene Objekte sind auch Sounds und Video. In der Regel ist dazu jedoch eine Angabe zum classid-Attribut erforderlich, die ein bestimmtes Plug-In oder Abspielprogramm im Browser erzwingt, sowie die Angabe von `<param>`-Tags für Initialwerte. Für Standard-Software gibt es classid-Codes, die dem classid-Attribut zugewiesen werden können. Das folgende Beispiel bindet den Windows Media Player ein und übergibt ihm eine MP3-Datei zum Abspielen:

```
<object id="MediaPlayer" classid="clsid:6BF52A52-394A-11D3-B153-00C04F79FAA6"
type="application/x-oleobject" style="width:300px; height:200px">
  <param name="src" value="beispiel.mp3">
  <param name="autoplay" value="false">
  <param name="controller" value="true">
</object>
```

Das Beispiel funktioniert in dieser Form jedoch nur im MS Internet Explorer, da die Einbettung über ein ActiveX-Control realisiert wird, eine bei anderen Browsern verpönte, weil für Sicherheitslücken anfällige Technologie.

Zahlreiche andere Formate haben ihre eigene Syntax zum Einbetten. Dem Wildwuchs ist hier leider noch Tür und Tor geöffnet, da vernünftige Standards fehlen. Anbieter spezieller Techniken wie RealMedia (Audio- und Video-Streaming) und andere informieren auf ihren Seiten darüber, wie Sie solche Formate in HTML einbetten. In einigen Fällen sind zusätzlich Einstellungen oder Erweiterungen beim Webserver erforderlich.

4.10 HTML-Formulare

Webseiten mit der Funktion »Webpräsenz« sind nur eine mögliche Variante. Vieles, was das Web attraktiv macht, fällt eher unter den Begriff »Webanwendung«. Der »Seitenbesucher« wird zum »User«, der Suchen startet, Beiträge in Foren schreibt, persönliche digitale Fotoalben verwaltet, Kleinanzeigen aufgibt usw. Auch dabei basieren die Webseiten auf HTML.

Die nötige Interaktion mit dem Anwender beginnt in der Regel mit einem Formular. HTML stellt eine Reihe von Standard-Formularelementen zur Verfügung, die sich an den Dialogelementen moderner grafischer Benutzeroberflächen orientieren. So gibt es Eingabefelder, Radiobuttons, Checkboxen, Auswahllisten, Schaltflächen usw.

Mit dem Formular allein ist es jedoch nicht getan. Daten, die ein Anwender in einem Formular bestimmt und absendet, müssen irgendwie verarbeitet werden. Das Instrument dafür sind serverseitige Scripts, beispielsweise PHP-Scripts. Richtige Anwendungen für Formulare werden wir deshalb auch erst im Buchteil zu PHP und MySQL finden. Im vorliegenden Abschnitt konzentrieren wir uns auf die Strukturierungs- und Gestaltungsmöglichkeiten von Formularen auf Webseiten.

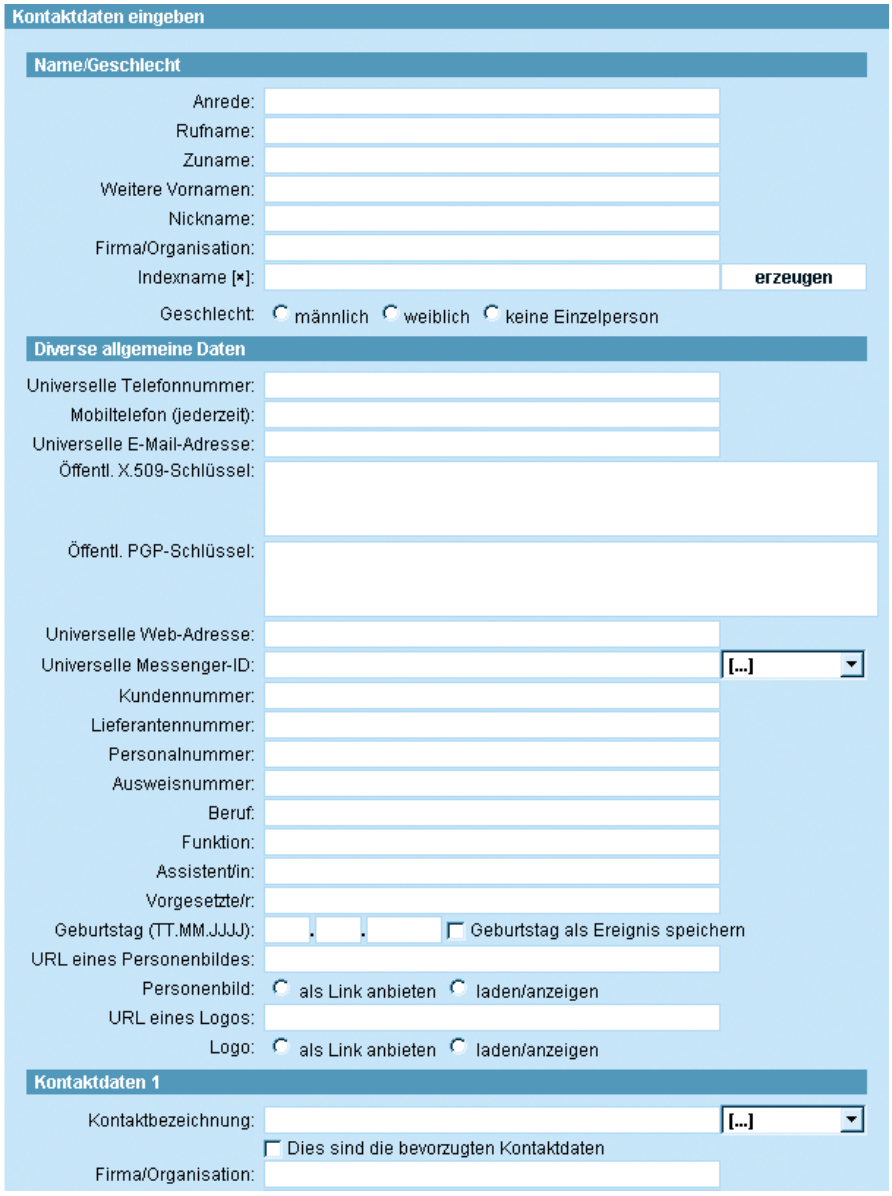


Abbildung 4.70: Komplexes und gestyltes HTML-Formular (Ausschnitt)

4.10.1 Formularbereiche

Formulare werden in HTML durch `<form...>...</form>` markiert. Im einleitenden `<form>`-Tag werden wichtige Informationen notiert, z.B. was passieren soll, wenn der Anwender das Formular absendet. Zwischen `<form>` und `</form>` können Formularelemente notiert werden, aber auch Text und andere Elemente, die dazu beitragen, das Formular zu strukturieren.

Betrachten wir zunächst das einleitende `<form>`-Tag in einem Beispiel:

```
<form action="/index.php" method="post">
   <!-- Formularinhalt -->
</form>
```

In diesem Beispiel wird bestimmt, dass die im Formular eingegebenen oder ausgewählten Daten beim Absenden an ein PHP-Script namens ***index.php***, das sich im Wurzelverzeichnis des Webservers befindet, übertragen werden. Zuständig für die Verarbeitungsadresse ist das Attribut `action=`, das als Wert einen URI erwartet. Es kann sich also um eine beliebige geeignete Internetadresse oder um ein lokal referenziertes Script handeln.

Erlaubt sind auch Angaben des Typs `mailto:jemand@irgendwo.xy`. In diesem Fall sollte im `<form>`-Tag unbedingt zusätzlich `enctype="text/plain"` notiert werden. Der Browser versucht dann, die Formulardaten über eine lokale Mail-Schnittstelle an die angegebene Mail-Adresse zu senden. Sicherlich ist diese Form der Formularverarbeitung die einfachste. Als Seitenanbieter gibt man einfach seine eigene Mail-Adresse in dieses Schema ein und schon bekommt man alle Formulardaten, die Anwender absenden, als E-Mail zugeschickt. Allerdings ist von dieser Praxis dringend abzuraten. Der Mail-Versand funktioniert nämlich bei vielen Anwendern nicht, da Browser normalerweise keine eigene Schnittstelle für den Mailversand besitzen und die erfolgreiche Kommunikation der Formulardaten vom Browser an ein Mail-Programm von vielen vagen Systemfaktoren abhängt. Bei betroffenen Anwendern wird Ihr Formular also für Ärger sorgen, da sie ein Formular, für dessen Ausfüllen sie möglicherweise einige Zeit investiert haben, am Ende nicht absenden können. Als Wert für das action-Attribut sollte also bei allen professionellen Angeboten ein Script angegeben werden, das die eingegebenen Daten ordentlich verarbeitet.

Ebenso wichtig wie das action-Attribut ist das Attribut `method=`. Als Werte sind hier entweder `get` oder `post` möglich. Damit wird die Art bestimmt, wie dem datenverarbeitenden Script die Daten übergeben werden. Auf die Details werden wir erst später im Buch eingehen. Doch auf eine leicht erkennbare Unterscheidung zwischen den Übertragungsmethoden `GET` und `POST` sei hier schon hingewiesen: Bei der GET-Methode werden die Formulardaten über den URI übertragen. In der Adresszeile des Browsers ist das daran erkennbar, dass dort nach Absenden des Formulars, also beim Aufruf des bei `action=` angegebenen Scripts, alle Namen und Werte der Formularelemente in einer Zeichenkette erscheinen. Nach Eingabe eines Suchbegriffs in Google erscheint im Browser beispielsweise:

http://www.google.de/search?hl=de&q=Siebenb%C3%BCrgen&btnG=Google-Suche&meta=

Ein Quelltextauszug aus der Google-Seite mit dem Formular zur Suche lautet:

```
<form name="gs" method="GET" action="/search">
```

Das datenverarbeitende Script hat also den Dateinamen *search* und liegt im Wurzelverzeichnis des Webservers. Die Übertragungsmethode GET sorgt dafür, dass die eingegebenen Daten des Suchformulars im URI stehen. Bei der POST-Methode ist dies nicht der Fall. Das Script erhält die Daten in diesem Fall über den Standard-Input-Kanal des Server-Rechners.

Welche Übertragungsmethode Sie in einem Formular angeben müssen, hängt davon ab, welche der Übertragungsmethoden vom verarbeitenden Script ausgewertet werden. Wenn Sie das Script selber programmieren, können Sie die Methode selbst bestimmen. Handelt es sich um ein fremdprogrammiertes Script, muss die Schnittstelle von Daten, die diesem Script übergeben werden können, vom Programmierer oder Anbieter ohnehin dokumentiert sein. Dazu gehört auch die vom Script erwartete Übertragungsmethode. Bei mailto-Formularen muss die Methode `POST` gewählt werden.

Das einleitende `<form>`-Tag kann noch weitere Attribute enthalten, z. B. das name-Attribut, wie im obigen Beispielformular von Google erkennbar. Dieses ist vor allem in Zusammenhang mit JavaScript von Bedeutung. Im Buchteil über JavaScript werden wir darauf noch näher eingehen. Weitere Attribute lernen Sie in ***Abschnitt 4.10.3*** über die Optimierung von Formularen kennen.

4.10.2 Formularelemente

Die nachfolgend vorgestellten Formularelemente können Sie zwischen `<form>` und `</form>` notieren.

Eingabefelder, Radiobuttons, Checkboxen und Schaltflächen

Eine ganze Reihe von Formularelementen in HTML werden alle nach dem gleichen Schema ausgezeichnet:

```
<input type="..." name="..." value="...">
```

<input type=	Bedeutung	Ansicht
"text"	Einzeiliges Eingabefeld	Stefan Münz
"password"	Eingabefeld für Passwörter	**********
"radio"	Radiobutton (ja/nein-Schalter). Von mehreren Radiobuttons mit gleichem name-Attributwert kann immer nur einer ausgewählt werden.	

Tabelle 4.20: Formularelemente mit dem input-Element

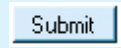
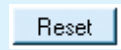
<input type=	Bedeutung	Ansicht
"checkbox"	Checkbox (ja/nein-Schalter)	
"button"	Schaltfläche ohne vordefinierte Funktion	
"submit"	Schaltfläche zum Absenden der Formulardaten	
"reset"	Schaltfläche zum Zurücksetzen des Formulars in den Anfangszustand. Eingaben werden gelöscht.	
"image"	Alternative zu "submit". Ein zusätzliches src-Attribut im <input>-Tag erlaubt das Referenzieren eines Grafik-URI. Die Grafik ist anklickbar und sendet das Formular ab.	 (Beispielgrafik)
"file"	Eingabefeld mit Durchsuchen-Schaltfläche zum Auswählen einer beim Anwender lokal gespeicherten Datei. Die Datei wird zusammen mit den Formulardaten abgesendet (»File-Upload«).	
"hidden"	Versteckte Information; Element ist nicht sichtbar.	

Tabelle 4.20: Formularelemente mit dem input-Element (Fortsetzung)

Einzeilige Eingabefelder eignen sich für alle Arten von Einzeldaten wie Namen, Mail-Adressen, Telefonnummern, Suchausdrücke usw. Passwortfelder sollten nur für entsprechende Zwecke genutzt werden. Dabei ist es wichtig zu wissen, dass lediglich die Darstellung am Bildschirm verschleiert wird. Die eingegebenen Daten werden dadurch noch nicht verschlüsselt.

Radiobuttons sind nur als Gruppe sinnvoll. Dabei erhalten alle zur Gruppe gehörigen Radiobuttons bei `name=` den gleichen Wert. Der Browser reagiert so, dass immer nur einer der Radiobuttons aktivierbar ist. Eine sinnvolle Anwendung sind beispielsweise zwei Radiobuttons zur Auswahl des Geschlechts männlich oder weiblich.

Checkboxen eignen sich vor allem zum Ein-/Ausschalten von Optionen wie etwa »Groß-/Kleinschreibung unterscheiden«.

Sowohl bei Radiobuttons als auch bei Checkboxen können Sie im <input>-Tag das Standalone-Attribut `checked` (in XHTML: `checked="checked"`) notieren. Damit erzwingen Sie, dass das entsprechende Element beim Aufrufen oder Zurücksetzen des Formulars vorausgewählt ist. Bei zusammengehörigen Radiobuttons kann natürlich nur eines der Elemente vorausgewählt werden.

Bei den Schaltflächen unterscheidet HTML grundsätzlich zwischen funktionslosen Schaltflächen (`type="button"`) und den beiden vordefinierten Schaltflächen zum Absenden der Formulareingaben (»OK«-Funktionalität bzw. Submit-Button – `type="submit"`) und zum

Zurücksetzen der Formulareingaben (»Abbrechen«-Funktionalität bzw. Reset-Button – `type="reset"`). Funktionslose Schaltflächen sind vor allem für clientseitige Interaktionsverarbeitung mit JavaScript gedacht. Die beiden anderen Typen sind dagegen Nachbildungen der Standardschaltflächen von Dialogen, wie man sie auch bei modernen grafischen Benutzeroberflächen kennt. Die reset-Schaltfläche ist jedoch, was Usability betrifft, mit Vorsicht zu genießen, vor allem bei umfangreicheren Formularen. Ein unachtsamer Anwenderklick auf eine solche Schaltfläche und alle zuvor getätigten Formulareingaben sind unwiderruflich weg. Die meisten Formulare bieten daher nur die submit-Schaltfläche an.

Die Beschriftung der Schaltfläche wird in allen Fällen im value-Attribut festgelegt.

Die Auszeichnung von Schaltflächen über `<input ...>` ist nur die ältere von zwei Varianten. Für die neuere Variante wurde mit HTML 4.0 das button-Element eingeführt. Die folgenden beiden Beispiele bewirken im Browser letztlich das Gleiche:

```
<input type="submit" value="Absenden">
<button type="submit" value="Absenden"></button>
```

Der Unterschied besteht vor allem darin, dass das input-Element ein Standalone-Element ohne End-Tag ist (in XHTML muss es daher in der Form `<input .../>` notiert werden). Das button-Element erlaubt dagegen Elementinhalt, und zwar sowohl Text als auch Block- und Inline-Elemente. Auf diese Weise können Sie Buttonbeschriftungen auffällig gestalten, beispielsweise auch mit Grafiken.

Die Attribute `name=` und `value=` bestimmen die Daten, die beim Absenden des Formulars übertragen werden. Das name-Attribut bezeichnet den Namen eines Parameters und das value-Attribut dessen Wert. Während das name-Attribut bei allen input-Typen außer bei `submit` und `reset` obligatorisch ist, ist das value-Attribut nicht zwingend erforderlich, da der Wert zumindest bei Eingabefeldern, Radiobuttons, Checkboxen usw. durch den Anwender bestimmt wird. Das value-Attribut eignet sich bei Eingabefeldern jedoch dazu, einen Default-Wert vorzugeben. Beispiel:

```
URL-Adresse: <input type="text" name="url" value="http:// ">
```

In diesem Fall wird das Eingabefeld mit dem Namen ***url*** mit der Zeichenfolge `http://` vorbelegt.

Bei der Wertzuweisung an das name-Attribut gelten gewisse Regeln. Der Parametername muss mit einem Buchstaben A–Z oder a–z beginnen. Als weitere Zeichen sind ebenfalls A–Z und a–z erlaubt sowie die Ziffern 0–9 und die Zeichen für Unterstrich (_), den Bindestrich (-), Doppelpunkt (:) und Punkt (.). In Hinblick auf Programmiersprachen wie JavaScript oder PHP, die auf die Parameter über diese Namen zugreifen, sollten Sie sich jedoch unbedingt auf Buchstaben, Ziffern und Unterstrich beschränken. Groß- und Kleinschreibung werden bei den genannten Programmiersprachen ebenfalls unterschieden.

Auf die optische Gestaltung von Elementen wie Eingabefeldern, wozu auch die Bestimmung von angezeigter Breite gehört, gehen wir in ***Abschnitt 4.10.4*** ein. Die Breite eines Eingabefelds hat jedoch nichts damit zu tun, wie viele Zeichen in dem Feld eingegeben

werden können. Um die Anzahl der möglichen Zeichen in einem Feld zu begrenzen, benötigen Sie das maxlength-Attribut. Ein Beispiel:

```
Vorname: <input type="text" name="given_name" maxlength="30">
```

In diesem Beispiel wird durch die Angabe `maxlength="30"` festgelegt, dass der Browser von Eingaben in diesem Feld maximal die 30 ersten Zeichen übertragen soll. Solche Begrenzungsangaben sind wichtig, wenn das verarbeitende Script die eingegebenen Daten beispielsweise in eine Datenbank schreibt, in der die Feldlängen entsprechend begrenzt sind. Allerdings ist auf die Zuverlässigkeit der Browser bei der Interpretation des maxlength-Attributs kein Verlass. Stattdessen empfiehlt es sich in jedem Fall, Feldeingaben vor dem Schreiben in eine Datenbank auch auf ihre Länge hin zu prüfen und gegebenenfalls abzuschneiden.

Speziellere Formen des input-Elements sind `type="image"`, `type="file"` und `type="hidden"`.

Die Angabe `type="image"` ersetzt `type="submit"` und ist funktionsgleich. Anstelle einer Standardschaltfläche kann eine frei wählbare Grafik als Absende-Button definiert werden, wie im folgenden Beispiel:

```
<input type="image" src="ok.png" alt="Absenden">
```

Die gewünschte Grafik wird beim src-Attribut als URI zugewiesen. Außerdem sollten Sie in diesem Fall auch das alt-Attribut notieren, das den Alternativtext enthält, falls die Grafik nicht angezeigt werden kann.

Die gleiche Funktionalität wie durch `type="image"` erreichen Sie übrigens über das button-Element wie folgt:

```
<button type="submit">
   <img src="ok.png" alt="Absenden">
</button>
```

Versteckte Formularelemente (`type="hidden"`) sind nur in Zusammenhang mit Scripting von Bedeutung. Wir werden deshalb erst bei den Scriptsprachen näher darauf eingehen.

Nicht ganz unproblematisch ist das input-Element mit der Funktionalität ***Datei-Upload*** (`type="file"`). Das serverseitige Script, das auf diese Weise Dateien vom Anwender übertragen bekommt, muss auf jeden Fall eine Reihe von Sicherheitschecks durchführen, um den Server nicht zu gefährden. Denn Dateien können Viren transportieren, sehr groß sein usw. Die erlaubte Maximalgröße ist zwar in HTML wie auch bei der Zeichenlänge von Eingabefeldern durch das maxlength-Attribut begrenzbar. Doch wie schon erwähnt ist kein Verlass darauf, dass alle Browser das Attribut konsequent beachten.

Wenn Sie in ein Formular ein File-Upload-Feld integrieren, sollte das einleitende `<form>`-Tag folgende Angaben erhalten:

```
<form action="..." method="post" enctype="multipart/form-data">
```

Die enctype-Angabe ist wichtig, damit serverseitig erkannt werden kann, dass es sich um mehrteilige Daten handelt. Der Wert `multipart/form-data` ist eine Mime-Type-Angabe für solche Zwecke. Als Übertragungsmethode sollte im Fall von File-Upload stets die POST-Methode gewählt werden.

Im `<input>`-Tag ist es außerdem möglich, mithilfe eines accept-Attributs die Dateiauswahl zu begrenzen, z. B.:

```
<input name="Datei" type="file" maxlength="100000" accept="image/*">
```

In diesem Beispiel wird dem Anwender erlaubt, eine lokal gespeicherte Grafikdatei (`image/`) beliebigen Typs (`*`) mit einer Maximalgröße von 100.000 Byte zum Hochladen auszuwählen.

Mehrzeilige Eingabebereiche

Ein mehrzeiliger Eingabebereich eignet sich für Freitext aller Art, etwa für Notizen, Bemerkungen, oder bei einem Forum oder Gästebuch für den Beitragstext. In HTML gibt es dafür ein eigenes Element:

```
<textarea name="bemerkung" cols="50" rows="10">Platz für Ihre Bemerkungen!</
textarea>
```

Das textarea-Element besteht aus Anfangs- und End-Tag. Dazwischen kann Text notiert werden. Dieser wird als Vorbelegungstext interpretiert.

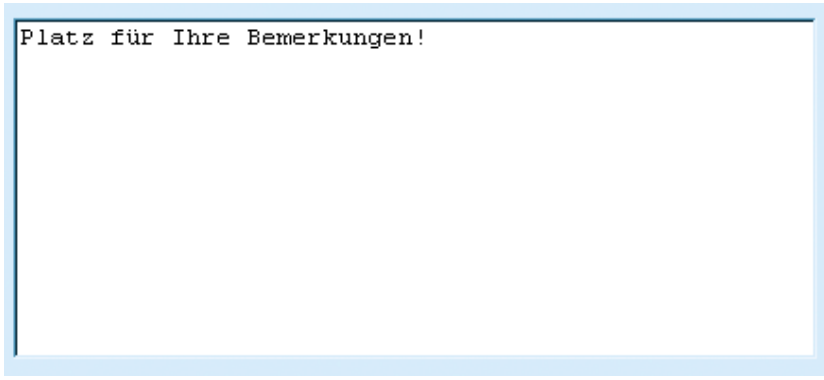


Abbildung 4.71: Mehrzeiliger Eingabebereich (textarea-Element)

Falls der Anwender mehr Text eingibt als die Größe des Bereichs zulässt, fügt der Browser automatisch eine Scrollleiste ein. Die Größe des Bereichs können Sie durch die Attribute `rows=` (Anzahl Zeilen) und `cols=` (Anzahl Spalten) festlegen. Diese Angaben sind allerdings nur für textmodusbasierte Browser wirklich von Bedeutung. Für grafische Browser empfiehlt es sich, die Größe des Eingabefelds zusätzlich mit entsprechenden CSS-Eigenschaften (`width` und `height`) festzulegen (in Zusammenhang mit der optischen Gestaltung von Formularen gehen wir noch genauer darauf ein).

Eingegebener Text wird übrigens automatisch umgebrochen. Durch Betätigen der Enter-Taste kann der Anwender Zeilenumbrüche im Text erzwingen.

Auswahllisten und Drop-down-Listen

HTML bietet ein Element für Listenauswahl in Formularen an: das select-Element. Ob es sich um eine längere Liste mit mehreren sichtbaren Einträgen handelt oder um eine Drop-down-Liste, die der Anwender aufklappen kann, bestimmen Sie durch ein Attribut.

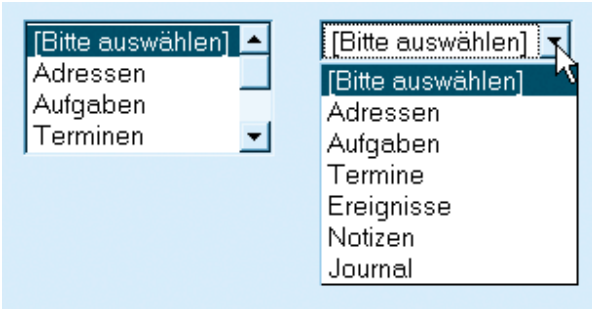


Abbildung 4.72: Auswahlliste als Liste mit mehreren Einträgen oder als Drop-down-Liste

Ein Beispiel:

```
<select name="pim_modul" size="4">
   <option value="null" selected>[Bitte auswählen]</option>
   <option value="adressen">Adressen</option>
   <option value="aufgaben">Aufgaben</option>
   <option value="termine">Termine</option>
   <option value="events">Ereignisse</option>
   <option value="notizen">Notizen</option>
   <option value="journal">Journal</option>
</select>
```

Die Liste wird in `<select>...</select>` eingeschlossen. Die einzelnen Listeneinträge werden durch `<option>...</option>` markiert. Der Text zwischen `<option>` und `</option>` ist der Text, den der Anwender als Listeneintrag angeboten bekommt. Der Wert des Attributs `value=` ist dagegen der Wert, der beim Absenden der Formulardaten als der ausgewählte Listenwert an das verarbeitende Script übertragen wird.

Das einleitende `<select>`-Element erhält bei `name=` den Namen des Parameters, unter dem die Listenauswahl an das verarbeitende Script übergeben wird. Das Attribut `size=` bestimmt das Aussehen der Liste. Weisen Sie als Wert `1` zu, ist das Ergebnis eine Drop-down-Liste. Weisen Sie einen höheren Wert zu, ist das Ergebnis eine Liste mit so vielen sichtbaren Einträgen wie im Wert angegeben. Enthält die Liste mehr Einträge als bei `size=` angegeben, kann der Browser dem Anwender automatisch eine Scrollleiste anzeigen.

Das obige Beispiel zeigt noch eine andere Möglichkeit: Auswahllisten können eine Default-Auswahl enthalten. Eines der notierten option-Elemente können Sie durch das

Standalone-Attribut `selected` als den vorausgewählten Listeneintrag bestimmen. Wenn Sie mit XHTML arbeiten, notieren Sie `selected="selected"`.

In vielen Fällen ist es auch sinnvoll, einen Eintrag anzubieten, der keinen Nutzwert hat und stattdessen zur Auswahl auffordert, so wie im Beispiel das erste notierte option-Element.

Für längere Listen mit logisch gruppierbaren Einträgen bietet HTML die Möglichkeit an, Listeneinträge zu Gruppen zusammenzufassen. Ein Beispiel:

```
<select name="country" size="7">
 <optgroup label="Europa">
  <option value="de">Deutschland</option>
  <option value="au">Österreich</option>
  <option value="it">Italien</option>
  <option value="es">Spanien</option>
 </optgroup>
 <optgroup label="Afrika">
  <option value="eg">Ägypten</option>
  <option value="ke">Kenia</option>
  <option value="ug">Uganda</option>
 </optgroup>
</select>
```

Die Gruppierung wird durch Einfügen von optgroup-Elementen erreicht. Zwischen `<optgroup>` und `</optgroup>` werden die jeweils zur Gruppe gehörigen Listeneinträge wie gewohnt mithilfe von option-Elementen notiert. Das einleitende <optgroup>-Tag erhält über das Attribut `label=` den Wert, der als Gruppenüberschrift angezeigt werden soll.



Abbildung 4.73: Auswahlliste mit gruppierten Einträgen

Moderne Webbrowser setzen gruppierte Listen in der Regel so um wie abgebildet.

Bei manchen Listen kann es auch sinnvoll sein, dem Anwender die Auswahl mehrerer Einträge zu erlauben. HTML-seitig ist das kein Problem. Im einleitenden `<select>`-Tag wird einfach das Standalone-Attribut `multiple` (in XHTML: `multiple="multiple"`) eingefügt. Problematischer ist es, dass der Anwender nicht weiß, dass er mehrere Einträge auswählen kann, und meistens auch gar nicht, wie. Bei expliziter Mehrfachauswahl ist es also zweckmäßig, im umgebenden Text des Elements auf die Möglichkeit der Mehrfachauswahl hinzuweisen. Da viele Anwender auch nicht wissen, dass sie mehrere Einträge

durch Anklicken bei gedrückter Strg-Taste auswählen können, kann auch dazu ein entsprechender Hinweis im Text nicht schaden.

4.10.3 Formulardesign und Formularoptimierung

Seiten mit Formularen werden von Anwendern meist länger angezeigt und intensiver wahrgenommen als andere Seiten. Allein schon deshalb ist es Anbieterpflicht, Formulare so zu gestalten, dass der Anwender sich ohne Probleme darin zurechtfindet.

Folgende Punkte sind wichtig:

- Formularbereiche sollten optisch erkennbar sein.
- Feldbeschriftungen und Formularelemente sollten einheitlich zueinander ausgerichtet sein. Der Zusammenhang zwischen einer Feldbeschriftung und dem zugehörigen Feld muss eindeutig erkennbar sein.
- Eingabefelder sollten ordentlich positioniert sein und so weit sinnvoll eine einheitliche Länge haben, damit das Formular nicht zu unruhig wirkt.
- In umfangreicheren Formularen sollten logisch/thematische zusammengehörige Elemente gruppiert werden. Die Gruppen sollten Gruppenüberschriften erhalten.
- Wenn Felder Standardwerte haben, kann dem Anwender Arbeit erspart werden, indem die Felder mit den Standardwerten vorbelegt sind oder Standardwerte voreingestellt oder vorausgewählt sind.
- Felder, deren Zweck sich nicht selbstverständlich erschließt, oder Felder, die als Wert ein bestimmtes Format oder einen bestimmten Wertebereich erhalten, sollten erläutert werden.
- Felder, die für das verarbeitende Script Pflichtfelder sind, sollten dem Anwender als solche kenntlich gemacht werden.

Anhand eines einfachen Beispielformulars sollen diese Punkte erläutert werden. Der vollständige Quelltext der HTML-Datei des Beispiels lautet:

Listing 4.10: HTML-Datei mit strukturiertem Formular

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<style type="text/css">
input, fieldset, legend {
  font-size:13px; font-family:Arial, sans-serif;
}
</style>
<title>Formular</title>
</head>
```

```
<body>
<h1>Login</h1>
<form action="login.php" method="post"
      style="background-color:#DDDDDD; padding:4px">
<fieldset><legend>Zugangsdaten</legend>
<table><tr>
<td style="width:150px; font-weight:bold; text-align:right">
  <label for="user">* Benutzername:</label>
</td><td>
   <input type="text" maxlength="40" name="user"
          id="user" style="width:200px">
   (max. 40 Zeichen)
</td>
</tr><tr>
<td style="font-weight:bold; text-align:right">
   <label for="pwd">* Passwort:</label>
</td><td>
   <input type="password" maxlength="10" name="pwd"
           id="pwd" style="width:200px">
   (max. 10 Zeichen)
</td></tr></table>
</fieldset><fieldset><legend>Optionen</legend>
<table><tr>
<td style="width:150px; font-weight:bold;
           text-align:right; vertical-align:top">
   <label for="cookies">Cookies:</label>
</td><td>
   <input type="radio" name="cookies" id="cookies"
          value="yes" checked>ja<br>
   <input type="radio" name="cookies" value="no">nein
</td>
</tr><tr>
<td style="font-weight:bold; text-align:right;
           vertical-align:top">
   <label for="public">Benutzername:</label>
</td><td>
   <input type="radio" name="public" id="public"
          value="yes">anderen Usern anzeigen<br>
   <input type="radio" name="public"
          value="no" checked>anderen Usern nicht anzeigen
</td></tr></table>
</fieldset>
<p><button type="submit" accesskey="s" style="font-weight:bold">
   Login <span style="text-decoration:underline">s</span>enden
</button></p>
<fieldset><legend>Hinweis</legend>
<p>Mit <span style="font-weight:bold">*</span>
gekennzeichnete  Felder müssen ausgefüllt werden!</p>
</fieldset>
</form>
</body>
</html>
```

In einem HTML-4.0- und CSS-fähigen Browser sieht die Beispieldatei etwa folgendermaßen aus:

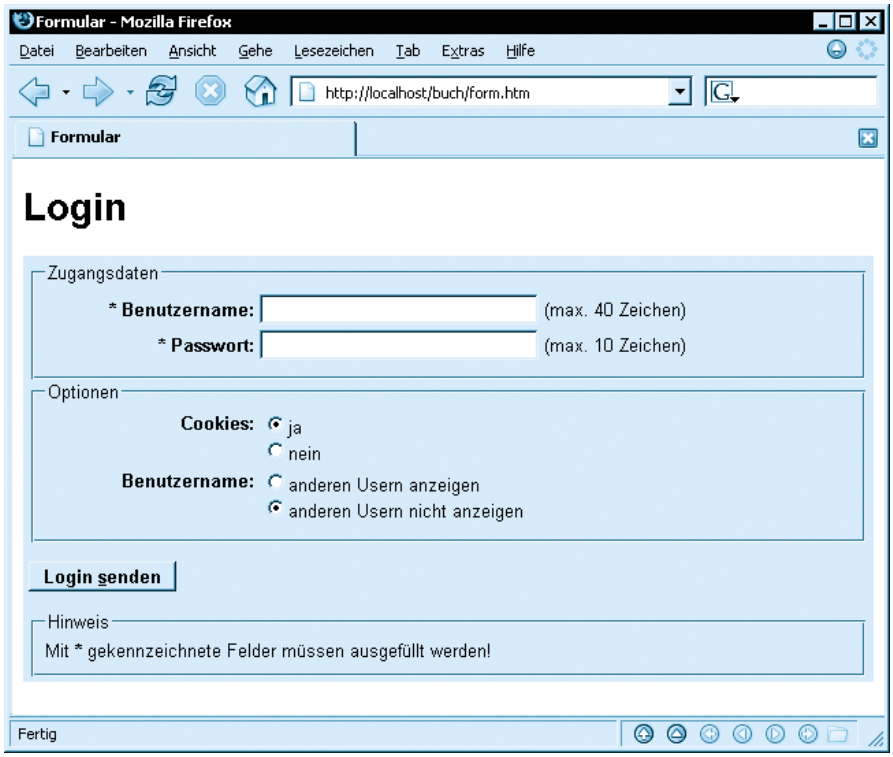


Abbildung 4.74: Strukturiertes Formular

Das Formular selbst wird im Beispiel optisch gekennzeichnet durch die hellgraue Hintergrundeinfärbung (`<form ... style="background-color:#DDDDDD; ...">`). Für die Gruppierung der einzelnen Bereiche wird ein eigens dafür gedachtes HTML-Element benutzt: das fieldset-Element. Insgesamt werden im Beispiel drei solcher Fieldsets definiert, erkennbar an den Rändern, die sie erzeugen. Mit `<fieldset>...</fieldset>` wird der gewünschte Bereich des Formulars eingeschlossen. Hinter jedem einleitenden <fieldset>-Tag folgt jedoch noch ein Text, der mit `<legend>...</legend>` ausgezeichnet ist. Dieser Text wird als Gruppenüberschrift des fieldset-Bereichs angezeigt und vom Browser bei der Anzeige links oben in den fieldset-Rahmen positioniert.

Einen Nachteil haben die fieldset-Bereiche allerdings: Sie verhindern, dass eine durchgängige rahmenlose Tabelle notiert werden kann, um Beschriftungen und Formularelemente zueinander auszurichten. Deshalb übernehmen im Beispiel zwei solcher Tabellen diese Aufgabe. Durch geeignete CSS-Angaben werden die Zellen mit den Feldbeschriftungen rechtsbündig und wo nötig am oberen Rand ausgerichtet. Die einzeiligen Eingabefelder erhalten durch die CSS-Eigenschaft `width` die gleiche Länge, auch wenn im Passwortfeld nur zehn Zeichen eingegeben werden können. Durch die einheitliche Länge wirkt das For-

mular jedoch ruhiger. Stattdessen wird der Anwender hinter den Eingabefeldern durch Kurzhinweise darüber aufgeklärt, wie lange die Feldeingaben maximal sein dürfen.

Durch das Markup der rahmenlosen Tabelle stehen Feldbeschriftung und Formularfeld nicht unmittelbar beieinander. Bei nicht visueller, also z. B. akustischer Ausgabe der Webseite könnte dies zu Problemen bei der Zuordnung von Beschriftung und Feld führen. HTML bietet hierfür eine Lösung an, die wir im Beispielformular ebenfalls verwendet haben. Der Beschriftungstext wird jeweils mit `<label for="...">...</label>` ausgezeichnet. Das for-Attribut erhält als Wert einen id-Namen. Im zugehörigen Formularelement wird dieser Name in einem id-Attribut notiert. Dadurch wird ein logischer Zusammenhang zwischen Beschriftung und Formularelement hergestellt.

Für die Felder ***Benutzername*** und ***Passwort*** gibt es keine sinnvollen Vorbelegungswerte. Die beiden Radiobutton-Gruppen der Optionen ***Cookies*** und ***Benutzername*** dagegen erhalten mithilfe des Attributs `checked` je eine Default-Auswahl. Auf diese Weise ist sichergestellt, dass in jedem Fall eine gültige Option beim Absenden des Formulars übertragen wird.

Der Submit-Button zum Absenden des Formulars wird im Beispiel mithilfe des button-Elements realisiert. Grund ist, dass die Beschriftung einen als unterstrichen formatierten Buchstaben enthält (das »s« bei ***Login senden***). Bei einer Wertzuweisung des Beschriftungstextes an das value-Attribut eines `<input type="submit">`-Konstrukts wäre das nicht möglich. Unterstrichen wird das »s« deshalb, weil im einleitenden `<button>`-Tag das Attribut `accesskey="s"` notiert ist. Das accesskey-Attribut funktioniert in Formularelementen genauso wie bei Hyperlinks. Durch einen Hotkey wie Alt+S (browserabhängig, siehe auch Tabelle 4.18 ***auf Seite 193***) kann der Anwender das Formular über die Tastatur absenden.

Ebenso wie das accesskey-Attribut kann in Formularelementen analog zu Hyperlinks ein tabindex-Attribut notiert werden. Und genauso wie bei den Links kann durch die als Wert zugewiesene Zahl die Reihenfolge beim Anspringen der Felder über die ⇧-Taste festgelegt werden.

4.10.4 Optische Verfeinerung von Formularen mit CSS

CSS ist, wie im Beispiel des vorherigen Abschnitts gezeigt, durchaus auch auf Formularelemente anwendbar. Bei Eingabefeldern, aber auch bei Auswahllisten und Schaltflächen lässt sich mit der CSS-Eigenschaft `width` eine gewünschte Breite festlegen. Bei textarea-Elementen für mehrzeilige Eingaben kann die Höhe wie erwartet durch `height` bestimmt werden. Möglich ist jedoch noch viel mehr: angefangen von eher ästhetischer Beeinflussung wie rahmenlose Eingabefelder, farbige Feldhintergründe oder Schriftformatierung bei Eingabefeldern bis hin zu sinnigen Möglichkeiten wie der rechtsbündigen Ausrichtung von Eingabefeldern für numerische Werte bleiben kaum Wünsche offen.

Da die bekannten Browser zum Rendern von Formularelementen teilweise auf Betriebssystemressourcen zurückgreifen, sind den Gestaltungsmöglichkeiten auf dieser Ebene jedoch auch Grenzen gesetzt. So kann es passieren, dass die Unterdrückung des Rahmens

bei Auswahllisten nicht funktioniert oder eine Checkbox einfach keine andere Hintergrundfarbe annehmen will.

Ein Beispiel soll einige der Möglichkeiten demonstrieren:

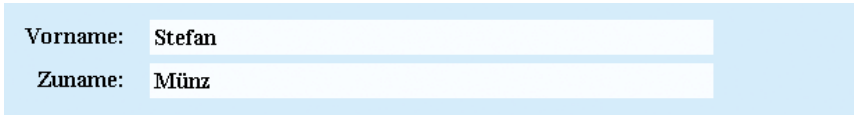


Abbildung 4.75: Formular und Eingabefelder mit CSS gestaltet

Im abgebildeten Beispiel wurde im zentralen style-Bereich definiert:

```
form {
  background-color:rgb(255,204,153);
  padding:10px;
}
input.text {
  border:none;
  padding:3px;
  background-color:rgb(255,255,153);
  font-family:Times,serif;
  font-size:17px;
  font-weight:bold;
}
```

Das form-Element erhält eine orangefarbene Hintergrundfarbe und einen Innenabstand von 10 Pixel zum Inhalt. Für Eingabefelder, die das Attribut `class="text"` haben, wird festgelegt, dass der Rahmen unterdrückt wird. Zum Eingabetext soll ein Innenabstand von 3 Pixel eingehalten werden. Das Feld erhält eine gelbe Hintergrundfarbe. Vom Anwender eingegebener Text soll in Times oder einer anderen Serifenschrift mit einer Schriftgröße von 17 Pixel und fett formatiert werden.

4.11 Allgemeine HTML-Attribute

Einige der HTML-Attribute, die in fast allen Elementen notiert werden dürfen, haben wir bereits kennen gelernt. Besonders attraktiv ist das title-Attribut, weil es einen sichtbaren und leicht nachvollziehbaren Output erzeugt, nämlich Tooltipp-Fenster beim Überfahren eines Elements mit der Maus. Gerade in den zuvor beschriebenen Formularelementen können title-Attribute gute Dienste leisten, indem sie Kurzhilfen zum jeweiligen Eingabefeld ausgeben, sobald der Anwender in das Feld positioniert.

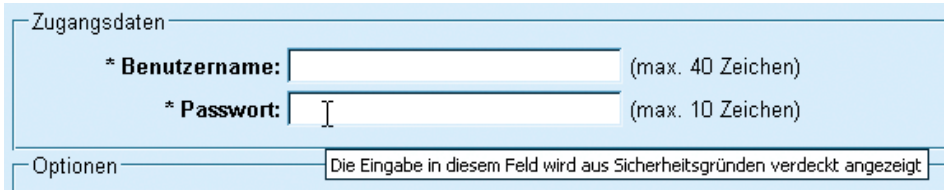


Abbildung 4.76: Kurzinfos zu Formularfeldern mit dem title-Attribut

Der Quelltextauszug zum Passwortfeld im abgebildeten Beispiel lautet:

```
<input type="password" maxlength="10" name="pwd"
       id="pwd" style="width:200px"
       title="Die Eingabe in diesem Feld wird aus
              Sicherheitsgründen verdeckt angezeigt">
```

Hier noch mal alle Allgemeinattribute auf einen Blick:

- `title=` ermöglicht es, Elemente unabhängig von ihrem Inhalt mit einem Labeltext auszuzeichnen. Dieser Text wird von den meisten Browsern in Form eines Tooltipp-Fensters beim Überfahren des Elementbereichs mit der Maus eingeblendet.
- `id=` ermöglicht es, beliebigen Elementen eines HTML-Dokuments einen dokumentweit eindeutigen Namen zu geben. Dieser Name kann in zentralen CSS-Bereichen angesprochen werden, um elementindividuelle Formatierungen zentral zu definieren. Der Name ist aber auch von Bedeutung, wenn mit JavaScript über das standardisierte DOM (Document Object Model) auf ein bestimmtes Element zugegriffen werden soll.
- `style=` ermöglicht es, Elemente individuell mit CSS-Formaten zu gestalten.
- `class=` ermöglicht es, beliebige Elemente zu Klassen zusammenzufassen. Alle Elemente, die dieses Attribut enthalten und dabei den gleichen Klassennamen als Wert zugewiesen bekommen, gehören zur entsprechenden Klasse. Für eine solche Klasse können in zentralen CSS-Bereichen klassenspezifische Formate definiert werden.
- `lang=` ermöglicht es, texttragenden Elementen eine Sprache zuzuordnen. Der Wert sollte ein Sprachenkürzel entsprechend internationalem Standard sein (maßgeblich sind hierbei die Normen ISO 639-1 und ISO 639-2, zu finden auf den Webseiten der Library of Congress unter *http://www.loc.gov/standards/iso639-2/*).
- `dir=` ergänzt das lang-Attribut und erlaubt es, die Schreibrichtung zu bestimmen.

Ferner können in den meisten HTML-Elementen auch so genannte Event-Handler notiert werden, wie z.B. `onMouseover=`, `onClick=` usw. Diese stellen jedoch eine direkte Verknüpfung zu einer clientseitigen Programmiersprache wie JavaScript her. Deshalb behandeln wir sie auch erst im Buchteil über JavaScript/DOM.

5 Seitenlayouts

In Kapitel 4 lernten Sie HTML-Elemente, CSS-Eigenschaften und den korrekten Umgang mit beiden Basissprachen kennen. Das Kapitel über Seitenlayouts zeigt, wie Sie mit diesen Bordmitteln professionelle und standardkonforme Seitenlayouts erstellen können. Gestalterische Kreativität und Genialität bei der Komposition von Farbe und Form ist zwar nicht vermittelbar, wohl aber das Handwerkszeug und die richtigen Kniffe, um gestalterische Wünsche umsetzen zu können. Das Kapitel hat Workshop-Charakter.

5.1 Seitenlayouts mit CSS

Wenn Webdesigner der ersten Generation Webseitenlayouts entwarfen, griffen sie dabei zu zwei wesentlichen Mitteln: einem so genannten blinden Pixel und blinde Tabellen. Der blinde Pixel war eine 1 × 1 Pixel große Grafik, die im GIF-Format abgespeichert wurde. Die Farbe des einen Pixels war egal, aber sie musste als transparent definiert sein. Der blinde Pixel konnte nun in img-Elemente eingebunden werden, wobei als Höhe und Breite der Grafik angegeben wurde, was gerade benötigt wurde. Auf diese Weise verschafften sich die Designer in Zeiten von Netscape 3 bis 4 und Internet Explorer 3 Platz auf der Seite, erzwangen Abstände und Ränder. Das zweite Mittel, die blinde Tabelle, war ebenso wichtig. Zwischen `<body>` und `</body>` wurde stets erst einmal das Gerüst einer Tabelle mit 100% Breite und 100% Höhe, aber ohne sichtbaren Rahmen notiert. Weil sich die Browser zum Teil starke Interpretationsfreiheiten erlaubten, waren meist noch ein, zwei innere Tabellen nötig, um mehrspaltige Layouts einigermaßen festzuzurren.

Hier hat mittlerweile ein deutlicher Paradigmenwechsel stattgefunden. Die attraktive optische Verteilung von Inhalten auf einer Seite wird heute nicht mehr über Tabellen gelöst, sondern über Bereiche, die mithilfe von CSS auf gewünschte Weise positioniert, zueinander ausgerichtet und ins Verhältnis gesetzt werden. Zu verdanken ist dieser Paradigmenwechsel in erster Linie den unermüdlichen Standardisierungsanstrengungen des W3-Konsortiums, gepaart mit der Bereitschaft und der Einsicht der Browser-Anbieter, diese Standards strikter einzuhalten.

Weiterführende Links:

CSS-Anwendungsbeispiele:
http://css.fractatulum.net/beispiele.htm
CSS-Zengarden (das gleiche Manifest in lauter unterschiedlichen, meditativ schönen Layouts):
http://www.csszengarden.com/
Little Boxes (CSS-Boxen für Seitenlayouts):
http://www.thenoodleincident.com/tutorials/box_lesson/boxes.html
Open Source Web Design (fertige, frei verfügbare Layouts):
http://www.oswd.org/
Slants (Polygon-Effekte mit CSS):
http://infimum.dk/HTML/slantinfo.html

5.1.1 Typische Seitenaufteilungen

Bevor wir auf die Techniken eingehen, werfen wir einen Blick auf typische Seitenaufteilungen.

Portallayout

Ein so genanntes Portal verwendet die Optik von Bauten wie dem Arc de Triumphe in Paris: zwei mächtige Stützen links und rechts und ein üppiger Oberbau.

Das Portallayout signalisiert einerseits überlegene Ruhe und Ausgewogenheit, andererseits aber auch eine Fülle von Informationen und unterschiedlichen Angeboten. Verwendet wird es gerne von Online-Zeitungen und -Zeitschriften, aber auch von großen Unternehmen oder Sites, die sich als werbefinanziertes Linkportal zu allgemeinen Themen wie Gesundheit oder Recht verstehen.

Eine der Säulen, meist die linke, wird für eine Navigationsleiste verwendet. Die andere Säule dient zum Featuring bestimmter Themen, aber auch für Werbung. Der Kopfbereich enthält meist auch Werbung sowie Logos, eine Navigation zum Direktaufruf häufig verlangter Unterseiten, ein Eingabefeld zum Durchsuchen der Site oder Login-Felder für Mitglieder, die sich nicht öffentliche Unterseiten aufrufen möchten.

Die Technik hinter Sites mit solchen Layouts besteht meist in dynamisch generierten Seiten. Das Bearbeiten der Seiten erfolgt in der Regel durch mehrere Redakteure über ein Content Management System.



Abbildung 5.1: Portallayout (hier: Süddeutsche Zeitung)

Winkellayout

Im Gegensatz zur Metapher »Portal« ist diese hier eine Neuschöpfung, die einfach einen ebenfalls häufig vorfindbaren Typ von Seitenlayout beschreiben soll. Beim Winkellayout wird der Seiteninhalt links und oben von optisch hervorgehobenen Leisten flankiert.

Das Winkellayout signalisiert mehr Dynamik als das Portallayout. Die Navigation ist dominanter, es wird mehr Hypertext-Feeling vermittelt. Der Inhalt erhält auch optisch mehr Raum.

Geeignet ist dieses Universallayout eigentlich für fast alle Arten von Seiten, deren Aufgabe nicht darin besteht, durch Originalität zu bestechen, sondern durch Funktionalität bei umfangreichen Inhalten. Für die Hauptnavigation links steht genügend Raum zur Verfügung. Die Kopfleiste wird in der Regel für ein Logo verwendet, das sich gut in Winkelnähe oben links platzieren lässt. Die restliche obere Leiste enthält je nach Projekt Werbebanner und/oder eine Zweitnavigation zum Direktaufruf häufig verlangter Unterseiten oder eine Möglichkeit zum Durchsuchen der Site.

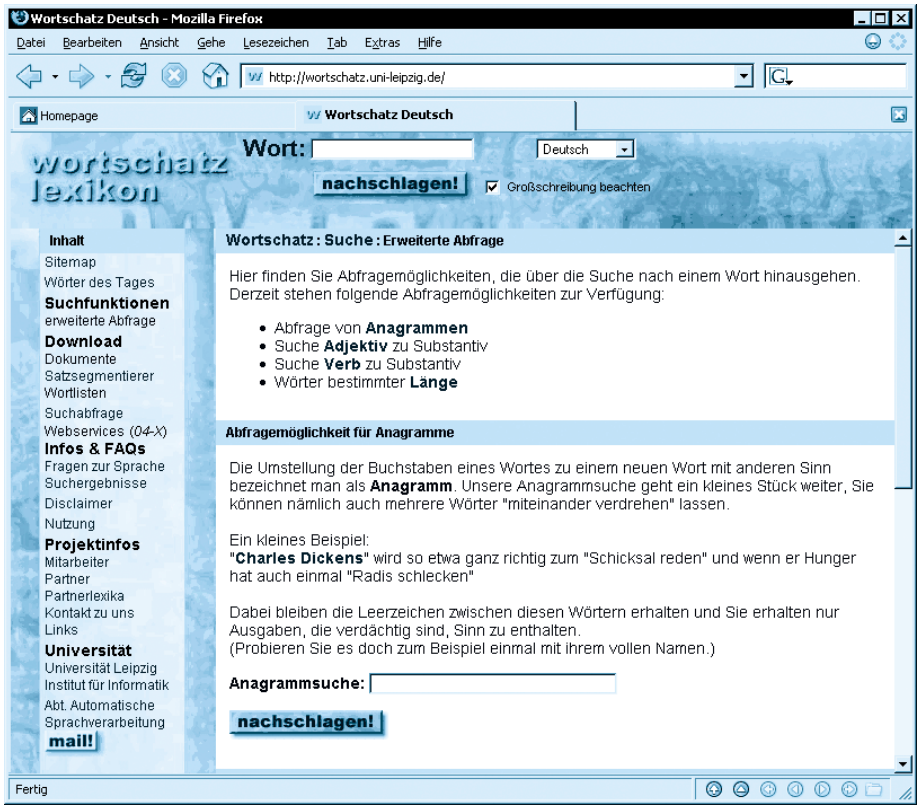


Abbildung 5.2: Winkellayout (hier: Wortschatzlexikon der Uni Leipzig)

Individuelles Layout mit verteilten Bereichen

Solche Layouts zeichnen sich dadurch aus, dass sie zwar ebenfalls mit verteilten und aufgabenspezifischen Bereichen arbeiten, diese aber einfach mal anders und unorthodoxer verteilen als bei den zuvor genannten beiden Typen.

Solche freieren Layouts werden meist für individuelle kleinere Homepages, Auftritte oder Webpräsenzen benutzt, die in der Regel auch keine Werbebanner schalten. Das Ziel ist eine ordentliche, professionelle Präsentation mit Betonung des eigenen Akzents. Daher wird auf solchen Sites auch eher mal mehr Farbe gewagt.



Abbildung 5.3: Unorthodoxe Bereichsaufteilung (hier: Homepage einer Travestie-Tanzband)

5.1.2 HTML- und CSS-Basis für ein Portallayout

Eines muss man bei allem Hype, den das boomende CSS ausgelöst hat, zugeben: Für ein dreispaltiges Layout war eine einfache Tabelle mit drei Zellen, wie man sie früher benutzt hat, gar nicht so ungeeignet. Dennoch werden wir die Lösung über CSS-positionierte Bereiche vorstellen, da sie die modernere und zukunftsorientiertere ist.

Schritt 1: Das HTML-Rohgerüst

Listing 5.1: HTML-Rohgerüst für Portallayout

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
   <head>
      <meta http-equiv="content-type" content="text/html;
         charset=ISO-8859-1">
      <title>Portal</title>
      <link rel="stylesheet" type="text/css" href="styles.css">
   </head>
   <body>
```

```
        <div id="header">
           Kopf
        </div>
        <div id="navigation">
           Navigation
        </div>
        <div id="content">
           Inhalt
        </div>
        <div id="features">
           Features
        </div>
     </body>
  </html>
```

Es fällt auf, dass das Markup sehr aufgeräumt wirkt. Den sichtbaren Dateikörper prägen vier Bereiche mit den id-Namen `header` (dies wird der Kopfbereich), `navigation` (der Navigationsbereich als linke Portalsäule), `content` (der inhaltstragende Bereich in der Mitte) und `features` (ein Bereich für Hinweise auf Highlights oder Aktuelles, die rechte Portalsäule). Innerhalb der div-Bereiche wird testweise einfach ein Wort geschrieben, das den entsprechenden Bereich benennt.

Wie erkennbar ist, geht aus dem Markup nichts über die optische Verteilung der Bereiche hervor. Würde die Seite ohne CSS angezeigt, dann würde einfach zuerst der Kopfbereich, dann die Navigation, dann der Inhalt und zum Schluss die Feature-Hinweise angezeigt. Abgesehen von der fehlenden Optik ist das durchaus eine sinnvolle Aufteilung, die in dieser Form etwa auch für eine akustische Ausgabe der Seite geeignet ist.

Im Dateikopf wird mittels <link>-Tag eine externe CSS-Datei namens ***styles.css*** eingebunden. Da das Layout einheitlich für etliche Seiten gelten soll, ist das Arbeiten mit extern definierten CSS-Formaten die einzig sinnvolle Lösung.

Schritt 2: Das CSS-Rohgerüst

Listing 5.2: Rohgerüst der CSS-Datei »styles.css« für ein Portallayout

```
html, body {
   background-color:white;
   margin:0px;
   padding:0px;
}

#header {
   background-color:rgb(190,215,235);
   width:100%;
   height:100px;
   padding:0px;
   margin:0px;
}

#navigation {
```

```
    position:absolute;
    background-color:rgb(210,235,255);
    top:100px;
    left:0px;
    width:180px;
    margin:0px;
    padding:0px;
}

#content {
    margin-top:0px;
    margin-left:180px;
    margin-right:180px;
}

#features {
    position:absolute;
    background-color:rgb(210,235,255);
    top:100px;
    right:0px;
    width:180px;
    margin:0px;
    padding:0px;
}
```

In dieser Phase definieren wir lediglich die Formate für die formale Verteilung der Bereiche.

Das Layout arbeitet mit einer Mischung aus positionierten und nicht positionierten Bereichen. Warum das so ist, werden wir noch erläutern. Damit jedoch auch die nicht positionierten Bereiche korrekt platziert werden, müssen die Default-Ränder, die der Browser dem sichtbaren Dateikörper spendiert, auf 0 gesetzt werden. Der erste Definitionsblock setzt Rand (`margin`) und Innenabstand (`padding`) für die Elemente `html` und `body` auf 0. Außerdem wird Weiß explizit als Standardhintergrundfarbe definiert.

Über die Syntax mit dem Gatterzeichen (`#id-Name`) werden die vier Bereiche definiert. Die Definitionen sorgen für die optische Verteilung der Bereiche als Portal. Der Kopfbereich (das div-Element mit dem id-Namen `header`) steht im normalen Textfluss, wird also nicht positioniert. Da das Element in HTML unmittelbar hinter dem einleitenden `<body>`-Tag notiert ist, beginnt es in der oberen linken Ecke des Anzeigefensters. Da dessen Ränder explizit auf 0 gesetzt wurden, ist sichergestellt, dass der div-Bereich wirklich an der Pixelposition 0,0 beginnt.

Zur Sicherheit erhält der Bereich `headers` jedoch auch noch mal einen expliziten Rand von 0 Pixel. Ebenfalls zur Sicherheit bekommt er explizit die volle verfügbare Breite von 100% zugewiesen. Zwingend nötig ist diese Angabe für Browser, die das Box-Modell von CSS richtig interpretieren, allerdings nicht, da Blockelemente, die sich im normalen Textfluss befinden, ohnehin auf die volle verfügbare Breite ausgedehnt werden sollen.

Als Höhe werden 100 Pixel festgelegt. Platz genug für die typischen Elemente eines solchen Kopfbereichs, wie Logo, Werbebanner, Suche und Highlight-Navigation.

Die beiden Säulen des Portals, also die Bereiche mit den id-Namen `navigation` und `features`, werden absolut positioniert, also aus dem normalen Textfluss herausgenommen. Daher erhalten sie auch Angaben zu `top` (Anfangsposition von oben) und `left` (Anfangsposition von links) bzw. `right` (Anfangsposition von rechts). Da die Anzeigebreite des Fensters nicht bekannt ist, ist die Eigenschaft `right` sehr hilfreich, um die Position der rechten Säule von rechts her zu definieren.

Beide Säulen erhalten eine Breite von 180 Pixel. Das sollte genügen, um Navigationslinks und Feature-Blöcke unterzubringen. Immerhin gehen dem eigentlichen Inhalt dadurch ja schon 360 Pixel Breite ab. Bei einigen portablen Geräten bekommen wir dadurch schon ein Problem mit der Anzeigebreite. Andererseits sind Geräte mit Mini-Displays nun mal nicht wirklich geeignet, um Seiten mit Portal-Charakter, also mit großer gleichzeitiger Angebotsfülle, anzuzeigen. Bei Geräten ab der typischen Standardbreite von 640 Pixel bleibt in unserem Layout aber zumindest schon ein gerade noch akzeptabler Mittelstreifen von 280 Pixel für den Inhalt übrig.

Der inhaltstragende Bereich, also das div-Element mit dem id-Namen `content`, bleibt wie der Kopfbereich im Textfluss, wird also nicht positioniert, um flexibel zu bleiben. Seine »Positionierung« wird stattdessen über definierte Ränder nach oben, links und rechts erreicht. Nach oben (`margin-top`) werden 0 Pixel definiert. Da ja der Inhaltsbereich ebenso wie der Kopfbereich im normalen Textfluss bleibt, beginnt er einfach direkt unterhalb des Kopfbereichs, und zwar mit sicherheitshalber erzwungenen 0 Pixel Abstand. Rechts und links (`margin-left` und `margin-right`) wird dagegen ein Rand von je 180 Pixel gesetzt. Das entspricht jeweils der Breite der Säulen.

Alle Innenabstände (`padding`) der vier Bereiche werden explizit auf 0 gesetzt. Einen optisch ordentlichen Abstand zum Bereichsrand werden wir stattdessen später bei den inneren Elementen erzeugen.

Die Bereiche `header`, `navigation` und `features` haben je eine Hintergrundfarbe erhalten. Beim Inhaltsbereich ist dies nicht nötig, da er die explizit definierte Hintergrundfarbe des Elternelements, also des body-Elements, erbt.

Zusammen mit den Testtexten in jedem Bereich können wir das Grundgerüst des Layouts nun im Browser wie abgebildet optisch gut nachvollziehen.

Es fällt auf, dass die Bereiche `navigation`, `content` und `features` keine definierte Höhe haben. Im Test sind sie überhaupt nur andeutungsweise sichtbar, weil sie ein Wort als Inhalt enthalten haben. Es wäre durchaus möglich, eine Höhe vorzugeben. Da die Bereiche jedoch unterschiedliche Inhalte erhalten können, was zu unterschiedlichen Realhöhen führt, werden wir später eine andere Lösung bevorzugen, für die ein kleines JavaScript erforderlich ist.

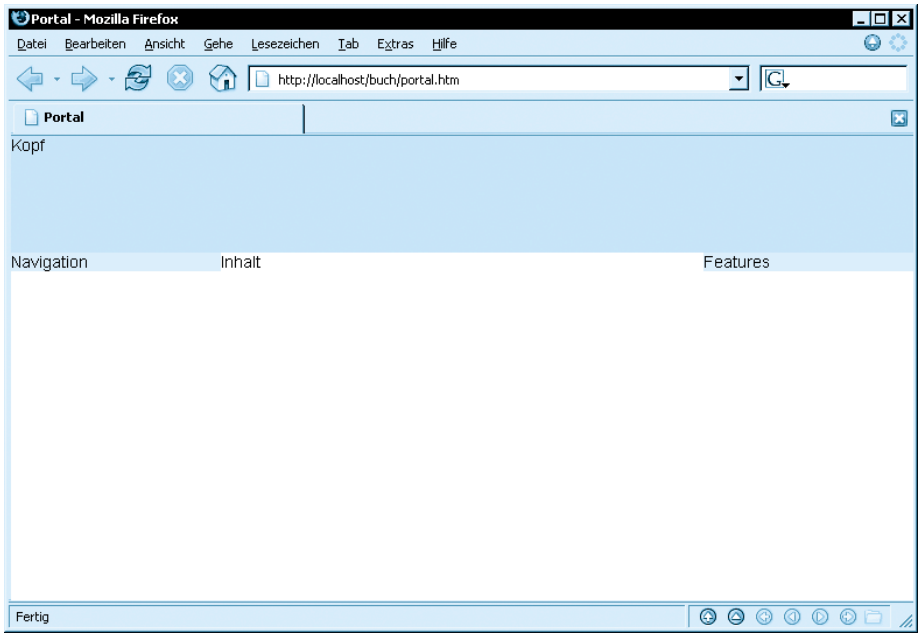


Abbildung 5.4: Rohgerüst des Portallayouts

Schritt 3: CSS-Definitionen für Elemente des Kopfbereichs

Nachdem die grundsätzliche Bereichsaufteilung realisiert ist, können wir das Stylesheet um »innere« Elemente erweitern. Im Kopfbereich soll links oben ein Logo platziert werden, dessen Ausmaß 82 Pixel Breite × 87 Pixel Höhe beträgt. Im HTML-Quelltext fügen wir das Bild in den Kopfbereich ein:

```
<div id="header">
<img id="logo" src="logo.gif" alt="News Web">
</div>
```

Das Bild erhält einen id-Namen `logo`. Ansonsten werden nur die Quelle und der obligatorische Alternativtext angegeben. Alle weitere Daten werden im Stylesheet notiert, und zwar wie folgt:

```
#logo {
   position:absolute;
   top:5px;
   left:12px;
   width:82px;
   height:87px;
}
```

Durch `#logo` wird der Bezug zum Element `id="logo"` hergestellt. Das betroffene Element, also die Logografik, wird absolut mit geeignetem Abstand oben und links positioniert. Breite und Höhe der Grafik werden ebenfalls hier angegeben.

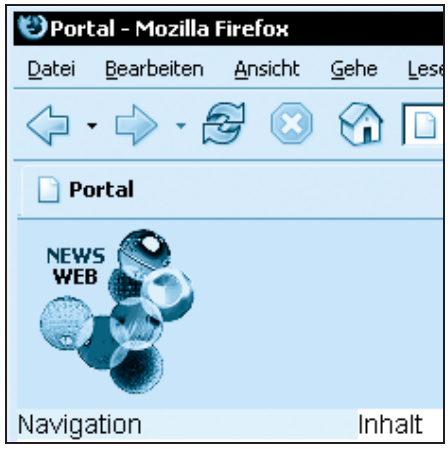


Abbildung 5.5: Im Kopfbereich platziertes Logo

Ähnlich verfahren wir mit dem Werbebanner. Die Bannergrafik muss jedoch wie bei solchen Bannern üblich in einen Link eingebettet werden. Den HTML-Code des Kopfbereichs erweitern wir wie folgt:

```
<div id="header">
<img id="logo" src="logo.gif" alt="REAL NEWS Logo">
<a id="banner" href="http://www.example.org/">
   <img id="banner_pic" src="banner.png" alt="Werbebanner">
</a>
</div>
```

Das a-Element des Links erhält den id-Namen `banner` und die dadurch anklickbare Grafik den id-Namen `banner_pic`. Bei der Grafik werden wieder nur die Quelle und der Alternativtext notiert. Den Rest erledigen wir im Stylesheet:

```
#banner {
   position:absolute;
   top:5px;
   left:196px;
}

#banner_pic {
   border:none;
   width:468px;
   height:60px;
}
```

Das a-Element, adressiert durch den Selektor `#banner`, wird absolut an geeigneter Stelle positioniert. Für die Grafik wird neben Breite und Höhe (468 × 60 Pixel ist übrigens eines der Standardformate für Werbebanner) auch die Angabe `border:none` notiert. Dadurch wird ein Rahmen um die Grafik unterdrückt, was in diesem Fall wichtig ist, da der Browser ansonsten wegen der Position innerhalb des Links einen Rahmen um die Grafik zeich-

nen würde. Die Grafik soll jedoch nur durch Überfahren mit der Maus oder Ansteuern mit der Tabulatortaste als Link erkennbar sein, da die Bannerfunktion Anwendern hinlänglich bekannt ist.

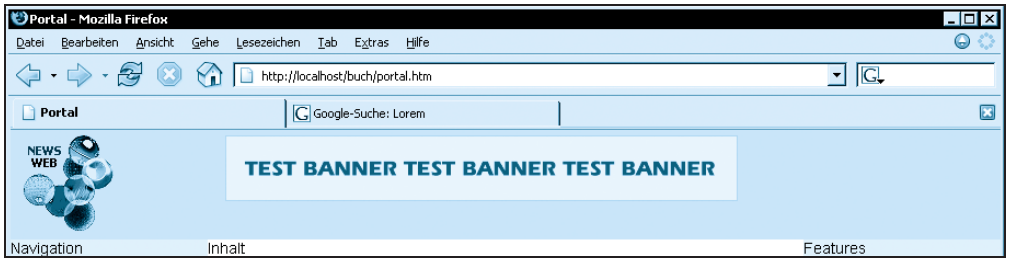


Abbildung 5.6: Kopfbereich mit Logo und Banner

Die Bannergrafik ist natürlich ein vorläufiger Dummy, ebenso wie die im Banner-Link angegebene Adresse.

Zuletzt soll nun unterhalb des Banners, unmittelbar am unteren Rand des Kopfbereichs, eine Leiste für »Hotlinks« platziert werden. Diese werden wir zunächst als Grafik realisieren. Um die darin sichtbaren Schaltflächen als Link zu definieren, können wir beispielsweise eine verweis-sensitive Grafik daraus machen. In ***Abschnitt 5.2*** werden wir darauf eingehen, wie sich attraktiv aussehende Linkleisten auch mithilfe von HTML und CSS realisieren lassen.

Die Grafik wird ähnlich wie das Logo absolut positioniert. In HTML sieht der Kopfbereich am Ende folgendermaßen aus:

```
<div id="header">
<img id="logo" src="logo.gif" alt="REAL NEWS Logo">
<a id="banner" href="http://www.example.org/">
   <img id="banner_pic" src="banner.png" alt="Werbebanner">
</a>
<img id="top_navigation" src="topnav.gif"
     alt="Obere Navigation">
</div>
```

Das Stylesheet wird für die Grafik mit dem id-Namen `top_navigation` wie folgt erweitert:

```
#top_navigation {
   position:absolute;
   top:79px;
   left:196px;
   border:none
   width:459px;
   height:21px;
}
```

Da die Grafik eine Höhe von 21 Pixel hat, bekommt sie als Wert für `top` 79 Pixel zugewiesen. Dadurch schließt die Unterseite der Grafik bei 100 Pixel ab, was genau der Unterkante des Kopfbereichs entspricht. Links wird die Grafik bündig zum Banner ausgerichtet. Auch die Elemente des Inhaltsbereichs werden später bündig dazu positioniert.



Abbildung 5.7: Kopfbereich mit Logo, Banner und Topnavigation

Schritt 4: CSS-Definitionen für Elemente des Inhaltsbereichs

Im Inhaltsbereich soll üblicher Text stehen – im Wesentlichen also Überschriften, Textabsätze, Listen und Tabellen. Die entsprechenden Elemente sollen sich optisch ins Layout einpassen und mithelfen, der Site Wiedererkennungswert zu verleihen. In unserem Beispiel definieren wir in der CSS-Datei folgende Formate:

```
h1,h2,p,ul,ol,table {
   margin-left:16px;
   margin-right:16px;
   font-family:"Century Schoolbook",serif;
}

h1 {
   font-weight:bold;
   font-style:normal;
   font-size:130%;
   line-height:1.1em;
   color:rgb(90,115,135);
   border-left:rgb(190,215,235) solid 20px;
   border-bottom:rgb(190,215,235) solid 4px;
   padding-left:12px;
   margin-top:32px;
}

h2 {
   font-weight:bold;
   font-style:normal;
   font-size:110%;
   line-height:1.1em;
   color:rgb(90,115,135);
   border-bottom:rgb(190,215,235) solid 2px;
   margin-top:24px;
}
```

```
p.intro {
   text-align:left;
   font-weight:normal;
   font-style:italic;
   font-size:100%;
   line-height:1.1em;
   margin-left:48px;
   color:black;
}

p.text {
   text-align:justify;
   font-weight:normal;
   font-style:normal;
   font-size:100%;
   line-height:1.1em;
   color:black;
}

li {
   text-align:left;
   font-weight:normal;
   font-style:normal;
   font-size:100%;
   line-height:1.1em;
   color:black;
}

td,th {
   text-align:left;
   font-style:normal;
   font-size:100%;
   line-height:1.1em;
   color:black;
}

th {
   font-weight:bold;
}
```

Mit `margin-left` und `margin-right` wird für alle inhaltstragenden Blockelemente, die direkt innerhalb des Inhaltsbereichs vorkommen (`h1`, `h2`, `p.text`, `p.intro`, `ul`, `ol`, `table.text`), ein Randabstand links und rechts von je 16 Pixel definiert. Dies bezieht sich auf das Elternelement. Da es sich um Elemente handelt, die in dieser Form im div-Bereich mit dem id-Namen `content` vorkommen sollen, entsteht ein absoluter Rand von 196 Pixel links und rechts. Damit beginnen diese Elemente auch bündig unterhalb des Werbebanners und der Topnavigation. Solche »gedachten Linien« sind für die ruhige und saubere Wirkung eines Layouts ungemein wichtig.

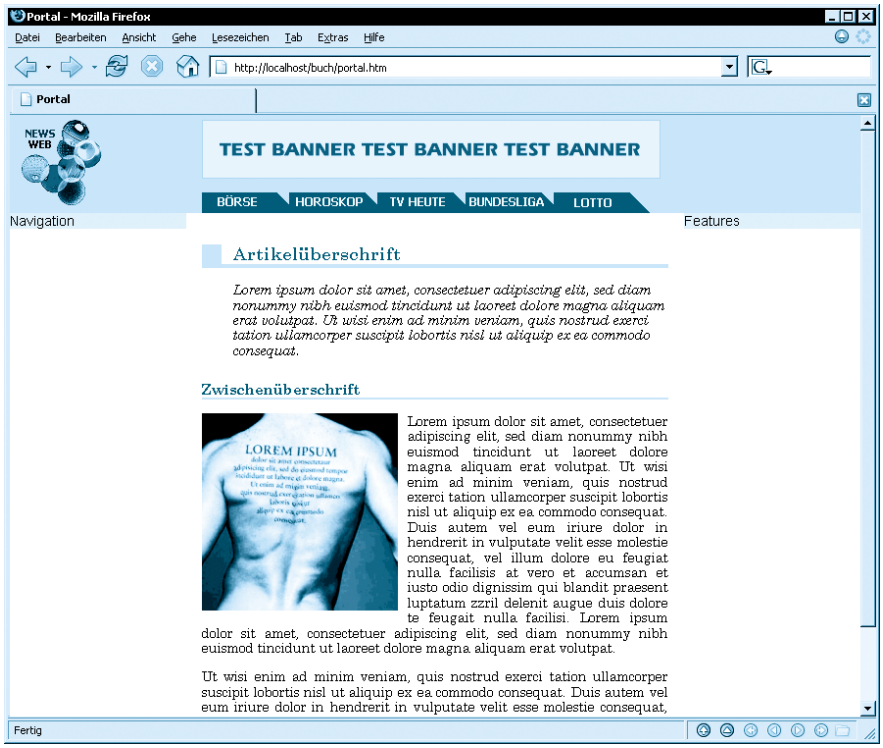


Abbildung 5.8: Portallayout mit Textformaten und Beispieltext

Als Grundschriftart für den Fließtext wird eine Serifenschrift (`Century Schoolbook`) gewählt. Als generische Schriftart wird folglich `serif` angegeben.

Es werden nur zwei Überschriftenebenen benötigt – daher erhalten nur die Überschriftenebenen 1 und 2 spezifische Formate. Mithilfe von Rahmenlinien, Schriftgrößen und Abständen werden die Überschriftenebenen optisch so definiert, dass sie mit dem Layout harmonieren und ihre Textstrukturierungsfunktion optisch unterstrichen wird.

Normaler Fließtext wird als Blocksatz ausgerichtet (`text-align:justify`). Auch das prägt bei längeren Texten die Gesamtwirkung des Layouts deutlich in Richtung Ruhe und Ausgeglichenheit. Ein Signal der objektiven Vermittlung von News geht davon aus. Eine Sonderstellung nimmt der Intro-Absatz unterhalb der Seiten-h1-Überschrift ein. Er wird kursiv mit Flattersatz formatiert und beginnt bündig unterhalb des Textes der h1-Überschrift.

Schritt 5: CSS-Definitionen für Inhalte der Säulenbereiche

Im Navigationsbereich werden wir fürs Erste – ähnlich wie bei der Topnavigation – einfach eine Grafik einfügen. Für die rechte Säule, also den Feature-Bereich, definieren wir Boxen, die kleinen Fenster ähneln und damit signalisieren, dass sie einen Ausblick auf Feature-Themen geben. Dazu werden zwei CSS-Formate benötigt:

```
div.box_title {
   margin-top:16px;
   margin-left:8px;
   margin-right:8px;
   background-color:rgb(130,130,130);
   color:white;
   font-family:Tahoma,sans-serif;
   font-size:12px;
   font-weight:bold;
   padding:1px;
}

div.box_content {
   margin-left:8px;
   margin-right:8px;
   margin-bottom:8px;
   background-color:white;
   border:rgb(130,130,130) solid 1px;
   color:black;
   font-family:"Century Schoolbook",serif;
   font-size:80%;
   font-weight:normal;
   padding:3px;
}.
```

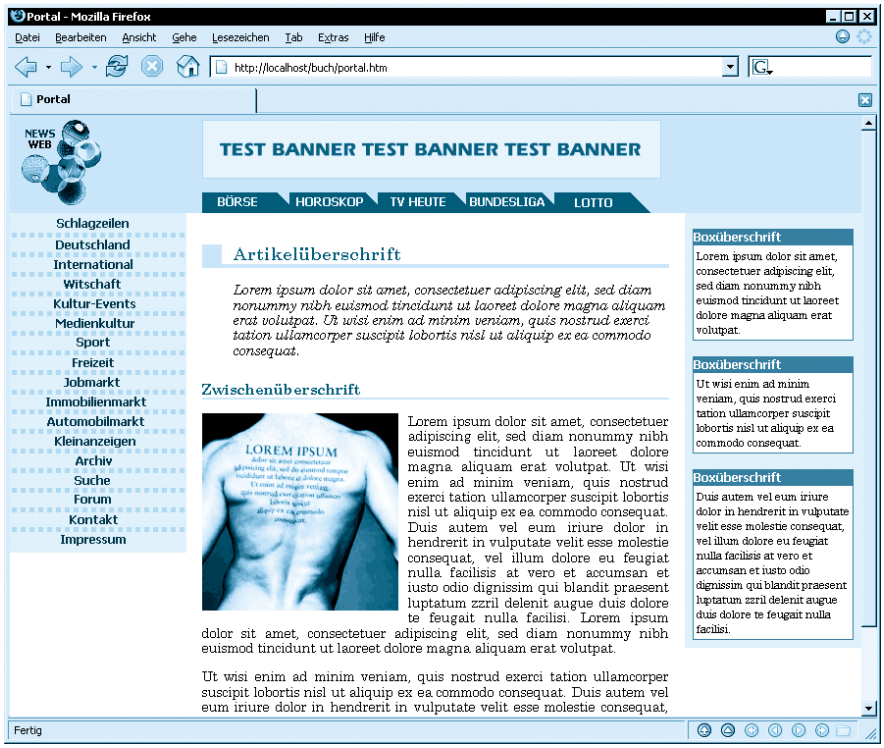


Abbildung 5.9: Portallayout mit Hauptnavigation und Feature-Boxen

Verwendet wird für beide Boxtypen das neutrale div-Element. Die Definitionen sind so gesetzt, dass jeweils zwei hintereinander notierte div-Elemente der Klasse `box_title` und `box_content` eine vollständige Box bewirken

Die Boxentitel sind dunkelgrau (`background-color:rgb(130,130,130)`) und erhalten weiße Schrift (`color:white`) in der serifenlosen Schriftart Tahoma (`font-family:Tahoma,sans-serif`), die auch bei den Linktexten der Navigationsleisten oben und links verwendet wird. Die Boxinhalte dagegen erscheinen in schwarzem Text (`color:black`) auf weißem Grund (`background-color:white`), wobei die Schriftart die gleiche ist wie im Inhaltsbereich, jedoch kleiner (`font-size:80%`). Die Boxinhalte erhalten einen Rahmen (`border:rgb(130,130,130) solid 1px`) in der Farbe der Boxentitelhintergrundfarbe. Links und rechts haben beide Boxtypen einen Abstand von 8 Pixel (`margin-left:8px`, `margin-right:8px`). Boxentitel haben oben einen Abstand von 16 Pixel (`margin-top:16px`), was für den nötigen Abstand zwischen den Feature-Boxfenstern sorgt.

Schritt 7: Anpassung der Höhen von Navigation, Inhalt und Features

Mit den Navigationsgrafiken, dem formatierten Fließtext im Inhalt und den gestalteten Feature-Boxen haben wir nun bereits eine attraktiv aussehende Seite. Was jedoch noch negativ auffällt, sind die unterschiedlichen Längen der beiden Säulen. Da diese nicht bis zum Artikelende herunter reichen, wirken sie auch gar nicht wie »bodenständige« Säulen. Hier macht sich ein Nachteil von mehrspaltigen Layouts mit CSS gegenüber klassischen Tabellenlayouts bemerkbar, der von Vertretern der neuen Layoutgeneration gerne unter den Tisch gekehrt wird. Bei Tabellen entsteht das Problem der unterschiedlichen Längen nicht, solange die Spalten eine einzige Tabellenzeile darstellen, bei der immer alle Zellen die Höhe der längsten haben.

Mithilfe eines kleinen JavaScripts können wir die Spalten unseres Portallayouts jedoch an der Artikelspalte ausrichten. Da das Script auf jeder Seite erneut seinen Dienst versehen muss, wird es in einer separaten JavaScript-Datei notiert. Die Datei hat folgenden Inhalt:

Listing 5.3: JavaScript für Layoutanpassung bei Portallayout

```
function correct_layout() {
   if(document.getElementById('content').offsetHeight) {
      var div_heights = new Array(
        document.getElementById('content').offsetHeight,
        document.getElementById('navigation').offsetHeight,
        document.getElementById('features').offsetHeight
      );
      div_heights = div_heights.reverse(div_heights.sort(num_sort));
      var style_height = div_heights[0] + "px";
      document.getElementById('navigation').style.height = style_height;
      document.getElementById('content').style.height = style_height;
      document.getElementById('features').style.height = style_height;
   }
}
function num_sort(a,b) {
   return a-b;
}
```

Auf den Quelltext gehen wir an dieser Stelle nicht näher ein. Das Script enthält eine Funktion namens `correct_layout()`, die dafür sorgt, dass alle drei Portalspalten, also die div-Bereiche mit den id-Namen `navigation`, `content` und `features`, die gleiche Höhe erhalten. Die JavaScript-Datei muss allerdings im HTML-Dokument eingebunden werden und die Funktion `correct_layout()` muss aufgerufen werden. Der Beginn des HTML-Quelltextes unserer Beispielseite lautet daher:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Portal</title>
<link rel="stylesheet" type="text/css" href="portal.css">
<script type="text/javascript" src="layout.js"></script>
</head>
<body onLoad="correct_layout()">
<!-- usw. -->
```

Die externe JavaScript-Datei – wir haben sie ***layout.js*** benannt – wird im HTML-Dokumentkopf über das script-Element (`<script...></script>`) eingebunden. Im src-Attribut wird die Datei als URI referenziert. Im type-Attribut wird der Mime-Type für JavaScript (`text/javascript`) angegeben.

Im einleitenden `<body>`-Tag wird ein Event-Handler `onLoad=` notiert. Dies ist ein HTML-Attribut, das eine Verknüpfung zu JavaScript-Code herstellt. Der Event `onLoad` tritt ein, wenn das HTML-Dokument vollständig an den Browser übertragen und geladen ist. Ist dies der Fall, wird die Funktion `correct_layout()` aufgerufen, welche die Höhenkorrektur der drei Layoutspalten besorgt.

Für die heute verbreiteten Browser mit grafischer Oberfläche haben wir nun ein ordentliches Portallayout gebastelt. Wir verwendeten dazu zeitgemäßes, schlankes HTML in Verbindung mit CSS. Einer der Ansprüche an solche Seiten ist jedoch auch, dass sie mit nicht grafischer oder nicht visueller Benutzer-Software gut zugänglich sind.

Ein hervorragendes Werkzeug zum Austesten der »Accessibility« ist der textbasierte Browser Lynx, den es für MS-DOS sowie für alle Arten von Unix-Shells gibt. Lynx beherrscht HTML 4.01 bestens, kann aber verständlicherweise kein CSS umsetzen. Unsere Portalseite sieht in Lynx wie abgebildet aus. Optik spielt bei der Beurteilung keine große Rolle. Wichtig ist, dass die Lektüre des Inhalts nachvollziehbar ist. Angezeigt werden auch keine Grafiken, sondern die im alt-Attribut notierten Alternativtexte. Hier zeigt sich, wie wichtig es ist, diese Texte so zu formulieren, dass sich der Sinn erschließt.

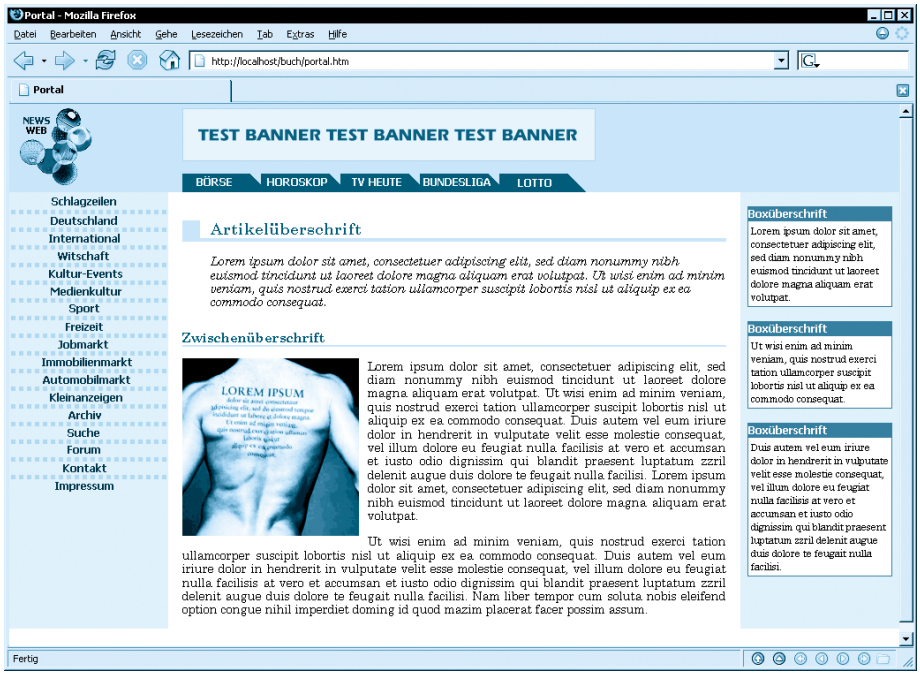


Abbildung 5.10: Mit JavaScript angepasste Spaltenhöhen beim Portallayout

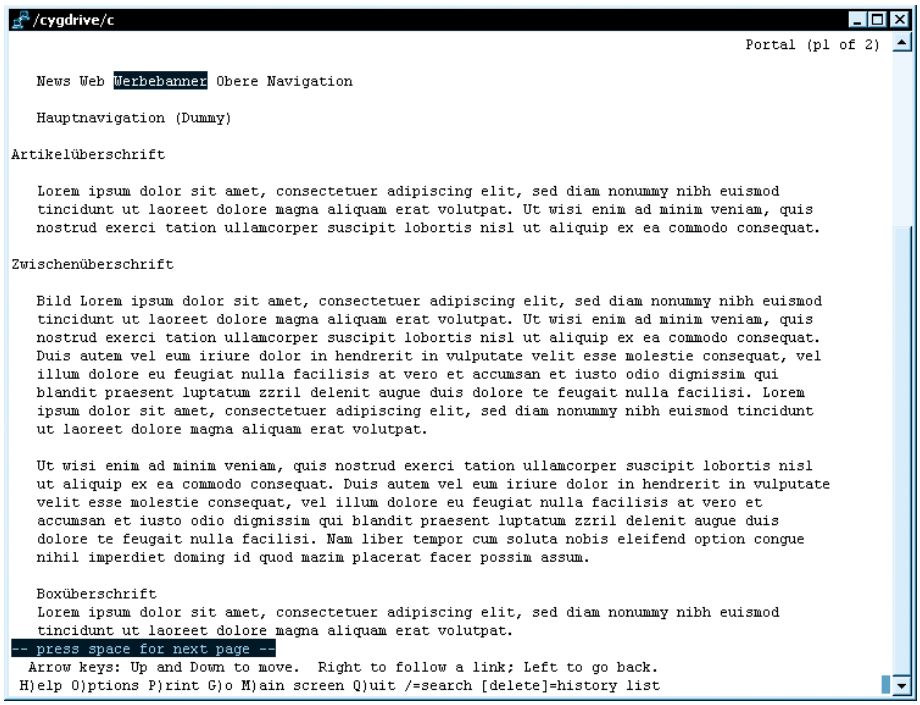


Abbildung 5.11: Portallayout-Seite im textbasierten Lynx-Browser

Da wir anstelle einer echten Navigation vorläufig nur Grafik-Dummies notiert haben, gibt es noch keine Links außer dem Werbebanner. Obere Navigation und Hauptnavigation sind jedoch erkennbar oben angeordnet und etwaige Links würden an den entsprechenden Stellen angezeigt. Anschließend folgt der Artikeltext, dank Verwendung üblicher HTML-Textelemente ordentlich dargestellt. Zum Schluss werden die Feature-Boxen angezeigt, einfach mit Boxenüberschrift und Inhalt der Boxen.

Da die beiden Säulen für Navigation und Feature-Boxen absolut positioniert sind, könnten sie im HTML-Quelltext auch an anderer Stelle notiert werden, etwa nach dem Artikelinhalt. In diesem Fall würde der Artikel in Lynx und vergleichbarer Ausgabe-Software vor der Hauptnavigation angezeigt. Solche Entscheidungen in Bezug auf den Accessibility-Aspekt sollten Sie in Ihre Layout-Überlegungen stets mit einbeziehen.

5.1.3 HTML- und CSS-Basis für ein Winkellayout

Als Nächstes werden wir Schritt für Schritt ein typisches Winkellayout entwickeln.

Schritt 1: Das HTML-Rohgerüst

Beim HTML-Rohgerüst entscheiden wir uns für folgende Struktur:

Listing 5.4: HTML-Rohgerüst für Winkellayout

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
   <head>
      <meta http-equiv="content-type" content="text/html;
         charset=ISO-8859-1">
      <title>Portal</title>
      <link rel="stylesheet" type="text/css" href="styles.css">
   </head>
   <body>
      <div id="top_section">
      </div>
      <div id="main_section">
         <div id="navigation">
           Navigation
         </div>
         <div id="content">
            Inhalt<br>
            Inhalt<br>
            Inhalt<br>
            Inhalt<br>
            Inhalt<br>
            Inhalt<br>
         </div>
      </div>
   </body>
</html>
```

In diesem Fall verwenden wir eine verschachtelte div-Struktur. Es gibt zwar Hardliner, die in solchen Fällen bereits wieder von *tagsoup* (zu Deutsch: Auszeichnungssuppe) sprechen, doch erstens stört diese Struktur in Hinblick auf die Zugänglichkeit des Layouts in nicht grafischen oder nicht visuellen Umgebungen keineswegs und zweitens hat sie unbestreitbare Vorteile, wie wir noch zeigen werden.

Auf der Ebene unterhalb des body-Elements werden zwei Basisbereiche definiert: zuerst ein Bereich mit dem id-Namen `top_section` und anschließend ein Bereich mit dem Namen `main_section`. Während der Bereich `top_section` zunächst leer bleibt, wird der Bereich `main_section` mit zwei inneren div-Bereichen gefüllt. Diese erhalten die sprechenden id-Namen `navigation` und `content`.

Die gewünschte optische Verteilung der Bereiche realisieren wir wieder mithilfe von CSS. Da das Layout für viele Seiten gelten soll, werden die entsprechenden Formatdefinitionen in einer externen Stylesheet-Datei namens *styles.css* vorgenommen. Diese wird über das style-Tag im HTML-Dokumentkopf eingebunden.

Schritt 2: Das CSS-Rohgerüst

In unserem Beispiellayout wollen wir mit einem Farbverlauf arbeiten. Dazu werden folgende Grafiken erstellt:



Abbildung 5.12: Grafiken für Farbverlauf

Die obere Grafik, 640 × 100 Pixel groß, wird als Hintergrund für die obere Leiste dienen, die untere Grafik, 100 × 240 Pixel groß, wird für die linke Leiste benötigt. Damit sind die nötigen Vorarbeiten erledigt. Die CSS-Datei erhält zunächst folgenden Inhalt:

Listing 5.5: Rohgerüst der CSS-Datei »styles.css« für Winkellayout

```
html, body {
   margin:0px;
   padding:0px;
   background-color:rgb(251,241,225);
   background-image:url(back-v.png);
   background-repeat:repeat-y;
}

#top_section {
```

```
    background-color:rgb(248,228,147);
    background-image:url(back-h.png);
    background-repeat:no-repeat;
    height:100px;
    margin:0px;
    padding:0px;
}

#main_section {
    margin:0px;
    padding:0px;
}

#navigation {
    position:absolute;
    left:0px;
    top:180px;
    width:240px;
}

#content {
    margin-left:240px;
    margin-right:0px;
    border-top:double 5px orange;
}
```

Da die meisten Elemente unseres Layouts nicht positioniert werden, sondern im normalen Textfluss bleiben, werden Rand (`margin`) und Innenabstand (`padding`) der Elemente `html` und `body` auf 0 gesetzt. So ist sichergestellt, dass die Inhalte exakt oben links beginnen. Die globale Hintergrundfarbe wird auf ein ganz helles, gegen Weiß tendierendes Gelborange gesetzt. Damit die rechte Leiste optisch zustande kommt, wird für den gesamten body-Bereich eine Hintergrundgrafik bestimmt – die kleinere der beiden Farbverlaufsgrafiken (***back-v.png***). Diese soll jedoch nicht über den gesamten Bereich wiederholt werden, sondern nur eine linke Spalte bilden. Mit `background-repeat:repeat-y` wird genau dieser Effekt erreicht.

Die Definitionen für `#top_section` beziehen sich auf das div-Element mit dem entsprechenden id-Namen. Der Bereich ist nicht positioniert. Da er als im HTML-Quelltext unmittelbar hinter dem einleitenden `<body>`-Tag notiert wird, beginnt er dank der auf 0 gesetzten Werte exakt links oben im Anzeigefenster. Als Höhe werden die gewünschten 100 Pixel festgelegt. Rand und Innenabstand werden auf 0 gesetzt. Ferner wird mit `background-image` die Hintergrundgrafik für den Farbverlauf eingebunden und zwar die breitere der beiden – wir haben sie ***back-h.png*** benannt. Wichtig ist auch die Angabe `background-repeat:no-repeat`, die dafür sorgt, dass die Grafik nur einmal angezeigt wird und kein Tapeteneffekt entsteht. Damit das hellgelbe rechte Ende der Grafik nahtlos in den weiteren Hintergrund übergeht, wird der entsprechende Farbwert als Hintergrundfarbe (`background-color`) für den Bereich definiert.

Der gesamte Bereich unterhalb des div-Bereichs `top_section` wird durch den Bereich `main_section` abgedeckt. Für `main_section` sind keine besonderen Definitionen vorgese-

hen. Ränder und Innenabstände werden sicherheitshalber auf 0 gesetzt. Die Abstände zu den Leisten oben und links werden stattdessen später bei den inhaltstragenden Elementen definiert.

Bleiben die Formatdefinitionen für die beiden div-Bereiche, die innerhalb von `main_section` notiert wurden: `navigation` und `content`. Da die Inhalte dieser beiden Bereiche nebeneinander stehen, aber sich nicht gegenseitig beeinflussen sollen, wird der eine Bereich (`navigation`) absolut positioniert. Da er sich nicht innerhalb eines anderen positionierten Elements befindet, beziehen sich die Angaben zu `top` und `left` auf die obere linke Ecke des Anzeigebereichs. Beim Bereich `content` wird die »Positionierung« über eine Angabe zu `margin-left` so vom linken Rand des Elternelements abgesetzt, dass keine Kollision mit `navigation` entsteht, zumal `navigation` auch eine Breite von 240 Pixel vorgegeben bekommt. Das Elternelement von `content` ist `main_section`. Dieses beginnt links an Position 0, also sind die 240 Pixel Linksabstand bei `content` zugleich der Abstand vom linken Rand des Anzeigebereichs.

Der Bereich `content` erhält schließlich noch links und oben je eine kräftige, warmrote Rahmenlinie oben. Links wird die gleiche Rahmenlinie erscheinen, jedoch ist sie dort Teil der Hintergrundgrafik (*back-v.png*) Durch die Rahmenlinien entsteht noch mal eine optische Betonung des Winkeleffekts.

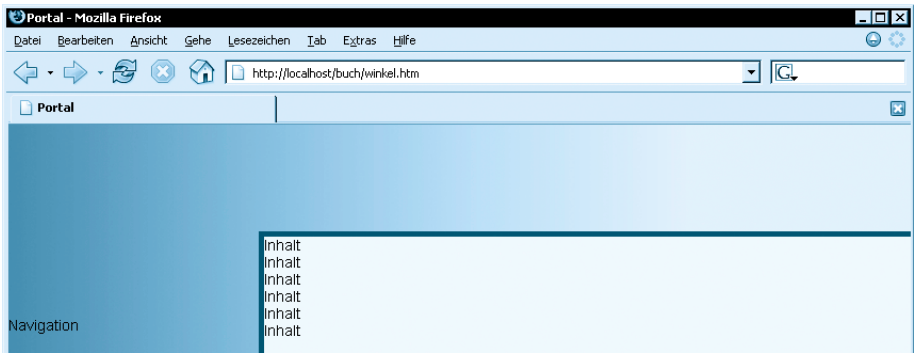


Abbildung 5.13: Rohgerüst des Winkellayouts

Die linke Leiste mit der Verlaufsgrafik passt sich in der Höhe automatisch der Länge des Seiteninhalts an. Der Grund ist, dass die Hintergrundgrafik, welche die linke Leiste optisch darstellt, dem body-Element zugewiesen wurde. Deshalb benötigen wir in diesem Beispiel kein JavaScript für Höhenanpassungen.

Schritt 3: CSS-Definitionen für Elemente des Kopfbereichs

In Winkelnähe oben links soll natürlich ein identitätsstiftendes Logo platziert werden. Weiterhin soll in der oberen Leiste ein Eingabefeld mit Absende-Button zum Durchsuchen der Site platziert werden. Die entsprechenden Elemente in HTML sind eine Grafikreferenz für das Logo sowie ein Formularelement mit einem Eingabefeld und einem Absende-But-

ton. Im Kopfbereich, also im div-Bereich mit dem id-Namen `top_section`, werden die Elemente notiert:

```
<div id="top_section">
   <img src="logo.gif" id="logo" alt="Logo">
   <form id="search" action="search.php" method="get">
      <span id="search_label">Suche nach:</span>
      <input type="text" id="search_text" name="search_text"
             value="Suche">
      <input type="submit" src="ok.gif" id="search_button"
             value="OK">
   </form>
</div>
```

Um die Elemente wunschgemäß zu positionieren, erhalten sie id-Namen. Das Stylesheet der externen CSS-Datei wird wie folgt erweitert:

```
#logo {
   position:absolute;
   top:5px;
   left:40px;
   width:150px;
   height:152px;
}

#search {
   position:absolute;
   top:60px;
   left:240px;
   white-space:nowrap;
}

#search_label {
   font-family:Verdana,sans-serif;
   font-size:13px;
   font-weight:bold;
   color:rgb(175,35,35);
   vertical-align:middle;
}

#search_text {
   width:240px;
   background-color:rgb(251,241,225);
   border:rgb(175,35,35) solid 1px;
   padding-left:4px;
   padding-right:4px;
   padding-top:2px;
   padding-bottom:2px;
   font-family:Verdana,sans-serif;
   font-size:13px;
   font-weight:bold;
   color:black;
```

```
    vertical-align:middle;
}

#search_button {
    background-color:rgb(175,35,35);
    border:none;
    padding-left:4px;
    padding-right:4px;
    padding-top:2px;
    padding-bottom:2px;
    font-family:Verdana,sans-serif;
    font-size:13px;
    font-weight:bold;
    color:rgb(251,241,225);
    vertical-align:middle;
}
```

Die Logografik (id-Name `logo`) und das Suchformular (id-Name `search`) werden absolut positioniert. Bezug für die Oben-links-Definitionen ist die obere linke Ecke des Anzeigefensters, da sich die Elemente nicht innerhalb eines anderen positionierten Elements befinden.

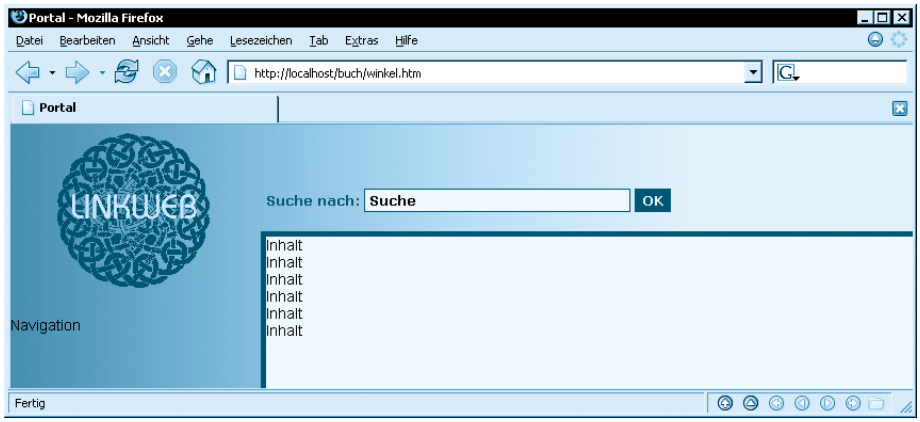


Abbildung 5.14: Kopfbereich mit Logo und Suchformular

Die Logografik füllt den Winkel und reicht dabei in die linke Leiste hinein. Das Formular ist bündig zum Winkel, den die roten Rahmen bilden und nah zum Inhalt platziert. Der Inhalt ist zum Logotext »Linkweb« mittig ausgerichtet. Dies sind wieder »gedachte Linien«, von denen früher bereits die Rede war.

Die Formularelemente werden mit CSS ansprechend und passend zum Layout formatiert.

Der Raum oberhalb des Suchformulars könnte noch genutzt werden, etwa für einen waagerechten Ticker. Andererseits unterliegen viele Webdesigner dem Zwang, jedes Fleckchen zu füllen. Sofern es dazu keinen zwingenden Grund gibt, ist dies allerdings nicht sehr anwenderfreundlich. Je mehr unterschiedliche Bereiche auf einer Seite nach Auf-

merksamkeit heischen, desto eher wird der Betrachter überfordert. Er gerät in den Stress, etwas zu verpassen.

Schritt 4: CSS-Definitionen für Navigation und Elemente des Inhaltsbereichs

Für die Navigation basteln wir wie schon im Beispiel des Portallayouts zunächst eine Dummy-Grafik, die einer späteren Navigation ähnelt. In HTML wird die Grafik in den div-Bereich `navigation` eingebettet:

```
<div id="navigation">
   <img id="navigation_pic" src="navi.gif" alt="Navigation">
</div>
```

Die Grafik erhält den id-Namen `navigation_pic`. Im Stylesheet werden ihre genaue Lage und ihre Ausmaße angegeben:

```
#navigation_pic {
   margin-top:0px;
   margin-left:25px;
   width:190px;
   height:281px;
}
```

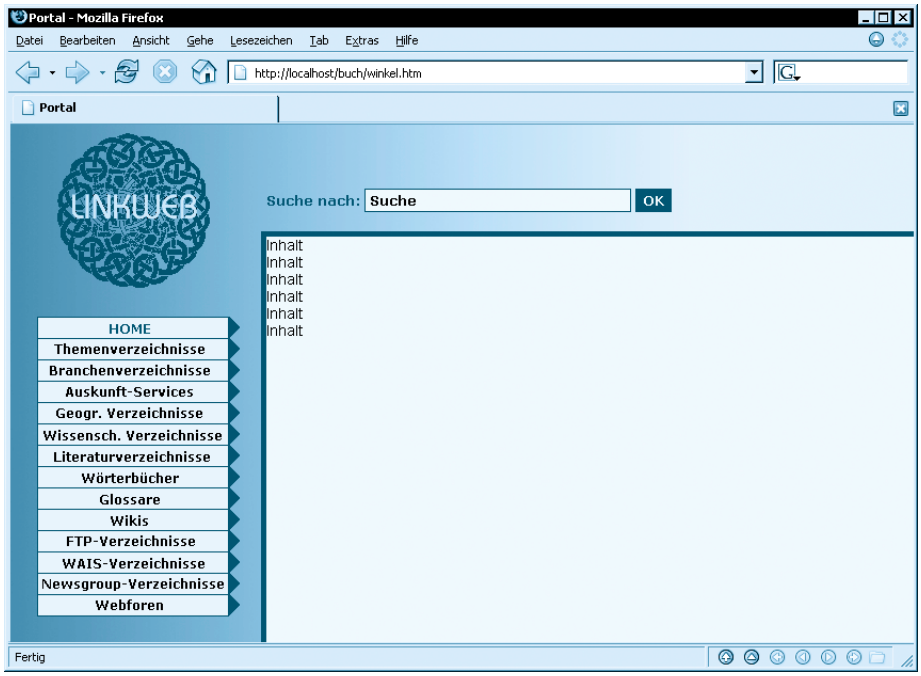


Abbildung 5.15: Winkellayout mit Grafik zur Demo einer möglichen Navigation

Die inhaltstragenden Elemente werden wie folgt definiert:

```
h1,h2,p,ul,ol {
   font-family:Verdana,sans-serif;
   margin-right:20px;
   margin-left:20px;
}

h1 {
   font-weight:normal;
   font-style:normal;
   font-size:17px;
   line-height:1.1em;
   color:rgb(175,35,35);
   border-bottom:rgb(175,35,35) solid 1px;
   padding-bottom:5px;
   padding-left:25px;
   margin-top:20px;
}

h2 {
   font-weight:bold;
   font-style:normal;
   font-size:13px;
   line-height:1.1em;
   color:rgb(175,35,35);
   border-bottom:rgb(175,35,35) solid 1px;
   padding-bottom:3px;
   padding-left:25px;
   margin-top:20px;
}

p {
   font-weight:normal;
   font-size:13px;
   line-height:1.1em;
   padding-left:25px;
   color:black;
}

ul,ol {
   padding-left:25px;
}

li {
   font-weight:normal;
   font-size:13px;
   line-height:1.1em;
   color:black;
   margin-top:6px;
   margin-bottom:6px;
}
```

```
a:link {
   font-weight:bold;
   color:black;
   text-decoration:underline;
}

a:visited {
   font-weight:bold;
   color:gray;
   text-decoration:underline;
}

a:hover, a:active {
   font-weight:bold;
   color:rgb(175,35,35);
   text-decoration:underline;
}
```



Abbildung 5.16: Winkellayout mit Inhalt

Auf der gesamten Seite wird – mit Ausnahme der Schrift im Logo – Verdana als Grundschrift benutzt. Bei Nichtverfügbarkeit kommt eine andere serifenlose Schrift zum Einsatz. Alle Blockelemente, die direkt innerhalb des div-Bereichs `content` vorkommen werden, erhalten links und rechts einen Rand von 20 Pixel (`margin-right:20px`, `margin-left:20px`). Die eigentlichen Texte von Überschriften, Textabsätzen und Listen erhalten jedoch zusätzlich einen linken Innenabstand von 25 Pixel (`padding-left:25px`). Dies erlaubt den Effekt,

dass die unteren Rahmenlinien für Überschriften 1. und 2. Ordnung (`border-bottom: rgb(175,35,35) solid 1px` bzw. `border-bottom: rgb(175,35,35) solid 1px`) ausgerückt gegenüber dem Text erscheinen. Die unteren Rahmenlinien der Überschriften korrespondieren optisch mit den Linien zwischen den Einträgen der Navigationsleiste.

Für Hyperlinks werden geeignete Pseudoelemente definiert. Links zu noch nicht besuchten Zielen erscheinen wie der Fließtext schwarz (`color:black`), werden jedoch zur Kenntlichmachung wie üblich unterstrichen. Außerdem werden sie fett dargestellt (`font-weight:bold`). Da einige Anwender das Unterstreichen von Links im Browser ausgeschaltet haben, wird es mit `text-decoration:underline` explizit festgelegt. Links zu bereits besuchten Zielen werden in abgeschwächtem Grau (`color:gray`) dargestellt, und Links, die gerade mit der Maus überfahren oder angeklickt werden (`a:hover` und `a:active`), werden im Rot der Rahmenfarben dargestellt (`color: rgb(175,35,35)`).

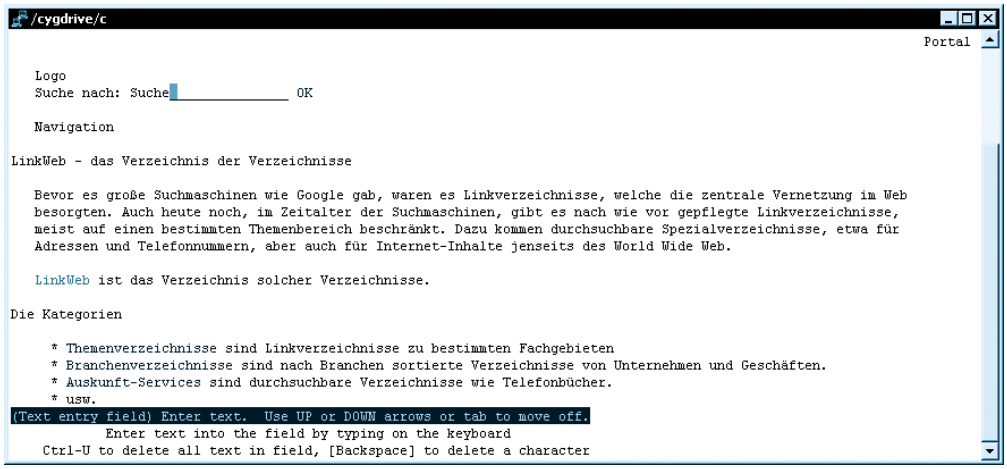


Abbildung 5.17: Winkellayout-Seite im textbasierten Lynx-Browser

Die bereits vom Portallayout her bekannte Gegenprobe gegen den Textbrowser Lynx ergibt ein zufriedenstellendes Ergebnis. Die Seite ist inhaltlich gut nachvollziehbar und sinnvoll strukturiert.

5.1.4 HTML- und CSS-Basis für ein freies Bereichslayout

Im dritten Workshop-Beispiel soll ein Layout erstellt werden, dass vom Aufbau her etwas ungewöhnlicher ist. Wir entscheiden uns für eine »Fenster-Metapher«. Der Inhalt soll innerhalb des Browser-Fensters in einem Bereich erscheinen, der selber nicht scrollt, sondern stattdessen eigene Scrollleisten für übergroße Inhalte haben soll. Der Bereich hat wie bei Anwendungs- oder Dialogfenstern üblich eine Titelleiste, welche den Titel der aktuellen Seite anzeigt. Am unteren Rand soll eine Hot-Link-Navigationsleiste für Links zu häufig verlangten Zielen angebracht werden. Und rechts befindet sich ein ähnlich gestaltetes Extrafenster für die vollständige Navigation. Um die optische Wirkung zu erhöhen, wer-

den wir ein dezentes »Desktop-Bild« einsetzen, das sich über das gesamte Browser-Fenster erstreckt.

Schritt 1: CSS und HTML für das Rohgerüst

Zunächst wählen wir eine Hintergrundgrafik aus. Die Wahl fällt auf das Motiv eines Blitzes, der den Nachthimmel erhellt.



Abbildung 5.18: Grafik für den »Desktop« des geplanten Layouts

Wichtig ist, dass die Grafik nicht zu viel Speicher beansprucht und trotzdem gut aussieht. Als Größe wählen wir 500 × 375 Pixel und speichern sie im JPEG-Format ab, da dieses bei solchen Grafiken am besten komprimiert. Wir erreichen eine akzeptable Dateigröße von 30 Kbyte.

Die Layoutvorgaben speichern wir wieder in einer separaten CSS-Datei. Zunächst werden dort folgende Formate definiert:

Listing 5.6: Rohgerüst der CSS-Datei »styles.css« für freies Layout

```
html, body {
   margin:0px;
   padding:0px;
   width:100%;
   height:100%;
   background-color:black;
```

```
}

#desktop {
   position:absolute;
   top:0px;
   left:0px;
   margin:0px;
   padding:0px;
   width:100%;
   height:100%;
   z-index:1;
}

#desktop_picture {
   width:98%;
   height:97%;
   border:none;
}

#credits {
   position:absolute;
   bottom:0px;
   padding-bottom:2px;
   padding-left:2px;
   font-family:"MS Sans Serif",sans-serif;
   font-size:11px;
   font-weight:normal;
   color:yellow;
}
```

Für die Basiselemente `html` und `body` werden alle Ränder und Abstände auf 0 gesetzt. Breite und Höhe werden explizit auf die Größe des Anzeigefensters gesetzt (100%). Die gesamte Anzeigefläche erhält eine schwarze Hintergrundfarbe.

Als Nächstes werden Formate für ein Element mit dem id-Namen `desktop` definiert. In HTML wird es sich dabei um ein div-Element handeln. Eigentlich könnte man das img-Element zum Referenzieren der Gewittergrafik, welches den id-Namen `desktop_image` erhält, auch direkt absolut positionieren. Doch in der Strict-Variante von HTML dürfen keine Inline-Elemente unmittelbar im Dateikörper stehen, sondern nur Block-Elemente. Daher der umgebende div-Bereich. Dieser wird absolut positioniert und beginnt ganz links oben im Anzeigefenster an Position 0,0. Ränder und Abstände werden auf 0 gesetzt und Breite und Höhe des Anzeigebereichs werden voll ausgenutzt. Die Angabe `z-index:1` ist von Bedeutung, da sich bei diesem Layout mehrere positionierte Bereiche überlagern werden. Durch die Eigenschaft `z-index` wird die gewünschte Schichtenverteilung explizit gesteuert.

Das Bild selbst erhält für Breiten- und Höhenangaben Prozentwerte. Es wird dadurch fast, aber nicht ganz über den gesamten Anzeigebereich gestreckt. Unten und rechts bleibt ein schwarzer Streifen übrig – bedingt durch die schwarze Hintergrundfarbe des body-Elements.

Ein Element mit dem id-Namen `credits` wird schließlich noch absolut positioniert, und zwar am unteren Rand (`bottom:0px`). Es handelt sich um Text, der im unteren schwarzen Streifen erscheinen soll.

Das passende HTML-Gerüst sieht wie folgt aus:

Listing 5.7: HTML-Rohgerüst für freies Layout

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
   <head>
      <meta http-equiv="content-type" content="text/html;
         charset=ISO-8859-1">
      <title>Freies Layout</title>
      <link rel="stylesheet" type="text/css" href="styles.css">
   </head>
   <body>
      <div id="desktop">
         <img id="desktop_picture" src="desktop.jpg"
              alt="Desktop-Bild">
      </div>
      <div id="credits">
         Homepage von Geronimo Fantastico, 2005
      </div>
   </body>
</html>
```

Die externe CSS-Datei wurde unter dem Namen ***styles.css*** gespeichert. Über das übliche `<link>`-Tag wird sie im HTML-Dokument eingebunden.

Im body-Bereich werden die beiden div-Bereiche mit den id-Namen `desktop` und `credits` notiert. Innerhalb des desktop-Bereichs wird das img-Element für die »Desktopgrafik« notiert. Sie erhält den für die CSS-Formatdefinitionen passenden id-Namen `desktop_picture`.

Im Bereich mit dem id-Namen `credits` wird ein geeigneter Text notiert. An dieser Stelle könnten natürlich auch spannendere Inhalte stehen, wie etwa ein wechselnder Spruch zum Tage oder ein Hinweis, ob der Homepage-Besitzer über ICQ gerade online erreichbar ist oder nicht.

Im Browser sieht das Ergebnis nun schon recht attraktiv aus. Beim Verändern der Fenstergröße bleibt das Layout erhalten. Die Grafik wird vom Browser entsprechend gezoomt.

Zur Anzeigequalität der Grafik ist zu bemerken, dass diese stark von den Fähigkeiten des Browsers abhängt, eine Grafik nichtlinear zu strecken. Die Grafik selbst ist zwar im Seitenverhältnis 4:3 abgespeichert worden, was auch dem Verhältnis von typischen Bildschirmauflösungen wie 1024 × 768 Pixel entspricht. Doch letztlich bestimmen viele Faktoren die Größe des Anzeigefensters im Browser und wir verzichten darauf, dem Anwender irgendeine Größe dieses Fenster aufzwingen zu wollen. Da die Grafik aber noch vom

inhaltstragenden »Fenster« und dem für die Navigation überdeckt wird, spielen kleine Auflösungsgrobheiten bei der Grafik letztlich keine so große Rolle.

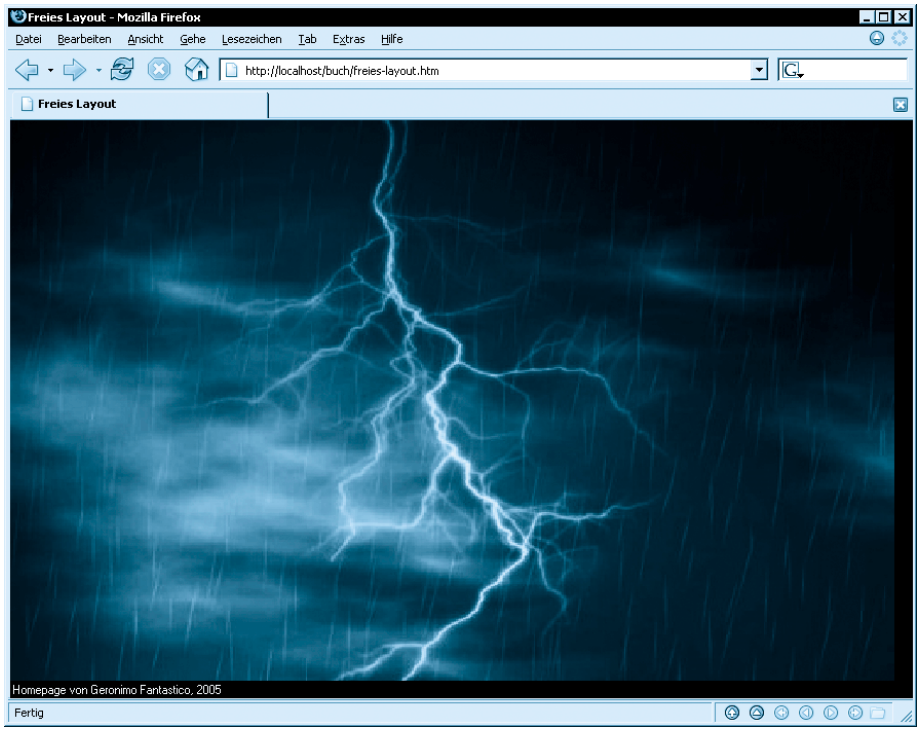


Abbildung 5.19: Rohgerüst des freien Layouts

Schritt 2: CSS-Definitionen für das Inhalts- und das Navigationsfenster

In der CSS-Datei werden folgende Formate für die beiden »Fenster« definiert:

```
#main_window {
   position:absolute;
   background-color:rgb(157,196,250);
   top:10%;
   left:10%;
   width:50%;
   height:70%;
   z-index:3;
   opacity:0.65;
   filter:alpha(opacity=65)
}

#main_window_title {
   background-color:rgb(0,45,162);
   width:100%;
   border-right:solid 8px black;
}
```

```
#main_window_content {
   width:100%;
   height:100%;
   border-bottom:solid 8px black;
   border-right:solid 8px black;
   overflow:scroll;
}

#nav_window {
   position:absolute;
   background-color:rgb(67,110,150);
   top:10%;
   left:63%;
   width:24%;
   z-index:3;
   height:70%;
   opacity:0.45;
   filter:alpha(opacity=45);
}

#nav_window_title {
   background-color:olive;
   width:100%;
   border-right:solid 8px black;
}

#nav_window_content {
   width:100%;
   height:100%;
   border-bottom:solid 8px black;
   border-right:solid 8px black;
   overflow:scroll;
}

h1#main, h1#nav {
   font-family:Arial,sans-serif;
   font-size:13px;
   font-weight:bold;
   color:white;
   margin:0;
   padding:0;
   padding-left:2px;
}
```

Die »Fenster« werden grundsätzlich durch die Bereiche mit den Namen `main_window` für den Inhalt und `nav_window` für die Navigation definiert. Diese werden absolut positioniert, jedoch ausschließlich mithilfe von Prozentangaben. Die beiden Bereiche für die Fenster werden dadurch so im Anzeigefenster ausgerichtet, dass sie dort insgesamt einigermaßen zentriert und mittig erscheinen. Beim Verändern der Größe des Browser-Fensters werden die Größen der Fensterbereiche automatisch angepasst.

Als besonderer Effekt wird bei beiden Bereichen mit Transparenz gearbeitet. Da die CSS-3.0-Eigenschaft `opacity` im Internet Explorer noch nicht funktioniert, dagegen jedoch die hauseigene Angabe zum Alpha-Filter (`filter:alpha(...)`), werden im Beispiel beide Eigenschaften notiert. Eine Hintergrundfarbe (`background-color`) wird ebenfalls angegeben. Dadurch wird auch sichergestellt, dass Browser, die keine der beiden Angaben zur Transparenz interpretieren (wie z.B. Opera bis einschließlich Version 7), zumindest die Hintergrundfarbe anzeigen, deren Farbwert sich am Hintergrundbild orientiert.

Wichtig ist auch die Angabe `z-index:3` bei den Definitionen zu `main_window` und `nav_window`. Nur durch den höheren z-index-Wert erscheinen beide Bereiche über dem zuvor definierten Bereich `desktop`.

Weitere Bereiche mit den Namen `main_window_title`, `main_window_content`, `nav_window_title` und `nav_window_content` definieren innere Bereiche von `main_window` bzw. `nav_window`. Diese Bereiche werden nicht positioniert, sondern bestimmen den Textfluss innerhalb ihrer Elternbereiche. Durch eine dunklere Hintergrundfarbe werden die Titelzeilenbereiche von den Inhaltsbereichen optisch getrennt. Die beiden Bereiche für Inhalte (`main_window_content` und `nav_window_content`) sollen gescrollt werden können, wenn ihr Inhalt die definierten Werte für Höhe und Breite sprengt. Dies besorgt die Definition `overflow:scroll`. Schließlich erhalten die Bereiche noch je eine Angabe zum Rahmen rechts und beim Inhalt auch unten.

Innerhalb der Fenstertitelzeilen sollen aus struktureller Sicht Überschriften stehen. Daher werden zwei spezielle Formate für Überschriften 1. Ordnung definiert, die sich optisch als Titelleisteneintrag eignen, aber logisch gesehen die Funktion zentraler Überschriften erfüllen.

Der body-Bereich des HTML-Dokuments wird um die entsprechenden Elemente und ihre gedachte Struktur erweitert. Der folgende Quelltextausschnitt zeigt den gesamten Inhalt zwischen `<body>` und `</body>`:

```
<div id="desktop">
   <img id="desktop_picture" src="desktop.jpg" alt="Desktop-Bild">
</div>
<div id="main_window">
   <div id="main_window_title"><h1 id="main">Seitentitel</h1></div>
   <div id="main_window_content">
      Inhalt
   </div>
</div>
<div id="nav_window">
   <div id="nav_window_title"><h1 id="nav">Menü</h1></div>
   <div id="nav_window_content">
      Navigation
   </div>
</div>
<div id="credits">
   Homepage von Geronimo Fantastico, 2005
</div>
```

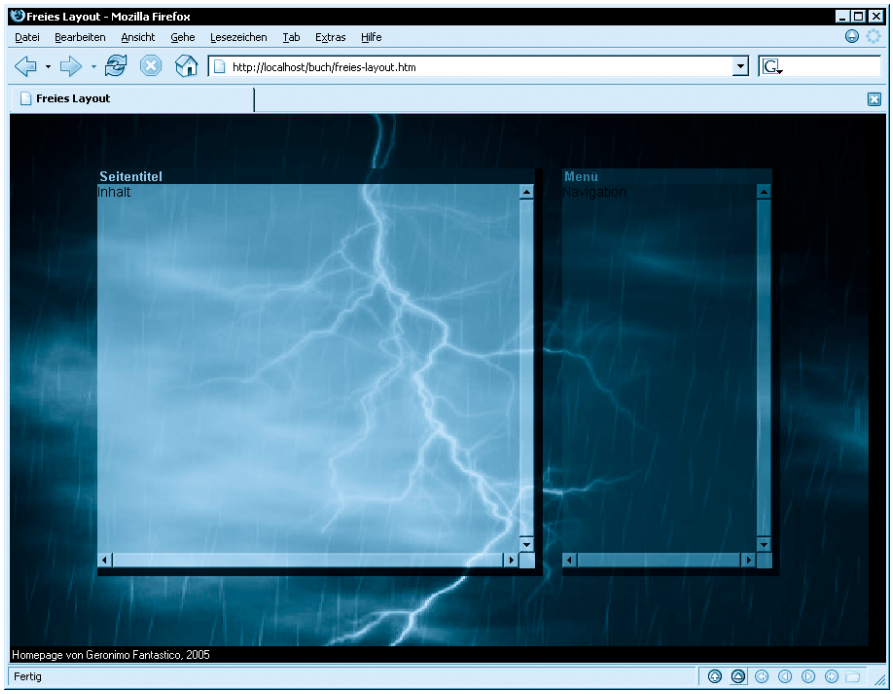


Abbildung 5.20: Transparente »Fenster« für Inhalt und Navigation im freien Layout

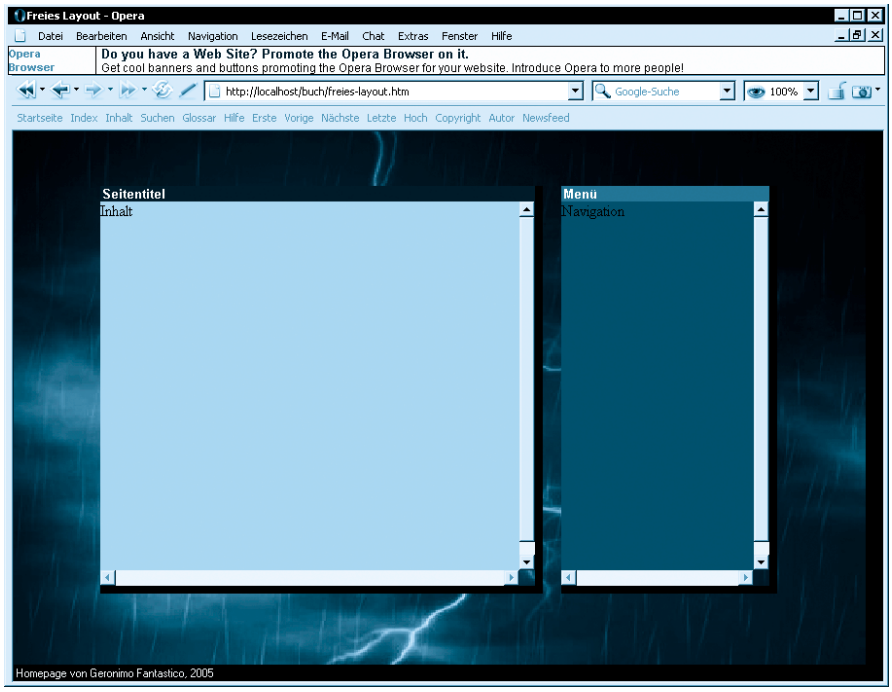


Abbildung 5.21: Beispiellayout im Browser ohne Transparenz (hier: Opera)

Die div-Struktur aus `main_window` mit den Unterbereichen `main_window_title` und `main_window_content` sowie aus `nav_window` mit `nav_window_title` und `nav_window_content` ist gut erkennbar und in ihrer Komplexität noch überschaubar.

Schritt 3: CSS-Definitionen für Inhalt und Navigation

Für die Navigation erstellen wir wie auch in den beiden zuvor behandelten Layouts eine Grafik, die eine Navigationsleiste simuliert. Für die Textinhalte im Bereich `main_window_content` werden im Stylesheet folgende Formate festgelegt:

```
h2, p, ul, ol, table {
   margin-left:10px;
   margin-right:10px;
}

h2, p, li, th, td {
   font-family:Arial,sans-serif;
   color:black;
   font-weight:bold;
}

h2 {
   font-size:125%;
   margin-top:10px;
   border-bottom:thin solid black;
}

p, li, th, td {
   font-size:90%;
}
```

Alle zu erwartenden inhaltstragenden Blockelemente erhalten Randabstände links und rechts von je 10 Pixel. Dies ist der Abstand zum Rand des simulierten Fensters. Auffällig ist, dass für alle Elemente eine fette Schriftart vorgesehen ist. Mit Arial wird außerdem eine Schriftart gewählt, die alles andere als exotisch ist, dafür aber wohl mit am besten lesbar. Ungewöhnliches bei der Schriftformatierung ist auch gar nicht mehr nötig – das Layout selbst bietet Individualität genug. Stattdessen steht bei der Schriftformatierung die Lesbarkeit auf dem farbigen Hintergrund mit transparent durchscheinendem Desktopbild im Vordergrund.

Überschriften werden nur in der 2. Ordnung benötigt. Diese erhalten eine schwarze Rahmenlinie unterhalb. Die Schriftgröße von 125% orientiert sich an der Schriftgröße des Elternelements. Da dieses ein div-Element ist und div-Elemente im Browser per Voreinstellung die Schriftgröße von normalem Fließtext haben, erscheinen h2-Überschriften also in einer Schriftgröße von 125%, gemessen an normalem Fließtext. Der normale Fließtext für Elemente wie Textabsätze, Listenpunkte oder Tabellenzellen wird auf 90% gesetzt. Das ist etwas kleiner als normal, sollte aber immer noch gut lesbar sein, sofern der Anwender keine extrem kleine Schriftgröße als Default-Schriftgröße eingestellt hat.

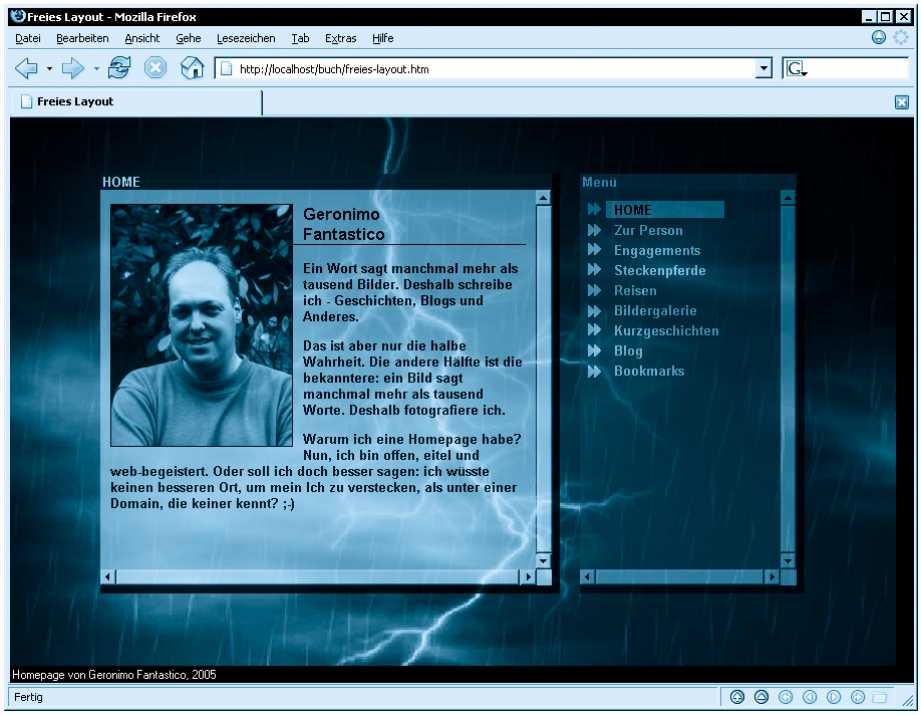


Abbildung 5.22: Freies Layout mit Inhalt

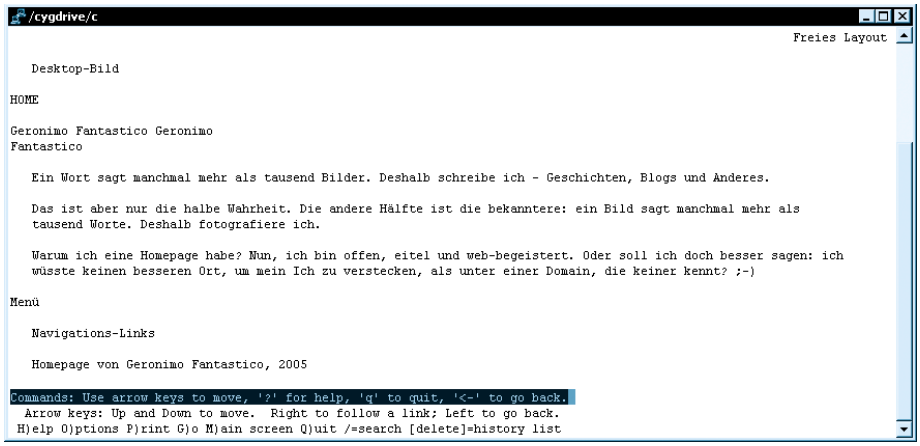


Abbildung 5.23: Seite mit freiem Layout im textbasierten Lynx-Browser

Alles in allem ist unser »freies Layout«, das sich in dieser Form eher für private Homepages oder vergleichbare Auftritte eignet, eine Mischung aus gewagten Mitteln, die mit einigen älteren Browsern gar nicht gut aussehen, und Rücksichtnahme auf Usability und unterschiedliche Umgebungsbedingungen und Ausgabemedien. Auch der Check mit dem textbasierten Lynx-Browser zeigt ein strukturell sinnvolles Erscheinungsbild. Dank der absoluten Positionierung wäre es übrigens ebenso möglich, die Navigation im HTML-

Quelltext oberhalb des Inhalts zu notieren. Bei einem Browser wie Lynx würde dann zuerst die Navigation und anschließend der Textinhalt angezeigt.

5.2 Navigations- und Menüleisten mit CSS

Anhand der drei Beispiellayouts wurde gezeigt, wie Seitenlayouts entstehen und welche Mittel dabei zum Einsatz kommen. In allen drei Beispielen wurde jedoch die Navigation nur als Dummy-Grafik dargestellt. Da elegante Link-Formatierung ein Thema für sich ist, wird es in diesem Abschnitt gesondert behandelt.

Vor einigen Jahren musste jedes Menü, das beispielsweise so genannte *Rollover-Effekte* (Effekt wie beim Auswählen aus einer Liste oder einem Menü) verwenden wollte, auf Grafiken für jeden Link und auf ein JavaScript zum Austauschen von Grafiken zurückgreifen. Dies ist dank CSS nicht mehr nötig. Scheinbar unauffällige CSS-Eigenschaften, wie das Umwandeln von Inline- in Block-Elemente oder die Pseudo-Elemente für Links in unterschiedlichen Zuständen, bilden die Basis für attraktive, CSS-basierte Navigationsleisten.

5.2.1 Vertikale Navigationsleiste mit Rollover-Effekt

In Kapitel 4 haben wir ein Beispiel für den Einsatz einer Hintergrundgrafik vorgestellt, das wir nun wieder aufgreifen möchten:

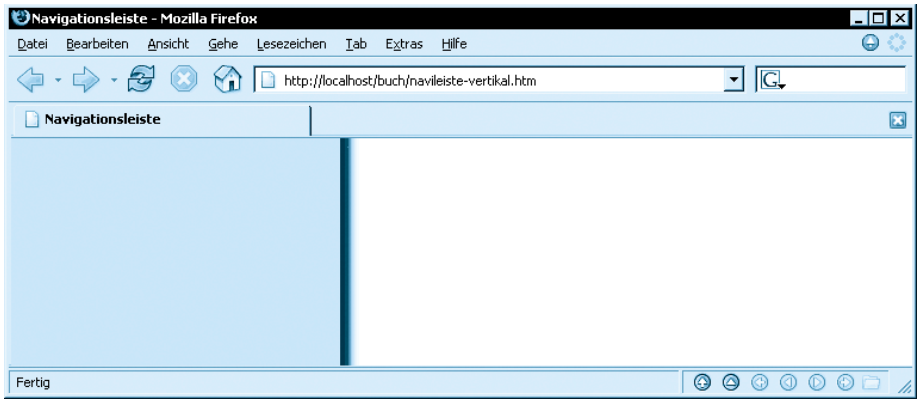


Abbildung 5.24: Seitenlayout mit vorgesehenem Bereich für die Navigationsleiste

Die Hintergrundgrafik, die in diesem Beispiel dem body-Element zugewiesen und mit `background-repeat:repeat-y` nur einmal senkrecht wiederholt wird, hat eine Nutzbreite von 280 Pixel. In HTML definieren und in CSS positionieren wir einen passenden Bereich. Der Quelltext der Seite lautet:

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

```
    <head>
       <meta http-equiv="content-type" content="text/html;
          charset=ISO-8859-1">
       <title>Navigationsleiste</title>
       <style type="text/css">
          body {
             background-color:white;
             background-image:url(wallpaper.gif);
             background-repeat:repeat-y;
          }
          #navigation {
             position:absolute;
             top:40px;
             left:30px;
             width:220px;
          }
       </style>
    </head>
    <body>
       <div id="navigation">
       </div>
    </body>
</html>
```

Im Stylesheet wird dem body-Element mit `background-image` die gewünschte Hintergrundgrafik zugewiesen. Die 1-Spalten-Wiederholung wird zugewiesen und als Hintergrundfarbe für die übrige Seite wird Weiß bestimmt. Ferner wird ein Element mit dem id-Namen `navigation` absolut positioniert, und zwar so, dass der Bereich in der Mitte der Hintergrundgrafik angezeigt wird. Da die farbige Fläche ohne Schatten eine Breite von 280 Pixel hat, wird durch einen linken Randabstand von 30 Pixel und einer Breite von 220 Pixel erreicht, dass sich optisch gesehen rechts ebenfalls ein Rand von 30 Pixel bis zur rechten Grenze des farbigen Bereichs ergibt.

Zwischen `<body>` und `</body>` wird ein div-Element mit dem id-Namen `navigation` notiert. Dadurch wird dieses Element wie im Stylesheet angegeben positioniert. Innerhalb des div-Bereichs sollen die Links der Navigationsleiste notiert werden.

Schritt 3: HTML-Quelltext der Navigationslinks

Die Navigationslinks werden innerhalb des div-Bereichs mit dem id-Namen `navigation` als gewöhnliche a-Elemente notiert, und zwar einfach in der gewünschten Reihenfolge, ohne andere Formatierungen. In unserem Beispiel lautet die Liste der Links:

```
<a class="navi_self navi_first" href="schlagzeigen.html">
   HOME
</a>
<a class="navi" href="tipps.html" title="Aktuelle Reisetipps">
   Aktuelle Reisetipps
</a>
<a class="navi" href="bildung.html" title="Bildungsreisen">
```

```
   Bildungsreisen
</a>
<a class="navi" href="abenteuer.html" title="Abenteuerreisen">
   Abenteuerreisen
</a>
<a class="navi" href="pilger.html" title="Pilgerreisen">
   Pilgerreisen
</a>
<a class="navi" href="kreuzfahrten.html" title="Kreuzfahrten">
   Kreuzfahrten
</a>
<a class="navi" href="offroad.html" title="Offroad-Touren">
   Offroad-Touren
</a>
<a class="navi" href="fahrrad.html" title="Fahrrad-Touren">
   Fahrrad-Touren
</a>
<a class="navi" href="wanderungen.html" title="Wanderungen">
   Wanderungen
</a>
<a class="navi" href="familien.html" title="Familienparks">
   Familienparks
</a>
<a class="navi" href="kur.html" title="Kurhotels">
   Kurhotels
</a>
<a class="navi" href="flug.html" title="Tipps für Flugreisen">
   Tipps für Flugreisen
</a>
<a class="navi" href="auto.html" title="Tipps für Autoreisen">
   Tipps für Autoreisen
</a>
<a class="navi" href="veranstalter.html" title="Reiseveranstalter">
   Reiseveranstalter
</a>
<a class="navi" href="versich.html" title="Reiseversicherungen">
   Reiseversicherungen
</a>
<a class="navi" href="rechner.html" title="Währungsrechner">
   Währungsrechner
</a>
<a class="navi" href="ferien05.html" title="Ferienübersicht 2005">
   Ferienübersicht 2005
</a>
<a class="navi" href="impressum.html" title="Impressum">
   Impressum
</a>
<a class="navi" href="feedback.html" title="Feedback">
   Feedback
</a>
```

Alle Links bekommen neben dem href-Attribut, welches das Verweisziel bestimmt, über das class-Attribut einen Klassennamen zugewiesen. Es gibt insgesamt drei Klassen von Navigationslinks: Der erste Link der Liste muss aus optisch-technischen Gründen eine eigene Klasse haben. Diesem Link wird daher die Klasse `navi_first` zugewiesen. Die aktuell angezeigte Seite soll ebenfalls optisch abweichend dargestellt werden. So kann der Anwender in der Navigationsleiste stets sehen, auf welcher Seite er sich gerade befindet. Da unsere Beispielseite die Startseite ist und der erste Link zur Startseite führt, ist es wieder der erste Link, dem die Klasse `navi_self` zur Kenntlichmachung der aktuellen Seite zugewiesen wird. Wie erkennbar, ist es kein Problem, einem Element zwei CSS-Klassen zuzuweisen. Die gewünschten Klassennamen werden in der Wertzuweisung an das class-Attribut einfach durch Leerzeichen getrennt. Alle übrigen Links im Beispiel haben die Standard-Linkklasse `navi`.

Alle Links erhalten außerdem ein title-Attribut, das den gleichen Wert enthält wie der jeweils anklickbare Verweistext. Der Zweck des title-Attributs ist, bei Links, deren Text wegen Überlänge abgeschnitten wird, über die Tooltipp-Fenster, die vom title-Attribut erzeugt werden, doch noch den gesamten Linktext sehen zu können.

Damit aus der Liste von Links optisch gesehen eine Navigationsleiste wird, müssen entsprechende zentrale CSS-Formatdefinitionen her. In unserem Beispiel lauten sie:

```
a.navi, a.navi_first, a.navi_self {
   display:block;
   white-space:nowrap;
   overflow:hidden;
   border-left:1px solid black;
   border-right:1px solid black;
   border-bottom:1px solid black;
   font-family:Tahoma,sans-serif;
   font-size:12px;
   font-weight:bold;
   text-decoration:none;
   letter-spacing:0.1em;
   padding:3px;
   text-align:center;
}
a.navi_first {
   border-top:1px solid black;
}
a.navi:link, a.navi_first:link,
a.navi:visited, a.navi_first:visited {
   background-color:rgb(115,115,75);
   color:white;
}
a.navi_self:link, a.navi_self:visited {
   background-color:gray;
   color:white;
}
a.navi:hover, a.navi_first:hover, a.navi_self:hover,
a.navi:active, a.navi_first:active, a.navi_self:active {
```

```
    background-color:white;
    color:black;
}
```

Zunächst werden Formateigenschaften definiert, die für alle drei erwähnten Linkklassen gelten, also `navi`, `navi_first` und `navi_self`. Gleich die erste Definition ist ein wichtiger Schritt: `display:block` sorgt dafür, dass die a-Elemente vom Browser wie Blockelemente behandelt werden. Jeder Link erzeugt damit eine neue Zeile im Textfluss und die Elementbox jedes Links erstreckt sich über die gesamte zur Verfügung stehende Breite. Diese wird vom Elternelement, dem div-Bereich `navigation`, vorgegeben und beträgt 220 Pixel.

Die Angaben zu `white-space` und `overflow` stellen sicher, dass der Linktext nicht umgebrochen werden kann (`nowrap`) und dass er, falls er breiter ist als 220 Pixel, einfach abgeschnitten wird (`hidden`). Beide Angaben tragen also dazu bei, dass das Layout der Navigationsleiste nicht »zerschossen« wird. Für eventuell abgeschnittene Linktexte haben wir ja in HTML die title-Attribute notiert.

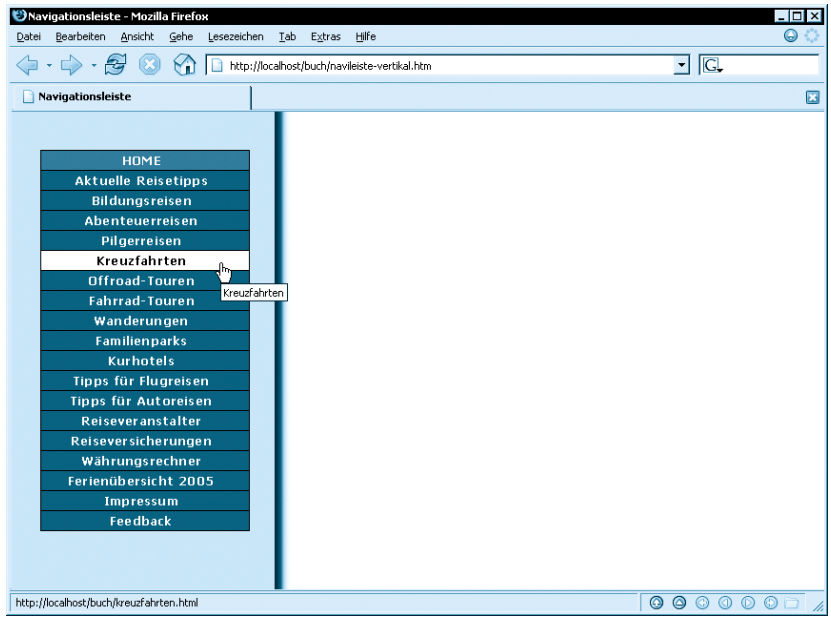


Abbildung 5.25: Vertikale Navigationsleiste mit Rollover-Effekt

Links aller drei Klassen erhalten links, rechts und unten einen dünnen schwarzen Rahmen und sollen den Linktext in weißer, fetter Schrift in der Schriftart Tahoma oder irgendeiner anderen serifenlosen Schrift bei leicht erweitertem Zeichenabstand anzeigen. Die Innenabstände zu den Grenzen der Elementbox soll in allen Fällen 3 Pixel betragen. Der Linktext soll zentriert ausgerichtet werden.

Der oberste Link (`class="navi_first"`) erhält auch oben eine dünne schwarze Rahmenlinie. Da die Links abstandslos aneinander anschließen, ist die untere Rahmenlinie

zugleich die obere des nachfolgenden Links, weshalb Letztere bei den Links der anderen Linkklassen auch nicht definiert ist.

Schließlich werden noch Hintergrundfarbe und Textfarbe festgelegt. Dabei wird zwischen den möglichen Zuständen der Links unterschieden, also zwischen `:link` (Verweis zu noch nicht besuchter Seite), `:visited` (Verweis zu bereits besuchter Seite), `:hover` (Verweis, der gerade über Maus oder Tastatur den Fokus erhält) und `:active` (Verweis, der gerade angeklickt bzw. ausgeführt wird). Zwischen Links zu besuchten und noch nicht besuchten Seiten wird in unserer Navigationsleiste optisch nicht unterschieden. Für die beiden Klassen `navi` und `navi_first` sind jedoch zwei getrennte Selektoren erforderlich, da beide unterschiedliche Hintergrundfarben erhalten. Zwischen den Zuständen `:hover` und `:active` wird optisch auch nicht mehr weiter unterschieden. Alle drei Linkklassen erscheinen in diesen beiden Zuständen mit schwarzem Text auf weißem Hintergrund.

Beim Austesten der Funktionalität mit verschiedenen Browsern fällt auf, dass es beim Internet Explorer erforderlich ist, auf den Linktext zu positionieren, damit der Hover-Effekt eintritt. Bei anderen Browsern (Firefox, Opera) genügt das Positionieren in die Elementbox eines Links. Neuere Browser als der Internet Explorer 6 unterstützen Pseudoelemente wie `:hover` (oder `:focus`) auch für andere Elemente als das a-Element.

Andere Effekte

Der Phantasie für CSS-Definitionen bei den einzelnen Linkzuständen sind kaum Grenzen gesetzt. In unserem Beispiel haben wir für `:hover` und `:active` lediglich die Hintergrundfarbe und die Textfarbe geändert, was einen klassischen Rollover-Effekt bewirkt. Ebenso können Sie aber auch mit anderen Effekten arbeiten. So bewirkt beispielsweise das Ändern von `border:outset` zu `border:inset` einen realistischen Button-Eindrück-Effekt.

5.2.2 Horizontale Navigationsleiste mit Rollover-Effekt

In diesem Beispiel soll folgende obere Navigationsleiste realisiert werden:

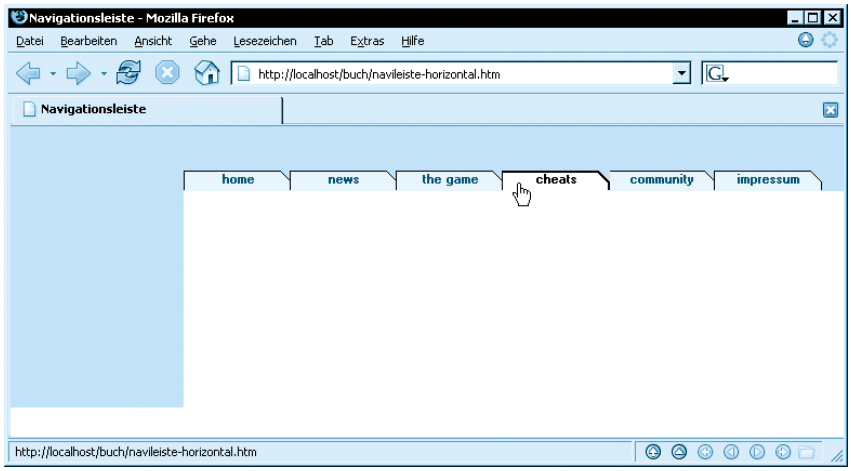


Abbildung 5.26: Horizontale Navigationsleiste mit Rollover-Effekt

Wie aus der Abbildung ersichtlich, wird bei dem Beispiel mit einem Karteireiter-Effekt gearbeitet. Um den Effekt zu erzielen, haben wir folgende zwei Grafiken erstellt, die wir als Hintergrundgrafiken einsetzen:

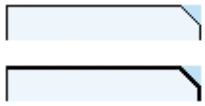


Abbildung 5.27: Grafiken für den Einsatz als Hintergrundgrafik

Beide Grafiken messen 100 × 18 Pixel. Die Grafiken wurden im GIF-Format als *reiter1.gif* und *reiter2.gif* abgespeichert. Die gelblich-beige Innenfarbe wurde in beiden Grafiken als transparent definiert.

Die zuvor abgebildete Seite im Browser hat folgenden Quelltext (Komplettlisting mit HTML und CSS-Definitionen):

Listing 5.8: HTML- und CSS-Quelltext für Seite mit horizontaler Navigationsleiste

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
   <head>
      <meta http-equiv="content-type" content="text/html;
         charset=ISO-8859-1">
      <title>Navigationsleiste</title>
      <style type="text/css">
         html, body {
            margin:0;
            padding:0;
            background-color:white;
         }
         #top_navigation {
            position:absolute;
            top:0px;
            left:0px;
            width:100%;
            height:60px;
            background-color:rgb(230,180,250);
            z-index:1;
         }
         #left_navigation {
            position:absolute;
            top:60px;
            left:0px;
            width:160px;
            min-height:200px;
            background-color:rgb(230,180,250);
            z-index:2;
         }
         a.top_link {
```

```
    display:block;
    position:absolute;
    width:100px;
    text-align:center;
    height:18px;
    font-family:"MS Sans Serif",sans-serif;
    font-size:12px;
    font-weight:bold;
    padding-top:1px;
    text-decoration:none;
}
a.top_link:link, a.top_link:visited {
    color:rgb(150,50,170);
    background-image:url(reiter1.gif);
    background-position:top right;
    background-repeat:no-repeat;
    background-color:rgb(245,240,205);
}
a.top_link:hover, a.top_link:active {
    color:black;
    background-image:url(reiter2.gif);
    background-position:top right;
    background-repeat:no-repeat;
    background-color:white;
}
a#home {
    bottom:0px;
    left:160px;
    z-index:3;
}
a#news {
    bottom:0px;
    left:258px;
    z-index:4;
}
a#game {
    bottom:0px;
    left:356px;
    z-index:5;
}
a#cheats {
    bottom:0px;
    left:454px;
    z-index:6;
}
a#comm {
    bottom:0px;
    left:552px;
    z-index:7;
}
a#impr {
    bottom:0px;
    left:650px;
```

```
            z-index:8;
         }
         a#home:hover, a#home:active,
         a#news:hover, a#news:active,
         a#game:hover, a#game:active,
         a#cheats:hover, a#cheats:active,
         a#comm:hover, a#comm:active,
         a#impr:hover, a#impr:active {
            z-index:9;
         }
      </style>
   </head>
   <body>
      <div id="top_navigation">
         <a href="index.htm" class="top_link" id="home">
            home</a>
         <a href="news.htm" class="top_link" id="news">
            news</a>
         <a href="game.htm" class="top_link" id="game">
            the game</a>
         <a href="cheats.htm" class="top_link" id="cheats">
            cheats</a>
         <a href="community.htm" class="top_link" id="comm">
            community</a>
         <a href="impressum.htm" class="top_link" id="impr">
            impressum</a>
      </div>
      <div id="left_navigation">
      </div>
   </body>
</html>
```

Im HTML-Part zwischen `<body>` und `</body>` werden zwei div-Bereiche mit den id-Namen `top_navigation` und `left_navigation` notiert. Letzterer dient in unserem Beispiel nur der optischen Vollständigkeit und spielt keine weitere Rolle. Im zentralen Stylesheet, also im HTML-Kopfbereich zwischen `<style>` und `</style>`, werden die beiden div-Bereiche absolut positioniert, und zwar so, dass sie einen Winkel links oben bilden. Beide Bereiche erhalten die gleiche hellviolette Hintergrundfarbe (`background-color: rgb(230,180,250)`).

Alle Links der oberen Navigationsleiste erhalten den Klassennamen `top_link` und zusätzlich individuelle id-Namen. Diese komfortable Ausstattung mit Namen hilft bei den CSS-Definitionen. Gemeinsame Eigenschaften aller Navigationslinks können im Stylesheet über den Selektor `a.top_link` definiert werden. Dazu gehört – wie schon im Beispiel der vertikalen Navigationsleiste – die Umwandlung des a-Elements vom Inline- in ein Blockelement (`display:block`). Da alle Links absolut positioniert werden, wird die entsprechende CSS-Definition ebenfalls schon für die Klasse angegeben, ebenso wie die gemeinsame Breite, die Textausrichtung und die Schriftformatierung des Verweistextes.

Als Nächstes folgen im Stylesheet zwei Formatblöcke, welche das Verhalten der Links in den unterschiedlichen Zuständen steuern. Mit dem aufzählenden Selektor `a.top_link:link`,

`a.top_link:visited` wird festgelegt, wie sich die Links mit dem Klassennamen `top_link` im Normalzustand verhalten, wobei nicht zwischen bereits besuchten (`:visited`) und noch nicht besuchten (`:link`) Linkzielen unterschieden wird. Links in diesen Zuständen sollen eine dunkelviolette Schriftfarbe haben (`color:rgb(150,50,170)`) auf gelblich-beigem Hintergrund (`background-color:rgb(245,240,205)`) haben. Ferner wird als Hintergrundgrafik (`background-image`) die Datei *reiter1.gif* angegeben. Das Hintergrundbild soll nur einmalig angezeigt werden (`background-repeat:no-repeat`).

Der Selektor `a.top_link:hover, a.top_link:active` spricht dagegen die Links in den Zuständen »fokussiert« (`:hover`) und »aktiviert« (`:active`) an. In diesen Fällen soll die Textfarbe auf Schwarz und die Hintergrundfarbe auf Weiß wechseln. Als Hintergrundbild soll jetzt *reiter2.gif* angezeigt werden.

Dass übrigens die Wechsel von Text- und Hintergrundfarbe überhaupt wirksam sind, liegt daran, dass der Hauptbereich in den Grafiken *reiter1.gif* und *reiter2.gif* als transparent definiert wurde. »Effektiv« sind also in beiden Grafiken nur die Rahmenlinien und die obere hellviolette Ecke.

Damit der Überlappungseffekt im Karteireiterstil zustande kommt, sind allerdings noch weitere Maßnahmen nötig. Diese betreffen die Links individuell. Für jeden Link-id-Namen werden im Stylesheet deshalb verweisspezifische Formate definiert, z. B. mit dem Selektor `a#home` für den Link mit dem id-Namen `home`.

Alle Links werden ja absolut positioniert. Bei den individuellen Formaten wird angegeben, wo genau. Definiert werden ein Linkswert (`left`) und ein Untenwert (`bottom`). Da die Links sich innerhalb des absolut positionierten div-Bereichs `top_navigation` befinden, sind dessen Elementgrenzen der Bezugspunkt für die Angaben. Der Bereich `top_navigation` beginnt oben bei 0 Pixel und hat eine definierte Höhe von 60 Pixel. Das heißt, dass `bottom:0px` so viel bedeutet wie 60 Pixel vom oberen Rand des Anzeigefensters gemessen.

Alle Links erhalten außerdem einen z-Index-Wert zum Steuern von Überlappungseffekten. Im Normalzustand überlappen weiter rechts stehende Links ihre linken Nachbarn um 2 Pixel, da die left-Werte der Links sich jeweils um 98 Pixel unterscheiden, die Linkbreite jedoch 100 Pixel beträgt. Mit dem letzten Selektor, der Aufzählung aller einzelnen Links für die Zustände `:hover` und `:active`, wird der z-Index-Wert des jeweils betroffenen Links so hoch gesetzt, dass der Link seinen rechten Nachbarn auf jeden Fall überlappt.

5.2.3 Horizontale Navigationsleiste mit Ausklappmenüs

In einer einfachen horizontalen Navigationsleiste lassen sich naturgemäß nur einige wenige Links unterbringen. Dennoch finden einige Seitenanbieter und Seitenbesucher Gefallen daran, die gesamte Navigation über solch eine horizontale Navigationsleiste abzuwickeln. Die Leiste – so der Wunsch – soll wie die Menüzeile einer typischen Windows-Anwendung temporär aufklappbare Menüs anbieten.

Ganz ohne ein Minimum an JavaScript kommt man dabei nicht aus. Doch der JavaScript-Aufwand lässt sich dank CSS so stark reduzieren, dass nicht mal ein richtiges »Script« erforderlich ist.

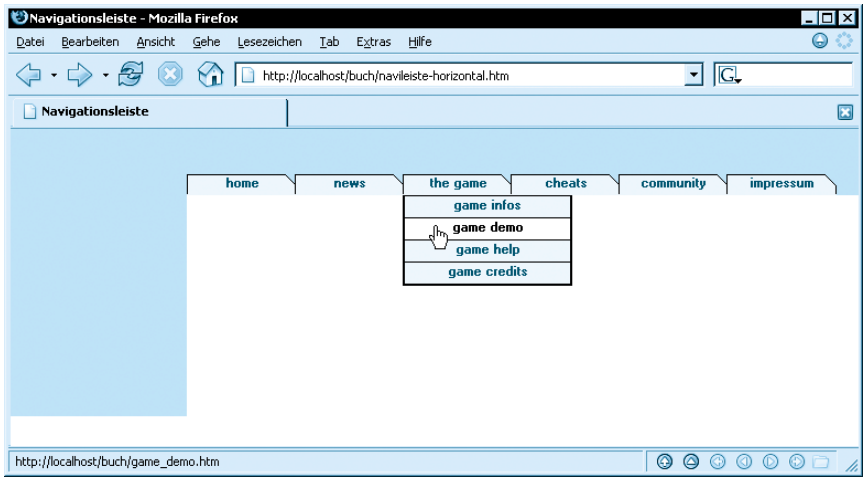


Abbildung 5.28: Horizontale Navigationsleiste mit abhängigem Aufklappmenü

Der Quelltext aus Listing 5.8 im vorhergehenden Abschnitt wird dabei an drei Stellen erweitert.

Im Stylesheet, also zwischen `<style>` und `</style>`, werden die folgenden Definitionen hinzugefügt:

```
#game_sublinks {
   position:absolute;
   top:60px;
   left:356px;
   width:150px;
   border-bottom:2px solid black;
   border-left:2px solid black;
   border-right:2px solid black;
   z-index:10;
   visibility:hidden;
}
.sub_link {
   display:block;
   width:100%;
   text-align:center;
   height:18px;
   font-family:"MS Sans Serif",sans-serif;
   font-size:12px;
   font-weight:bold;
   padding-top:1px;
   text-decoration:none;
   border-top:1px solid black;
}
.sub_link:link, .sub_link:visited {
   background-color:rgb(245,240,205);
   color:rgb(150,50,170);
}
```

```
.sub_link:hover, .sub_link:active {
   background-color:white;
   color:black;
}
```

Der Selektor `#game_sublinks` betrifft den Bereich, der für die Sublinks benötigt wird. Der Bereich wird absolut positioniert, und zwar so, dass er sich unterhalb des zugehörigen Links der Navigationsleiste befindet. Der Bereich erhält unten, links und rechts einen schwarzen, 2 Pixel dicken Rahmen. Sein z-Index-Wert ist hinreichend hoch gewählt, um den Dokumentinhalt, für den beispielsweise ein div-Bereich mit dem z-Index-Wert 1 definiert werden könnte, zu überlappen. Entscheidend ist jedoch die letzte Definition `visibility: hidden`. Dadurch wird verhindert, dass der Bereich per Default angezeigt wird.

Für die untergeordneten Links wird eine Klasse `sub_link` eingerichtet. Mit dem Selektor `.sub_link` werden allgemeingültige Formate für all diese Links definiert. Alle Links sollen wieder als Blockelement behandelt werden und den Bereich, innerhalb dessen sie notiert werden, zu 100% in der Breite ausfüllen. Ferner werden diverse Schriftattribute für den Verweistext bestimmt.

Die Selektoren `.sub_link:link`, `.sub_link:visited` und `.sub_link: hover`, `.sub_link:active` regeln das spezifische Aussehen der Links in den verschiedenen Zuständen. Parallel zur oberen Navigationsleiste werden Hintergrund- und Textfarbe beim Überfahren mit der Maus und beim Anklicken anders dargestellt.

So weit zu den CSS-Anpassungen. Als Nächstes wird im HTML-Teil zwischen `<body>` und `</body>` auf der gleichen Ebene wie die div-Bereiche `top_navigation` und `left_navigation` der neue Bereich mit dem id-Namen `game_sublinks` inklusive der gewünschten Sublinks angelegt:

```
<div id="game_sublinks"
   onMouseover="document.getElementById('game_sublinks').
                style.visibility='visible'"
   onMouseout="document.getElementById('game_sublinks').
               style.visibility='hidden'">
   <a href="game_infos.htm" class="sub_link">game infos</a>
   <a href="game_demo.htm" class="sub_link">game demo</a>
   <a href="game_help.htm" class="sub_link">game help</a>
   <a href="game_credits.htm" class="sub_link">game credits</a>
</div>
```

Die Links erhalten den Klassennamen `sub_link`.

Im einleitenden `<div>`-Tag fallen die beiden HTML-Attribute `onMouseover=` und `onMouseout=` auf. Diese so genannten Event-Handler stellen eine Verbindung zu einer Scriptsprache wie JavaScript her. Die Wertzuweisung ist jeweils eine Programmanweisung. Bei `onMouseover` (der Event bedeutet: beim Überfahren des Elements mit der Maus) wird der Bereich auf »sichtbar« gesetzt; bei `onMouseout` (wenn die Maus den Elementbereich wieder verlässt) wird er wieder auf »unsichtbar« gesetzt.

Schließlich muss noch der HTML-Quelltext des Links mit dem id-Namen `game` wie folgt erweitert werden:

```
<a href="game.htm" class="top_link" id="game"
   onMouseover="document.getElementById('game_sublinks').
                style.visibility='visible'"
   onMouseout="document.getElementById('game_sublinks').
               style.visibility='hidden'">
   the game
</a>
```

Auch hier werden noch mal die gleichen beiden Event-Handler eingefügt, diesmal in das einleitende `<a>`-Tag des Links.

Unser Beispiel hat nur für den Link »the game« ein Aufklappmenü definiert. Wenn Sie für weitere Links der oberen Navigationsleiste solche Menüs wünschen, müssen Sie analog vorgehen, d.h. Bereiche definieren und entsprechend positionieren, und die Links darin notieren. Die Event-Handler müssen ebenfalls bei allen betroffenen Aufklappmenü-Bereichen sowie im jeweiligen Link der Topnavigationsleiste notiert werden. Innerhalb der Klammern von `getElementById(...)` muss dabei jeweils der id-Name des Bereichs für das Aufklappmenü eingesetzt werden.

Beachten Sie jedoch auch, dass Sie durch das Scripting möglicherweise Seitenbesucher von der Möglichkeit ausschließen, alle Seiten aufzurufen. Denn bei Anwendern, die in ihrem Browser JavaScript deaktiviert haben, funktionieren die Aufklappmenüs nicht. Im vorliegenden Beispiel bietet es sich an, auf den Seiten, die bei Anklicken eines Links in der Topnavigationsleiste aufgerufen werden, die Links auf deren Unterseiten im normalen Seiteninhalt nochmals anzubieten.

Auch die Reihenfolge, in der die Links innerhalb des Quelltextes notiert werden, will bedacht sein. Da die Bereiche im Beispiel alle absolut positioniert sind, spielt es keine Rolle, in welcher Reihenfolge ihr Markup im Dokumentquelltext steht. Ein Test mit einem Text-Browser wie Lynx bringt jedoch ans Licht, welche Probleme sich bei einer nicht grafischen oder nicht visuellen Ausgabe des Seiteninhalts ergeben.

Alle CSS-Definitionen in diesem und im vorherigen Abschnitt wurden der Einfachheit halber im zentralen Style-Bereich des HTML-Dokuments notiert. Sollen die Definitionen für mehrere Seiten gültig sein, empfiehlt es sich natürlich, die Definitionen in eine separate CSS-Datei auszulagern.

6 Erweiterte Features von HTML und CSS

Einige Aspekte von HTML und CSS wurden in Kapitel 4 bewusst nicht behandelt, weil es sich um problembehaftete oder speziellere Features handelt. In diesem Kapitel gehen wir gesondert darauf ein.

6.1 Mehrfenstertechnik (Frames)

Als der damals führende Browser-Anbieter Netscape 1995 die Technik der Frames implementierte und dazu eine Reihe nicht standardisierter HTML-Elemente einführte, wurde das neue Feature von der Gemeinde der Webdesigner begeistert aufgenommen. Endlich war es möglich, beispielsweise links eine fixe Navigation zu haben, während sich rechts der scrollbare Inhalt befand. Die Navigation befand sich in einer separaten Datei und musste nicht in jeder Datei neu notiert werden.

So weit ist der Segen der Mehrfenstertechnik gut nachvollziehbar. Dennoch tat sich das W3-Konsortium schwer damit, die Frames in den HTML-Standard mit aufzunehmen. 1998 übernahm man das Konzept zwar in HTML 4.0, führte dazu aber eine eigene HTML-Variante neben »strict« und »transitional« ein: die Variante »Frameset«.

Mittlerweile gelten Frames in professionellen Kreisen als verpönt. Dafür gibt es mehrere Gründe:

- Inhaltsseiten haben selbst keine vollständige Struktur und in der Regel keine Navigationslinks.
- Anwender können z. B. keine Bookmarks auf bestimmte Unterseiten eines Angebots setzen, weil in der Adresszeile des Browsers immer nur der URI der Seite angezeigt wird, die das Frameset enthält. Durch etwas Programmieraufwand ist dieses Problem zwar lösbar, doch die meisten Anbieter, die Frames einsetzen, kümmern sich nicht darum.
- Frames sind ungünstig für Suchmaschinen. Die meisten Suchmaschinen lösen zwar die Frame-Struktur auf und indexieren die inhaltstragenden Seiten. Doch die Links bei Suchtreffern führen dann direkt zu den Unterseiten. Da diese aber nur einen Teil der Seite darstellen und meist keine eigene Navigation enthalten, landen die Seitenbesucher in einer Hypertext-Sackgasse, die oft auch vom Layout her wie eine »halbe Sache« wirkt. Der Besucher erhält keinen besonders guten Eindruck vom Anbieter der Seite.

- Sind Frames bei der Bildschirmanzeige noch praktisch, so stellen sie beim Ausdrucken von Webseiten nicht selten ein Problem dar. Hat der Anwender den Fokus im Frame für die Navigation, kann es beispielsweise passieren, dass der Browser nur die Navigation ausdruckt und nicht wie gewünscht den Inhalt. Unerfahrene Anwender sind dadurch schnell irritiert.
- Bei drei und mehr Frames soll pro Link nicht selten mehr als ein anderer Frame seinen Inhalt ändern. Mit HTML allein ist es jedoch nicht möglich, zwei oder mehr URIs pro Link zu adressieren. Deshalb wird zu JavaScript gegriffen, was aber nicht funktioniert, wenn der Anwender JavaScript ausgeschaltet hat. Die gesamte Navigation funktioniert dann nicht mehr.
- Frame-basierte Seiten sind ein Problem für Browser oder Ausgabegeräte, die nicht frame-fähig sind. Zwar erkennen modernere nicht grafische Browser wie Lynx Frames, doch das Handling für den Anwender ist meist unbefriedigend.
- Frames erfordern mehr Kommunikation zwischen Browser und Webserver, da pro Frame Ressourcen übertragen werden müssen.
- Frames verlocken zu einem unfairen, urheberrechtlich sehr bedenklichen Trick: Nur die Navigation gehört zur eigenen Site, während die Inhalte aus fremden Quellen stammen. In der Adresszeile des Browsers ist das für den Anwender jedoch nicht erkennbar.

Im Grunde gibt es zwei Hauptgründe, weswegen Frames immer noch zum Einsatz kommen:

- Bestimmte Inhalte scrollen nicht mit, z. B. eine Navigation, ein Logo usw.
- Wiederkehrende Teile wie der Code für eine Navigation muss nur einmal in einer separaten Datei notiert werden, aber nicht mehr in jeder einzelnen Datei.

Für »non-scrolling-regions« steht mit der CSS-Angabe `position:fixed` mittlerweile eine Alternative zur Verfügung. Wir werden in diesem Kapitel auch zeigen, wie Sie diese Alternative richtig einsetzen.

Gegen das Argument der einmaligen Notation von Code wird von Frames-Gegnern in der Regel angeführt, dass es serverseitige Möglichkeiten gebe, Code nicht immer wieder kopieren zu müssen. Dies läuft jedoch auf dynamisch erzeugte Seiten hinaus. Im einfachsten Fall über Server Side Includes, meist aber über Scriptsprachen wie PHP. Wenn Sie ohnehin eine Site entwickeln, die nicht mit statischen HTML-Dateien, sondern etwa mit PHP-Scripts arbeitet, dann sind Frames ohnehin nur lästig und problematisch. Wenn Sie ein HTML-basiertes Projekt dagegen unter anderem auch auf Wegen wie CD-ROM verbreiten möchten, müssen Sie statische HTML-Dateien verwenden. In diesem Fall aber zieht das Argument der Frames-Gegner nicht und es ist unter Umständen durchaus vertretbar, sich bewusst für den Einsatz von Frames zu entscheiden.

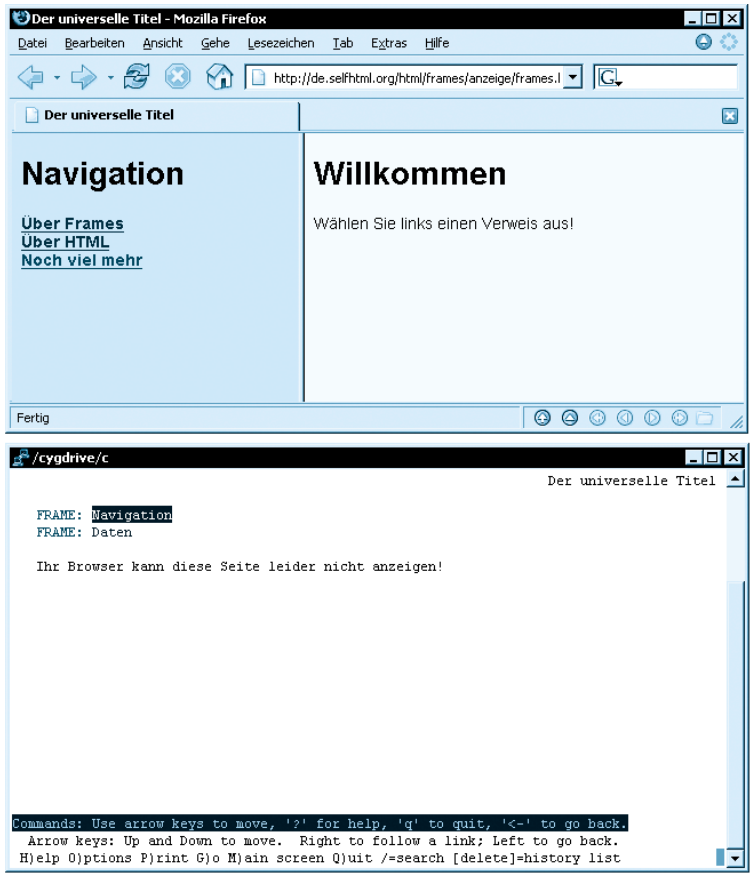


Abbildung 6.1: Frameset im grafischen Browser (oben) und in Lynx (unten)

Link zum Thema Frames-Problematik:

Artikel »Warum Frames out sind«:
http://www.subotnik.net/html/frames.html

6.1.1 HTML mit Framesets und Frames

Das »Denken in Frames« erfordert zunächst einmal, sich Gedanken über mehrere unterschiedliche Dateien zu machen. Erforderlich sind eine Datei, in der das Frameset definiert wird, also die Aufteilung der Einzelfenster sowie je eine Datei bzw. Quelle, die beim Aufruf der Frameset-Datei in die einzelnen Frame-Fenster geladen werden soll.

Datei mit Frameset-Definitionen

Das nachfolgende Listing zeigt den kompletten Quelltext einer Datei mit einer Frameset-Definition:

Listing 6.1: Datei mit Frameset-Definition

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
<html>
   <head>
      <title>Mein erstes Frameset</title>
   </head>
   <frameset cols="250,*">
      <frame src="verweise.htm" name="Navigation">
      <frame src="startseite.htm" name="Daten">
      <noframes>
         Ihr Browser kann diese Seite leider nicht anzeigen!
      </noframes>
   </frameset>
</html>
```

Zunächst fällt auf, dass eine andere Dokumenttyp-Deklaration verwendet wird. In Dateien, die ein Frameset definieren, müssen Sie diese Deklaration verwenden. Damit erfüllt eine solche Seite den HTML 4.01-Standard in der Variante für Framesets.

Die weitere Struktur weicht ebenfalls stark von sonst üblichen HTML-Dokumenten ab. Das html-Element enthält nicht wie sonst üblich die Elemente `head` und `body` als Inhalt, sondern die Elemente `head` und `frameset`. Das body-Element wird also durch das frameset-Element ersetzt. Im head-Element können jedoch die gleichen Daten stehen wie bei gewöhnlichen HTML-Dokumenten. Das title-Element ist Pflicht und andere Angaben, wie etwa meta- oder link-Elemente, sind ohne Ausnahme erlaubt und meist auch sehr sinnvoll.

Das frameset-Element (`<frameset>...</frameset>`) bestimmt die Verteilung der Fenster. Dazu kann es die beiden Attribute `rows=` und `cols=` enthalten. Durch das rows-Attribut werden Fenster untereinander angeordnet (***rows*** = Reihen), durch das cols-Attribut nebeneinander (cols = ***columns*** = Spalten).

Im obigen Beispiel-Listing werden durch die Angabe `cols="250,*"` zwei Frame-Fenster nebeneinander angeordnet. Das linke Frame-Fenster soll 250 Pixel breit sein. Das rechte soll so breit sein, wie der Rest des Browser-Anzeigefensters es erlaubt. Die Wertzuweisung besteht also in kommagetrennten Werten, wobei eine Zahl eine Pixelangabe bedeutet. Ferner sind Prozentangaben erlaubt wie etwa `cols="30%,*"`. Das Sternchen ist ein Platzhalter und bedeutet »die Breite soll vom Browser bestimmt werden«.

Selbstverständlich ist eine Aufteilung in mehr als zwei Spalten oder Reihen möglich. So definiert beispielsweise `rows="200,250,*"` drei Frame-Fenster untereinander, wobei das erste 200 Pixel hoch ist, das zweite 250 Pixel hoch und das dritte soll so hoch sein wie der Rest der Anzeigehöhe des Browser-Fensters es ermöglicht.

Möglich sind auch Angaben wie `cols="100,3*,5*"`. Dies ist sinnvoll, wenn Sie ein oder zwei Frames mit einer Pixelangabe zu Höhe und Breite versehen möchten, die übrigen Höhen bzw. Breiten dagegen flexibel gestalten. Die Angabe im Beispiel bedeutet: drei Frame-Fenster nebeneinander, wobei das erste 100 Pixel breit sein soll. Von der verbleibenden Breite soll

das zweite Fenster drei Achtel einnehmen und das dritte fünf Achtel. Zahlen, gefolgt von einem Sternchen, werden als Verhältnisangabe betrachtet, ähnlich wie Prozentangaben, jedoch bezogen auf den unbekannt großen, verbleibenden Teil des Browser-Fensters. Die Summe der Zahlen ist der gemeinsame Nenner. Die entstehenden Bruchwerte sind für den Browser die Vorgabewerte zur Berechnung der tatsächlichen Framefenster-Breite bzw. -Höhe.

Die Attribute `cols=` und `rows=` dürfen auch beide gleichzeitig verwendet werden. Ein Beispiel:

```
<frameset  rows="100,*" cols="25%,*">
   <frame src="obenlinks.html" name="f1">
   <frame src="obenrechts.html" name="f2">
   <frame src="untenlinks.html" name="f3">
   <frame src="untenrechts.html" name="f4">
</frameset>
```

Durch eine solche Aufteilung wird ein Gitter mit so vielen Spalten mal Reihen wie angegeben erzeugt. Häufig werden jedoch auch andere Aufteilungen gewünscht, wie etwa »oben ein Streifen über die gesamte Breite und unten zwei Spalten«. Für solche Aufteilungen besteht die Möglichkeit, Frameset-Definitionen zu verschachteln:

```
<frameset rows="100,*">
   <frame src="oben.html" name="f1">
   <frameset rows="250,*">
      <frame src="unenlinks.html" name="f2">
      <frame src="untenrechts.html" name="f3">
  </frameset>
</frameset>
```

Es werden also ein äußeres und ein inneres Frameset definiert. Das äußere Frameset besorgt mit `rows=` die Aufteilung in zwei Frame-Fenster untereinander. Das frame-Element für die obere Quelle wird auch erst einmal notiert. Anstelle des frame-Elements für das untere Frame-Fenster wird jedoch ein weiteres, inneres frameset-Element notiert. Dieses bewirkt eine Aufteilung in zwei weitere, nebeneinander anzuordnende Frame-Fenster (`cols=`).

Für jeden kommaseparierten Wert, der bei einem rows- oder cols-Attribut zugewiesen wird, muss innerhalb des frameset-Elements also entweder ein frame-Element notiert werden, welches den Fensterinhalt angibt, oder ein weiteres, inneres frameset-Element, welches den Raum des Frame-Fensters für eine Aufsplittung in weitere Frame-Fenster nutzt.

Während das frameset-Element aus Anfangs- und End-Tag besteht, ist `<frame...>` ein Standalone-Tag. In XHTML muss es folglich in der Form `<frameset ... />` notiert werden. Das frame-Element hat zwei wichtige Standardattribute. Beim src-Attribut wird der URI des Inhalts angegeben, der im entsprechenden Frame-Fenster angezeigt werden soll. Es kann sich um eine lokal referenzierte Datei handeln oder auch um eine entfernte, absolute Internetadresse. Inhalte können HTML-Dateien sein, aber natürlich auch andere Quellen, wie Grafiken, Flash-Movies, PDF-Dateien oder Ähnliches. Auch dynamische Quellen wie

PHP-Scripts können selbstverständlich in jedem einzelnen Frame-Fenster als Quelle angegeben werden. Bei Grafik- oder Multimedia-Quellen ist allerdings zu beachten, dass hierbei keine Möglichkeit besteht, einen Alternativtext anzugeben.

Das andere wichtige Attribut für jedes frame-Element ist das name-Attribut. Der frei vergebbare Name ist vor allem wichtig, wenn Seiten, die innerhalb eines Frame-Fensters angezeigt werden, mit einem Link den Inhalt eines anderen Frame-Fensters verändern wollen. Bei solchen Links muss das andere Frame-Fenster unter dem hier vergebenen Namen mit adressiert werden.

Noframes-Bereiche

Zwischen `<frameset>` und `</frameset>` darf neben inneren frameset-Elementen und frame-Elementen noch ein weiteres Element stehen, ausgezeichnet durch `<noframes>...</noframes>`. Dieses Element erlaubt es, für Browser oder andere Client-Programme, welche keine Frames anzeigen können, einen alternativen Inhalt anzubieten. Ein Beispiel:

```
<frameset rows="100,*">
   <frame src="oben.html" name="f1">
   <frameset rows="250,*">
      <frame src="unenlinks.html" name="f2">
      <frame src="untenrechts.html" name="f3">
  </frameset>
  <noframes>
    <h1>Liebe Besucher</h1>
    <p>
       Diese Site verwendet Frames.
       Bitte rufen Sie die <a href="noframes.html">
       Navigation für nicht-frame-fähige
       Browser</a> auf!
    .</p>
  </noframes>
</frameset>
```

Das noframes-Element darf am Ende einer frameset-Struktur vor dem schließenden `</frameset>`-Tag des äußersten Framesets notiert werden. Als Inhalt sind beliebige Block- und Inline-Elemente erlaubt.

Interessant ist, dass das noframes-Element auch in HTML-Dokumenten notiert werden darf, die innerhalb eines Framesets in einem Frame-Fenster angezeigt werden. Dort muss allerdings im Dokumenttyp die Transitional-Variante von HTML angegeben werden. Das noframes-Element kann dann als ganz normales Element im Dateikörper notiert werden. Innerhalb davon können z.B. Rückverweise auf eine Seite mit Navigationslinks notiert werden. Browser wie Lynx, die zwar Framesets erkennen, jedoch nicht anzeigen können, stellen den Inhalt von noframes-Elementen im Dateikörper dar. Browser, die Frames anzeigen können, ignorieren die Inhalte dagegen.

6.1.2 Links zu anderen Frame-Fenstern

Wie bereits erwähnt, müssen Links innerhalb eines Framesets, deren Ziel in einem anderen als dem aktuellen Frame-Fenster angezeigt werden soll, das gewünschte Frame-Fenster benennen. Anzugeben ist der Name, der bei `<frame ... name=>` vergeben wurde. Ein Beispiel:

```
<a href="index.htm" target="inhalt">Startseite</a>
```

Dieser Link öffnet sein Ziel in einem Fenster, das mit `<frame ... name="inhalt">` definiert wurde. Das target-Attribut gibt den gewünschten Fensternamen an (*target* = Ziel).

Bei Navigationslinks, die alle auf ein anderes Frame-Fenster verweisen, gibt es jedoch noch eine andere Möglichkeit, nämlich im Kopfbereich des HTML-Dokuments ein Zielfenster vorzugeben. Beispiel:

```
<head>
   <!-- andere Kopfdatendefinitionen -->
   <base target="inhalt">
</head>
```

Das base-Element kann ebenfalls ein target-Attribut enthalten. So notiert wie im Beispiel hat es die Wirkung, dass alle Linkziele des aktuellen Dokuments in einem Frame-Fenster namens `inhalt` geöffnet werden.

Groß- und Kleinschreibung werden übrigens unterschieden. Wenn beispielsweise ein Fenster mit `<frame ... name="Inhalt">` definiert wurde, kann es nicht mit `<a ... target="inhalt">` angesprochen werden! target-Angaben zu nicht existierenden Frame-Fenstern bewirken in den meisten Browsern übrigens, dass der Link stattdessen in einem neuen Fenster oder in einem neuen Tab geöffnet wird.

Wichtig zu wissen ist ferner, dass HTML-Dokumente, die in Hyperlinks oder im base-Element das target-Attribut benutzen, in der Dokumenttyp-Deklaration nicht mit der HTML-Variante »strict« ausgezeichnet werden dürfen. Das target-Attribut gehört zu den HTML-Bestandteilen, die im »eigentlich gewünschten« HTML keinen Platz mehr haben, ebenso wenig wie Frames allgemein. Verwenden Sie für Dokumente, in denen target-Attribute vorkommen, stattdessen die HTML-Variante »transitional«, also so:

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

6.1.3 Rahmen und andere Frame-Eigenschaften

Normalerweise zeigen frame-fähige Browser zwischen den Frame-Fenstern systemabhängig aussehende Rahmen an, die es dem Anwender unter anderem erlauben, das Fensterverhältnis durch Ziehen der Rahmen mit der Maus zu ändern. Ein häufiger Wunsch von Seitenanbietern sind »unsichtbare Fensterrahmen«. Dadurch erscheint dem Anwender die gesamte Seite wie aus einem Dokument, obwohl in Wirklichkeit ein Frameset am Werk ist.

Meist wurde dieses Feature in der Vergangenheit deshalb verlangt, um zwar eine nahtlose Ansicht im Browser-Fenster anzubieten, jedoch mit einem fixen Bereich für Logo, Navigation usw., der nicht mit scrollt. Der Wunsch ist nachvollziehbar, doch das Mittel, hierfür zu einem Frameset zu greifen, ist mittlerweile nicht mehr zeitgemäß. Fixe Bereiche sollten mithilfe von CSS definiert werden, wie in ***Abschnitt 6.1.4*** beschrieben.

Generell wird die Rahmendicke wie im folgenden Beispiel bestimmt:

```
<frame src="index.htm" name="inhalt" frameborder="10">
```

Das frameborder-Attribut ist laut HTML-Spezifikation in `<frame>`-Tags erlaubt. Dort notiert, bestimmt es die Rahmendicke in Pixel, und zwar für alle Rahmen zwischen diesem Frame-Fenster und benachbarten Frame-Fenstern. Um keine Rahmen anzuzeigen, wird der zugewiesene Wert auf 0 gesetzt. Notieren Sie die frameborder-Angabe sicherheitshalber in allen `<frame>`-Tags. Andernfalls erreichen Sie in den Browsern möglicherweise nicht den gewünschten Effekt.

Ein anderer Fall ist, dass Sie die Rahmen zwischen den Frame-Fenstern zwar optisch für sinnvoll halten, aber nicht möchten, dass der Anwender die Größe der Frame-Fenster damit verändern kann. Für diesen Fall gibt es ein weiteres Attribut, wie das nachfolgende Beispiel zeigt:

```
<frame src="index.htm" name="inhalt" noresize>
```

Das Standalone-Attribut `noresize` (also »kein Resize«, keine Größenänderung) verhindert für die Rahmen um das betroffene Frame-Fenster, dass diese mit der Maus in ihrer Position veränderbar sind. Beachten Sie, dass Sie bei XHTML `noresize="noresize"` notieren müssen.

Schließlich können Sie noch das Scrollverhalten von Frame-Fenstern beeinflussen. Dazu können Sie im `<frame>`-Tag eines Fensters das Attribut `scrolling=` wahlweise mit den Werten `yes`, `no` oder `auto` versorgen: `yes` erzwingt Scrollleisten sogar dann, wenn der Inhalt gar kein Scrollen erfordert; `no` verhindert Scrollleisten in jedem Fall, auch dann, wenn der Inhalt Scrollen erfordert, und `auto`, die Voreinstellung, bewirkt, dass Scrollbars dann angezeigt werden, wenn es der Inhalt erfordert.

Auf weitere Eigenschaften gehen wir an dieser Stelle nicht mehr ein, da diese entweder nicht zum HTML-Standard gehören oder nicht mehr zeitgemäß sind.

6.1.4 Fixe Bereiche ohne Frames

Wie bereits erwähnt, werden Frames in den meisten Fällen nur eingesetzt, um einen fixen Bereich zu haben, der nicht mit scrollt (auch als ***Non-Scrolling-Region*** bezeichnet). In diesem Abschnitt möchten wir zeigen, wie so etwas auch ohne Frames funktioniert – aus der Überlegung heraus, dass es einfach zu schade ist, die zahlreichen Nachteile von Frames in

Kauf zu nehmen, wenn das gewünschte Ziel auch anders erreichbar ist. Das HTML-Dokument hat folgenden Inhalt:

Listing 6.2: HTML-Dokument mit Bereichen für Navigationslinks und Inhalt

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
   <head>
      <meta http-equiv="content-type" content="text/html;
            charset=ISO-8859-1">
      <title>Fixer Bereich</title>
      <link rel="stylesheet" type="text/css" href="fix.css">
   </head>
   <body>
      <div id="navigation">
         <a href="s2.html">Seite 2</a><br>
         <a href="s3.html">Seite 3</a><br>
         <a href="s4.html">Seite 4</a>
      </div>
      <div id="content">
         <h1>Seite 1</h1>
         <p>viel Inhalt</p> ... <p>viel Inhalt</p>
      </div>
   </body>
</html>
```

Die beiden Bereiche erhalten die id-Namen `navigation` und `content`. In der eingebundenen CSS-Datei ***fix.css*** werden folgende Formate definiert:

Listing 6.3: CSS-Formatdefinition für fixen und normalen Bereich

```
#navigation {
   position:fixed;
   top:20px;
   left:20px;
   padding-right:20px;
   border-right:blue solid 4px;
}
#content {
   margin-left:200px;
   margin-right:20px;
}
```

Der Bereich `navigation` wird mit der Angabe `position:fixed` fest positioniert. Beim Scrollen wandert sein Inhalt nicht mit. Die restlichen Definitionen betreffen Randabstände sowie einen blauen dicken Rahmen rechts von den Links.

Der Bereich `content` soll im normalen Textfluss bleiben, erhält also keine Angabe zur Positionierung. Mit `margin-left:200px` wird ein hinreichend großer linker Rand geschaffen, der die Anzeige der Navigationslinks in diesem Bereich erlaubt.

Eigentlich ist das bereits alles und in Browsern, welche `position:fixed` interpretieren, funktioniert auch alles wie gewünscht.

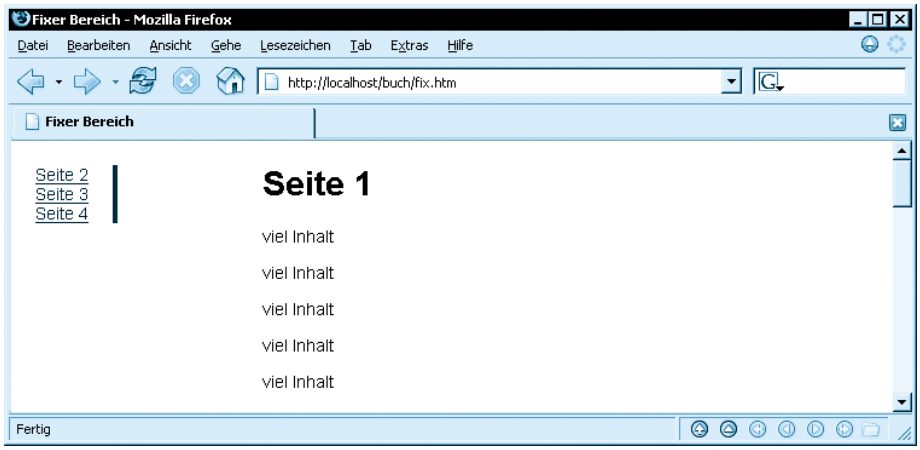


Abbildung 6.2: Fixer Bereich im Firefox-Browser

Der Inhalt kann gescrollt werden, während die Links fest an ihrer Position bleiben. Leider interpretiert der MS Internet Explorer bis einschließlich Version 6.0 `position:fixed` nicht. Stattdessen ignoriert er die Positionierungsangabe, was durchaus ein korrektes Verhalten ist, aber dazu führt, dass die beiden Bereiche beide im normalen Textfluss verbleiben.



Abbildung 6.3: Nicht erkannter fixer Bereich im MS Internet Explorer

Der div-Bereich für die Navigationslinks verhält sich also wie ein gewöhnliches Blockelement im Textfluss. Es steht an der Stelle, wo es notiert wird (zuerst), und nimmt die gesamte verfügbare Breite ein, so dass der div-Bereich für den Inhalt erst unterhalb davon angezeigt wird.

Dieses Verhalten ist jedoch nicht erwünscht. Andererseits kann ein so verbreiteter Browser nicht einfach ignoriert werden. Wir benötigen also eine Behelfslösung. Dank anderer, proprietärer Fähigkeiten des Internet Explorers ist das Verhalten von `position:fixed` »simulierbar«.

Workaround für den Internet Explorer

Dazu wird zunächst der Inhalt der CSS-Datei *fix.css* wie folgt geändert:

```
body > #navigation {
   position:fixed;
   top:20px;
   left:20px;
   padding-right:20px;
   border-right:blue solid 4px;
}

body > #content {
   margin-left:200px;
   margin-right:20px;
}
```

Geändert wird die Syntax der Selektoren dahingehend, dass von den CSS-2.0-spezifischen Adressierungsmöglichkeiten für Elemente Gebrauch gemacht wird. Diese werden vom Internet Explorer ebenfalls nicht interpretiert. Als Ergebnis ignoriert dieser Browser die Formatdefinitionen für beide div-Bereiche komplett.

Im zweiten Schritt bekommt der Internet Explorer stattdessen eine nur für ihn lesbare CSS-Datei spendiert. Dazu wird im HTML-Dokument – am besten hinter dem `<link>`-Tag, welches das normale Stylesheet einbindet – Folgendes notiert:

```
<!--[if gte IE 5]>
   <link rel="stylesheet" type="text/css" href="ie-fix.css">
<![endif]-->
```

Der gesamte Code stellt aus HTML-Sicht einen Kommentar dar. Eigentlich sollten Browser den Inhalt von Kommentaren ignorieren und die meisten tun das auch. Doch nicht so der Internet Explorer. Hier kann der Kommentar interpretierbaren Inhalt haben. Das Konstrukt `[if gte IE 5]>....` `<![endif]` wird, wenn ohne Leerraum nach bzw. vor den Kommentarklammern notiert, als Verarbeitungsanweisung interpretiert. `[if gte IE 5]>` bedeutet so viel wie: »Wenn ein Internet Explorer mit Versionsnummer größer oder gleich 5.0 das hier liest, dann soll er den HTML-Code bis zu `<![endif]` trotz Kommentar interpretieren.« Genutzt wird diese Fähigkeit in unserem Fall, um mittels `<link>`-Tag eine weitere CSS-Datei speziell für den Internet Explorer einzubinden. Im Beispiel nennen wir sie *ie-fix.css*.

Diese spezielle CSS-Datei erhält folgenden Inhalt:

```
html, body {
   overflow:hidden;
   width:100%;
   height:100%;
}
#navigation {
   position:absolute;
   top:20px;
   left:20px;
   padding-right:20px;
   border-right:blue solid 4px;
}
#content {
   position:absolute;
   top:0px;
   left:200px;
   padding-top:20px;
   height:expression(document.body.clientHeight - 20 + "px");
   width:expression(document.body.clientWidth - 200 + "px");
   overflow:auto;
}
```

Zunächst wird für die Basiselemente `html` und `body` festgelegt, dass diese die volle Breite und Höhe einnehmen. Gleichzeitig wird bestimmt, dass Inhalte, die länger oder breiter sind, einfach abgeschnitten werden (`overflow:hidden`).

Die beiden div-Bereiche mit den id-Namen `navigation` und `content` werden daraufhin beide absolut positioniert. Bis auf die Art der Positionierung sind die Formatdefinitionen für den Bereich `navigation` die gleichen wie in der normalen CSS-Datei.

Bei den Definitionen für `content` gibt es hingegen einige Unterschiede. Mit `top` und `left` muss dessen gewünschte Anfangsposition bestimmt werden, da er ja absolut positioniert wird. Eigentlich soll er 20 Pixel von oben beginnen. Damit der Internet Explorer die vertikale Scrollleiste jedoch wie üblich oben im Fenster beginnen lässt, weisen wir `top:0px` zu und schaffen den gewünschten Abstand nach oben über den Umweg `padding-top:20px`.

Besonders spannend sind die Zuweisungen an die CSS-Eigenschaften `height` und `width`. Anstelle eines festen Werts wird ein Wert zugewiesen, der mithilfe von JScript errechnet wird. Der Internet Explorer erlaubt das Zuweisen gewisser Funktionen an CSS-Eigenschaften, unter anderem die Funktion `expression()`. Diese ermöglicht das Ausführen von JavaScript/JScript-Code. In den beiden Beispielangaben werden die Breite und die Höhe des zu positionierenden Bereichs aus der tatsächlichen Breite und Höhe bestimmt, die der in HTML in diesem Bereich notierte Inhalt einnimmt. Die beiden Objekteigenschaften `document.body.clientWidth` und `document.body.clientHeight` liefern die Werte für den gesamten body-Bereich. Abgezogen werden die gewünschten Startpositionen des Inhaltsbereichs (der Obenwert bei der Höhe und der Linkswert bei der Breite). Ausdrücklich zugewiesen wird dann noch `overflow:auto`. Somit wird der Bereich gescrollt, falls sein Inhalt es erfordert.



Abbildung 6.4: Simulierter funktionsfähiger fixed-Bereich im Internet Explorer

In unserem Beispiel gibt es nun keinerlei Unterschiede in der Funktionalität etwa zwischen Firefox, Opera und anderen einerseits und Internet Explorer andererseits. In allen Browsern bleiben die Navigationslinks fix an ihrer Position.

Wenn Sie den hier beschriebenen Workaround für fix positionierte Bereiche an anderer Stelle, beispielsweise oben, nutzen wollen, müssen Sie gegebenenfalls in der CSS-Datei für den Internet Explorer bei `html` und `body` nicht `overflow:hidden` angeben, sondern `overflow-y:hidden`.

6.1.5 Eingebettete Frames

Nachdem wir im vorherigen Abschnitt eine Möglichkeit beleuchtet haben, um Frames zu vermeiden, werden wir nun noch eine spezielle Form von Frames kennen lernen, die kein Frameset benötigt, sondern direkt im normalen body-Inhalt notiert werden kann.

Eingebettete Frames (auch als ***Inline-Frames*** bezeichnet) ermöglichen es, an einer gewünschten Stelle innerhalb des sichtbaren Inhalts eines HTML-Dokuments Inhalte aus anderen Dateien oder Quellen einzubinden. Dazu stellt HTML das iframe-Element zur Verfügung (iframe = inline frame). Wie der Name schon vermuten lässt, hat dieses Element Inline-Charakter, d.h., es kann – ähnlich wie Grafikreferenzen – auch mitten im Text notiert werden.

Zum reinen Einbetten einer anderen Quelle kann übrigens auch das object-Element verwendet werden, das im Gegensatz zum iframe-Element zum strict-Standard von HTML gehört (siehe auch ***Abschnitt 4.9.8***). Bei Verwendung des iframe-Elements müssen Sie dagegen den Dokumenttyp für die HTML-Variante »transitional« angeben. Erforderlich ist die Verwendung des iframe-Elements jedoch dann, wenn Hyperlinks ihr Verweisziel im Fenster des Inline-Frames öffnen sollen. Genau diesen Anwendungsfall werden wir im nachfolgenden Beispiel behandeln.

Der Quelltext der Datei mit dem eingebetteten Frame lautet:

Listing 6.4: HTML-Dokument mit eingebettetem Frame-Fenster

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd">
<html lang="de">
   <head>
      <meta http-equiv="content-type" content="text/html;
            charset=ISO-8859-1">
      <title>Eingebetteter Frame</title>
   </head>
   <body style="text-align:center">
      <h1>Blindtext</h1>
      <p>
         <a href="seite1.htm" target="content">Seite 1</a> |
         <a href="seite2.htm" target="content">Seite 2</a> |
         <a href="seite3.htm" target="content">Seite 3</a> |
         <a href="seite4.htm" target="content">Seite 4</a>
      </p>
      <p>
         <iframe name="content" src="seite1.htm" frameborder="0"
            style="width:60%; height:200px;
            border:thin solid #C0C0C0">
               Ihr Browser unterstützt
               leider keine eingebetteten Frames!
         </iframe>
      </p>
   </body>
</html>
```

Das iframe-Element wird mit Start- und End-Tag notiert. Zwischen `<iframe>` und `</iframe>` kann Inhalt für Browser notiert werden, die das iframe-Element nicht kennen oder interpretieren. Dabei dürfen auch beliebige Block- und Inline-Elemente notiert werden.

Wie beim frame-Element wird auch beim iframe-Element eine Default-Quelle über das src-Attribut referenziert. Es kann sich um einen beliebigen URI handeln, also um eine lokal referenzierte Datei oder auch um eine absolute Internetadresse. Wie auch beim frame-Element ist das name-Attribut wichtig, wenn Linkziele im eingebetteten Frame geöffnet werden sollen.

Weitere Angaben zum Frame-Fenster wie Breite und Höhe des Frame-Fensters werden im obigen Beispiel zeitgemäß über CSS gelöst. Die Angabe `width:60%` im Beispiel bedeutet: 60% der Breite des Elternelements. Dieses ist das umgebende p-Element, welches sich im normalen Textfluss innerhalb des body-Elements befindet. Da es ein Block-Element ist, nimmt es die maximal verfügbare Breite ein, also die gesamte Breite des Anzeigefensters abzüglich der Default-Ränder, die der Browser setzt.

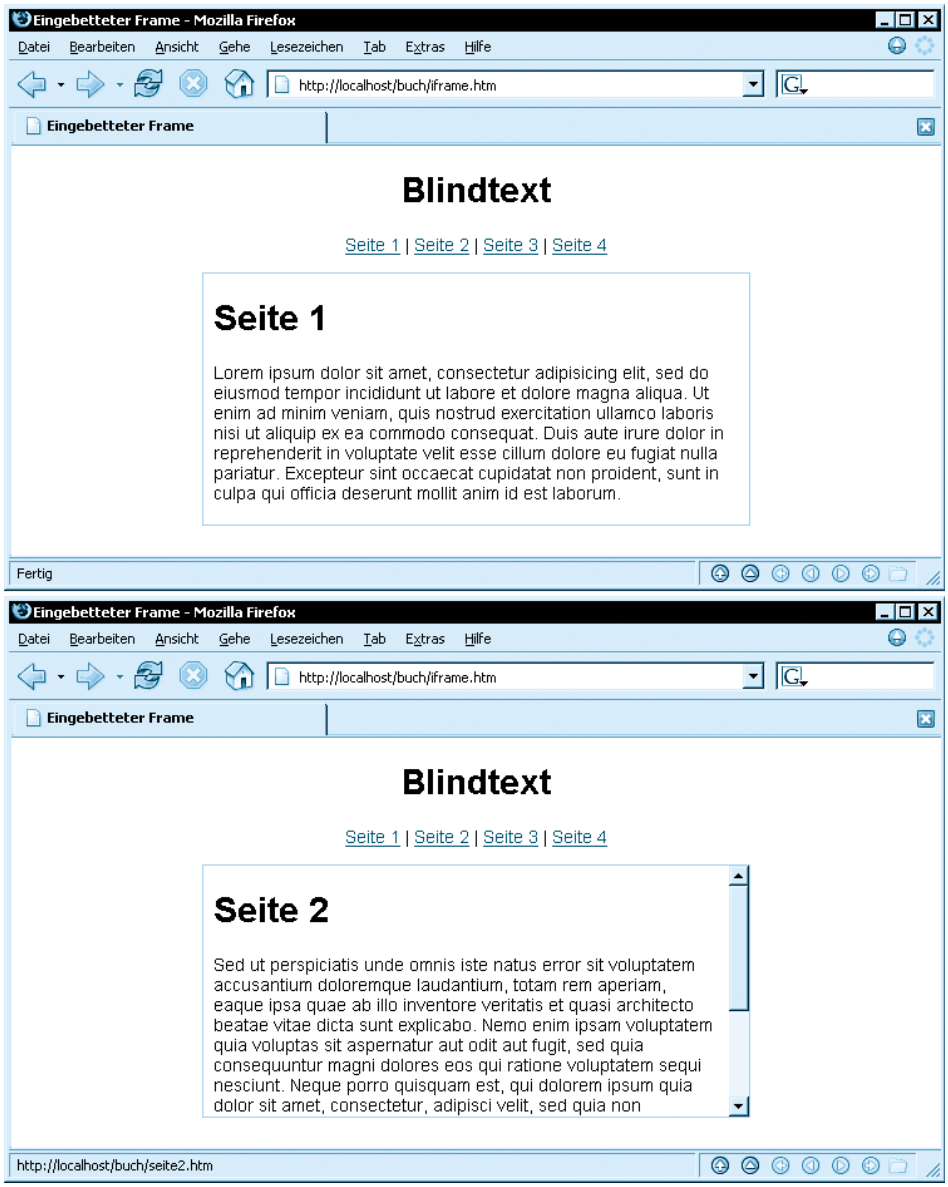


Abbildung 6.5: Im eingebetteten Frame-Fenster angezeigte Verweisziele

Mithilfe von CSS lässt sich auch der Rahmen des eingebetteten Frames optimal gestalten. Im Beispiel haben wir einen dünnen grauen Rahmen (`border:thin solid #COCOCO`) gewählt. Dennoch ist zusätzlich das HTML-Attribut `frameborder="0"` notiert. Damit wird explizit der Frame-eigene Rahmen unterdrückt, den der Internet Explorer andernfalls unabhängig von der CSS-Rahmendefinition anzeigt.

Die Links im Beispiel haben im einleitenden `<a>`-Tag ein Attribut `target="content"`. Dadurch wird der Fensterbezug zum iframe-Element hergestellt, das mit `name="content"`

definiert wird. Die Verweisziele werden im Frame-Fenster geöffnet. Scrollleisten zeigt der Browser nach Bedarf an. Durch `scrolling="no"` können Sie die Anzeige von Scrollleisten verhindern.

6.2 Automatische Überschriftennummerierung

Insider betonen immer wieder, dass HTML und CSS keine Programmiersprachen seien, sondern Auszeichnungs- bzw. Beschreibungssprachen. Das ist korrekt. Leider hat das Fehlen von operativen Möglichkeiten aber auch Nachteile. Einer davon ist das Problem, einen Nummerierungsalgorithmus etwa für Überschriften anzugeben. Das Problem wurde vom W3-Konsortium registriert und so steht seit CSS2.0 eine Funktionalität zur Verfügung, die es erlaubt, eine individuelle Überschriftennummerierung zu »programmieren«. Doch leider hapert es mit der Browser-Unterstützung. Weder der MS Internet Explorer 6.0 noch der sonst so standardstarke Firefox-Browser 1.0 unterstützen das entsprechende Feature. Lediglich Opera ist dazu in der Lage.

Aus diesem Grund werden wir zwei Vorgangsweisen beschreiben, die sich ergänzen. Zunächst werden wir die in CSS vorgesehene Methode behandeln. Für andere Browser stellen wir anschließend eine Lösung vor, die ein wenig JavasScript-Programmierung erfordert.

6.2.1 Überschriftennummerierung mit CSS

Ein Beispieldokument enthält diverse Überschriften 1., 2. und 3. Ordnung. Diese sollen nummeriert werden. Der HTML-Quelltext lautet:

Listing 6.5: HTML-Dokument mit Überschriften

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
   <head>
      <meta http-equiv="content-type" content="text/html;
            charset=ISO-8859-1">
      <title>Nummerierte Überschriften</title>
      <link rel="stylesheet" type="text/css" href="h-styles.css">
      </style>
   </head>
   <body>
      <h1>Die Fauna (Tierwelt)</h1>
      <h2>Die Avifauna (Vögel)</h2>
      <h3>Laufvögel</h3>
      <h3>Hühnervögel</h3>
      <h3>Gänsevögel</h3>
      <h3>Kranichvögel</h3>
      <h2>Die Entomofauna (Insekten)</h2>
      <h3>Felsenspringer</h3>
      <h3>Fischchen</h3>
```

```
      <h3>Fluginsekten</h3>
      <h2>Die Ichthyofauna (Fische)</h2>
   </body>
</html>
```

Die Überschriften sollen in der Form 1, 1.1, 1.1.1 usw. nummeriert werden. Dazu wird im Beispiel im HTML-Dateikopf eine CSS-Datei namens *h-styles.css* eingebunden. Diese Datei hat folgenden Inhalt:

Listing 6.6: CSS-Datei mit Formatdefinitionen für Überschriftennummerierung

```
h1:before {
   content:counter(level_1) ". ";
   counter-increment:level_1;
   counter-reset:level_2;
}
h2:before {
   content:counter(level_1) "." counter(level_2) " ";
   counter-increment:level_2;
   counter-reset:level_3;
}
h3:before {
   content:counter(level_1) "." counter(level_2) "." counter(level_3) " ";
   counter-increment:level_3;
}
```

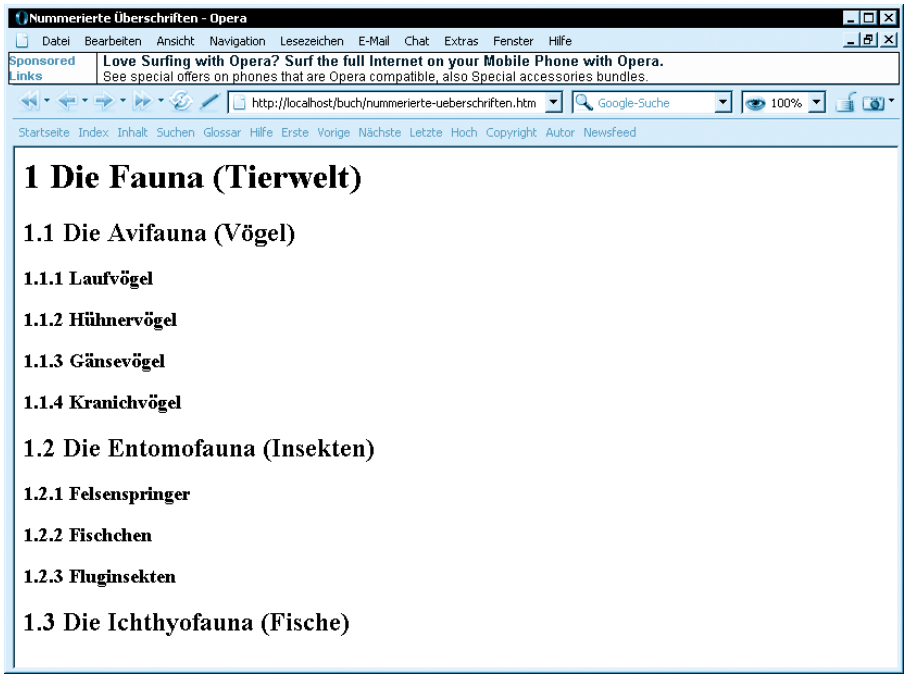


Abbildung 6.6: Mit CSS nummerierte Überschriften im Opera-Browser

Die Überschriftennummerierung benutzt das Pseudoelement `:before`, welches es erlaubt, einen festlegbaren Inhalt vor einem Element zu notieren. Durch die Selektoren `h1:before`, `h2:before` und `h3:before` wird für Überschriften 1., 2. und 3. Ordnung jeweils festgelegt, was vor einer solchen Überschrift automatisch eingefügt werden soll. Der Inhalt dessen, was einem Element automatisch vorangestellt werden soll, wird mithilfe der CSS-Eigenschaft `content` definiert. Das kann ein bestimmter, in Anführungszeichen zu setzender Text sein, wie z. B. `content:"Abschnitt: "`. In unserem Beispiel wird jedoch erst eine Funktion namens `counter()` notiert. Diese Funktion gibt den Wert einer Zählervariable aus. Bei h1-Überschriften hat die Zählervariable in unserem Beispiel den Namen `level_1`, bei h2-Überschriften `level_2` und bei h3-Überschriften `level_3`.

Der Wert, welcher `content:` zugewiesen wird, wird jedoch aus zwei oder mehreren Teilen zusammengesetzt. Die Teile werden jeweils durch Leerzeichen getrennt. Bei h1-Überschriften folgt hinter dem aktuellen Zählerstand der Zählervariable `level_1`, ermittelt durch die Funktion `counter()`, einfach ein Leerzeichen (`" "`). Es hat die Aufgabe, die Überschriftennummer vom Überschriftentext zu trennen.

Bei h2- und h3-Überschriften ist die Zusammensetzung des Werts für `content:` komplexer. Bei h2-Überschriften muss ja der aktuelle h1-Zähler mit ausgegeben werden, und bei h3-Überschriften der aktuelle h1-Zähler und der aktuelle h2-Zähler. Zwischen den Zahlen soll jeweils ein Punkt stehen. Auch dieser muss explizit angegeben werden (`"."`).

Nachdem eine Überschrift ihre korrekte Nummerierung erhalten hat, müssen jedoch noch die Zählervariablen verwaltet werden. Für diese eigentlich typischen Programmieranweisungen stellt CSS die Eigenschaften `counter-increment` (Zählervariable um 1 erhöhen) und `counter-reset` (Zähler auf 1 zurücksetzen) zur Verfügung.

Bei allen drei Überschriftentypen muss die jeweilige Zählervariable, nachdem eine Überschrift dieses Typs ihre aktuelle Nummerierung erhalten hat, um 1 erhöht werden. So erhöht `counter-increment:level_1` den Wert der Zählervariablen `level_1` für h1-Überschriften. Der Eigenschaft `counter-increment` wird also einfach die gewünschte Zählervariable zugewiesen.

Auf gleiche Weise funktioniert die Eigenschaft `counter-reset`. Die zugewiesene Zählervariable wird auf 1 zurückgesetzt. Dies ist wichtig, damit eine Überschriftenfolge wie h1 – h2 – h2 – h1 – h2 zu 1 – 1.1 – 1.2 – 2 – 2.1 führt und nicht zu 1 – 1.1 – 1.2 – 2 – 2.3.

6.2.2 Überschriftennummerierung mit JavaScript/DOM

Um die automatische Überschriftennummerierung auch für andere Browser zu ermöglichen, erstellen wir folgendes JavaScript in einer separaten Datei, die wir im Beispiel unter dem Namen *enumerate.js* abspeichern. Diese Datei erhält folgenden Inhalt:

Listing 6.7: Script mit Funktion enumerate() für Überschriftennummerierung von h1, h2, h3

```
function enumerate(h1_num) {
  if(!document.getElementsByTagName) return;
  if(window.opera) return;
  var body = document.getElementsByTagName("body")[0];
```

```
    h2_num = h3_num = 0;
    h1_num -= 1;
    for(i = 0; i < body.childNodes.length; i++) {
      if(body.childNodes[i].nodeName.toLowerCase() == 'h1') {
        h1_num += 1;
        h2_num = h3_num = 0;
        h1_text = body.childNodes[i].innerHTML;
        h1_numtext = h1_num + " " + h1_text;
        body.childNodes[i].innerHTML = h1_numtext;
      }
      else if(body.childNodes[i].nodeName.toLowerCase() == 'h2') {
        h2_num += 1;
        h3_num = 0;
        h2_text = body.childNodes[i].innerHTML;
        h2_numtext = h1_num + "." + h2_num + " " + h2_text;
        body.childNodes[i].innerHTML = h2_numtext;
      }
      else if(body.childNodes[i].nodeName.toLowerCase() == 'h3') {
        h3_num += 1;
        h3_text = body.childNodes[i].innerHTML;
        h3_numtext = h1_num + "." + h2_num + "." + h3_num + " " +
                     h3_text;
        body.childNodes[i].innerHTML = h3_numtext;
      }
    }
  }
```

Das Script besteht aus einer einzigen Funktion namens `enumerate()`. Auf Einzelheiten der Programmierung gehen wir in diesem Kapitel nicht näher ein. Wichtig zu wissen ist an dieser Stelle nur, was die Funktion leistet. So wie hier notiert, wertet sie alle h1-, h2- und h3-Elemente eines HTML-Dokuments aus und setzt ihrem Elementinhalt die Nummerierung voran. Dabei erwartet die Funktion beim Aufruf eine Zahl. Diese Zahl interpretiert sie als die Basiszahl für die erste h1-Überschrift im Dokument. Wenn Sie der Funktion also beispielsweise den Wert 4 übergeben, nummeriert sie 4, 4.1, 4.1.1 usw.

Die Funktion ist so geschrieben, dass Opera-Browser sie nicht bis zum Ende durchlaufen. Der Grund ist, dass Opera wie im vorherigen Abschnitt gesehen bereits die Nummerierung via CSS umsetzt. Falls Sie keine CSS-basierte Nummerierung verwenden möchten, sondern nur das hier vorgestellte JavaScript, dann müssen Sie in der Funktion folgende Zeile löschen:

```
if(window.opera) return;
```

Nun muss die Funktion noch im HTML-Dokument aufgerufen werden. Der vollständige Quelltext des Beispieldokuments lautet:

Listing 6.8: HTML-Dokument mit automatisch nummerierten Überschriften

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

```
    <head>
       <meta http-equiv="content-type" content="text/html;
             charset=ISO-8859-1">
       <title>Nummerierte Überschriften</title>
       <link rel="stylesheet" type="text/css"
             href="h-styles.css">
       <script type="text/javascript" src="enumerate.js">
       </script>
    </head>
    <body onLoad="enumerate(1)">
       <h1>Die Fauna (Tierwelt)</h1>
       <h2>Die Avifauna (Vögel)</h2>
       <h3>Laufvögel</h3>
       <h3>Hühnervögel</h3>
       <h3>Gänsevögel</h3>
       <h3>Kranichvögel</h3>
       <h2>Die Entomofauna (Insekten)</h2>
       <h3>Felsenspringer</h3>
       <h3>Fischchen</h3>
       <h3>Fluginsekten</h3>
       <h2>Die Ichthyofauna (Fische)</h2>
    </body>
</html>
```

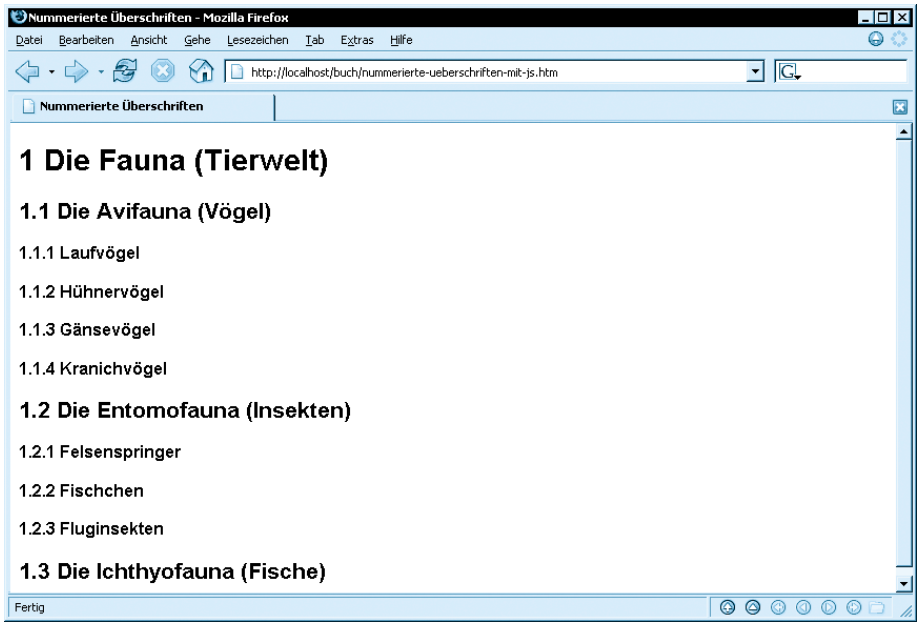


Abbildung 6.7: Automatische Überschriftennummerierung dank JavaScript

Im Kopf des HTML-Dokuments werden sowohl die CSS-Datei *h-styles.css* (zu deren Inhalt siehe Listing 6.6 ***auf Seite 293***) als auch die externe Scriptdatei *enumerate.js* eingebunden. Letztere enthält die weiter oben vorgestellte Funktion `enumerate()`. Eingebunden

wird das Script über ein script-Element, das mit Anfangs- und End-Tag notiert werden muss. Im src-Attribut wird die Scriptdatei als URI angegeben.

Damit ist das Script mit der Nummerierungsfunktion zwar eingebunden, aber die Nummerierungsfunktion muss noch explizit aufgerufen werden. Das darf aber erst geschehen, nachdem der Browser alle Überschriften des Dokuments eingelesen hat. Der Funktionsaufruf erfolgt deshalb im einleitenden <body>-Tag mithilfe des Event-Handlers `onLoad=`. Dieser Event wird ausgelöst, wenn der Browser das Dokument vollständig eingelesen hat. Die Überschriften werden zu diesem Zeitpunkt bereits angezeigt, und zwar ohne Nummerierung. Die Funktion `enumerate()` erledigt ihre Arbeit jedoch blitzschnell und in den meisten Fällen wird ein Anwender gar nicht merken, dass die Überschriften zunächst einen Moment lang nicht nummeriert dargestellt werden.

Die Funktion `enumerate()` wird im Beispiel mit dem Wert 1 aufgerufen. Das bedeutet, die Zählung für h1-Überschriften beginnt bei 1.

Die Nummerierung via JavaScript funktioniert nur, wenn ein Anwender JavaScript in seinem Browser aktiviert hat, und wenn der Browser DOM-fähiges JavaScript interpretiert. Damit keine JavaScript-Fehler auftreten, verhindert die Funktion `enumerate()`, dass unfähige Browser sie vollständig ausführen. Alle heute verbreiteten und webtauglichen Browser sollten die Funktion jedoch korrekt ausführen.

6.3 @-Regeln in CSS

Zur spezielleren Syntax von CSS zählen die »at-Regeln«. Folgende Regeln gehören dazu:

- die @import-Regel zum Einbinden anderer, in einer externen CSS-Datei notierten Formatdefinitionen,
- die @media-Regel zum Unterscheiden ausgabetypspezifischer Formatdefinitionen,
- die @page-Regel für druckorientierte Stylesheet-Definitionen,
- die @charset-Regel für CSS-eigene Zeichensatzdefinitionen.

Eine Regel kann eine einzelne Verarbeitungsanweisung sein, die mit einem abschließenden Semikolon endet. Es gibt jedoch auch Regeln, an die sich ein kompletter Block anschließt, begonnen und beendet durch geschweifte Klammern – genau so wie Formatdefinitionen zu einem Selektor.

6.3.1 Die @import-Regel

In den bisherigen Beispielen haben wir externe CSS-Dateien stets über `<link rel= "stylesheet" ...>` eingebunden. CSS bietet mit der @import-Regel jedoch auch eine eigene Möglichkeit zum Einbinden externer Stylesheets an. Ein Beispiel:

```
<style type="text/css">
   @import url(styles.css) screen;
   @import url(../special/styles-print.css) print;
```

```
    #example {
       background-color:yellow;
    }
</style>
```

Die @import-Regel wird innerhalb eines zentralen style-Bereichs im Kopf einer HTML-Datei notiert. Wichtig ist zunächst zu wissen, dass die @import-Regel vor allen anderen Definitionen im Stylesheet notiert werden muss.

Die Quelle der externen CSS-Datei wird über `url(...)` angegeben. Erwartet wird eine URI-Angabe. Die Datei kann also entweder lokal oder in Form einer absoluten Internetadresse referenziert werden.

Hinter der URL-Angabe kann sofort das abschließende Semikolon stehen. Es kann jedoch auch noch ein so genannter Medientyp angegeben werden. In diesem Fall wird angewiesen, dass die CSS-Definitionen aus der eingebundenen Datei nur für das angegebene Ausgabemedium gelten. Welche Ausgabemedien hier angegeben werden können, beschreiben wir weiter unten in Zusammenhang mit der @media-Regel.

6.3.2 Die @media-Regel

CSS erlaubt es, bei Formatdefinitionen nach Ausgabemedien zu unterscheiden, so beispielsweise die Unterscheidung nach Bildschirm- und Druckerausgabe. Ein Browser kann dann beim Anwenderwunsch, eine Seite auszudrucken, die Formate des print-Stylesheets verwenden, während für die Bildschirmanzeige die screen-Definitionen berücksichtigt werden. Gerade in Verbindung mit positionierten Elementen und mehreren Spalten kann es sinnvoll sein, für Ausdrucke ein eigenes Stylesheet zu entwickeln, welches die Elemente papierfreundlicher verteilt.

Daneben gibt es aber auch speziellere Medientypen, um der Vielfalt internetfähiger Geräte und auch Spezialausgabegeräten für Anwender mit Seh- oder anderen Schwächen gerecht zu werden. Die nachfolgende Tabelle listet die in der CSS-Spezifikation 2.1 vorgesehenen Medientypen auf. Spätere CSS-Versionen können weitere oder geänderte Medientypen enthalten. Maßgeblich dafür sind nicht zuletzt Entwicklungen auf dem Hardwaremarkt.

Medientyp	Bedeutung
`all`	Alle Medientypen (Default)
`braille`	Taktile Ausgabemedien mit Braille-Zeile
`embossed`	Seitenorientierte Braille-Drucker
`handheld`	Kleingeräte mit Display (Pocket-Computer, Handys, Tablet-PCs)
`print`	Drucker
`projection`	Beamer, Projektoren
`screen`	Übliche moderne farbige Computerbildschirme
`speech`	Sprach-Synthesizer (anstelle von `speech` ist auch `aural` möglich)

Tabelle 6.1: Medientypen in CSS

Medientyp	Bedeutung
tty	Textmodus-Ausgaben (Teletext, manche Handys, Textbrowser)
tv	Fernseher (eingeschränktes Scrollen, dafür soundfähig)

Tabelle 6.1: Medientypen in CSS (Fortsetzung)

Die @media-Regel wird als Block um die jeweils zugehörigen Formatdefinitionsblöcke gesetzt. Ein Beispiel:

```
@media print {
   body {
      background-color:white;
      color:black;
   }
   p,li,th,blockquote {
      font-family:Garamond, serif
      font-size:10pt;
   }
}
@media screen {
   body {
      background-color:black;
      color:white;
   }
   p,li,th,blockquote {
      font-family:Verdana, sans-serif;
      font-size: 10pt ;
   }
}
```

So wie im Beispiel notiert, könnte das Konstrukt in einer separaten CSS-Datei oder in einem style-Bereich innerhalb der HTML-Dateikopfdaten stehen.

Das Beispiel zeigt einige typische Formatdefinitionen, bei denen eine Unterscheidung zwischen Bildschirm (`@media print`) und Drucker (`@media print`) sinnvoll ist. Wenn bei der Bildschirmpräsentation helle Schriften auf dunklem Hintergrund verwendet werden, so ist es gut, für Druckausgaben dunkle Schriften auf weißem Grund anzugeben. Zwar verhindern die Browser zum Teil eigenmächtig, dass eine rabenschwarze Seite beim Ausdrucken eine halbe Tonerkartusche verbraucht, doch eine explizite Anweisung in CSS sorgt für Sicherheit. Auch Schriftarten und Schriftgrößen sind oft für die Bildschirmpräsentation optimiert. So ist die beliebte Microsoft-Schriftart Verdana vorwiegend für die Verwendung auf Websites entwickelt worden, weniger für Druckdokumente. Und Schriftgrößenangaben in Pixel sind ebenfalls bildschirmorientiert. Eine Unterscheidung nach Medientyp erlaubt es, für die Druckausgabe Punktangaben zur Schriftgröße zu notieren, was für die Bildschirmausgabe wiederum nicht empfehlenswert ist.

Beachten Sie bei der Syntax die verschachtelten Blöcke. Die @media-Regel schließt alles ein, was zwischen der öffnenden geschweiften Klammer hinter der Medientypangabe und ihrem Gegenstück, der schließenden geschweiften Klammer steht. Dazwischen befinden

sich typische Selektoren mit Formatdefinitionen, die ebenfalls wie üblich in geschweiften Klammern stehen, also eigene Blöcke darstellen.

Es ist auch erlaubt, mehrere Medientypen anzugeben, also z. B.:

```
@media screen, projection, tv { ... }
```

In diesem Fall gelten die zugeordneten Formatdefinitionen für alle genannten Gerätetypen.

6.3.3 Die @page-Regel

Das CSS-Boxmodell geht von einem *Viewport* aus. Bei der Ausgabe einer Webseite im Browser ist dies das aktuelle Anzeigefenster im Browser. Bei Ausgabe auf einen Drucker ist es – einen seitenorientierten Drucker, also keinen Endlosdrucker vorausgesetzt – eine Seite. Eine Seite kann jedoch unterschiedlich groß sein. So weichen beispielsweise das amerikanische US-Letter-Format und das vergleichbare deutsche A4-Format in ihren Längen- und Breitenverhältnissen voneinander ab. Ferner wird bei Druckseiten, die später gebunden werden sollen, in der Regel zwischen rechten und linken Seiten unterschieden. Für solche druckspezifischen Bedürfnisse steht die @page-Regel zur Verfügung.

Die Unterstützung der @page-Regel durch die Browser ist jedoch noch sehr dürftig. Die beste Unterstützung leistet derzeit der Opera-Browser.

Das nachfolgende Beispiel stellt einen möglichen Einsatz der @page-Regel vor:

```
@page {
   size:21.0cm 14.85cm;
   margin-top:2.2cm;
   margin-bottom:2.0cm;
}
@page :left {
   margin-left:1.7cm;
   margin-right:2cm
}
@page :right {
   margin-left:2cm;
   margin-right:1.7cm
}
```

Eine speziell für die @page-Regel geschaffene CSS-Eigenschaft ist `size`. Damit lässt sich innerhalb des Blocks einer @page-Regel die gewünschte Seitengröße definieren. Als Wert werden entweder zwei numerische Angaben erwartet, von denen die erste als Breite und die zweite als Höhe der Seite interpretiert wird. Oder Sie geben nur das allgemeine Seitenformat vor, nämlich `portrait` (Hochformat) oder `landscape` (Querformat). Bei numerischen Angaben sind printorientierte Einheiten wie `cm`, `mm` oder `inch` dringend zu empfehlen.

Über die Adressierung `@page:left` und `@page:right` können Sie zwischen linken und rechten Seiten beim Druck unterscheiden. Im obigen Beispiel werden die linken und rechten

Ränder entgegengesetzt definiert, was beim Ausdruck der Berücksichtigung eines Binderands (im Beispiel 0.3 Zentimeter) entspricht.

Weitere druckspezifische CSS-Formatierungen

Die folgenden CSS-Eigenschaften gehören nicht zur @page-Regel, sind jedoch in Zusammenhang mit printorientierter Ausgabe von Bedeutung. Vor allem innerhalb von Formatdefinitionen für den Medientyp `print` können solche Angaben sinnvoll sein. Die Eigenschaften können innerhalb aller üblichen Selektoren notiert werden.

Drei Eigenschaften namens `page-break-before`, `page-break-after` und `page-break-inside` erzwingen und verhindern Seitenumbrüche. Ein Beispiel:

```
@media print {
   h1 {
      font-family:Tahoma,sans-serif;
      font-size:15pt;
      page-break-before:always;
   }
   p, li, blockquote {
      font-family:inherit;
      font-size:10pt;
      page-break-inside:avoid;
   }
}
```

Die Eigenschaften `page-break-before` (Seitenumbruch vor diesem Element) und `page-break-after` (Seitenumbruch nach diesem Element) erlauben neben den Standard-Wertzuweisungen `auto` und `inherit` (Voreinstellung und »wie Elternelement«) die Angaben `always` (Seitenumbruch in jedem Fall erzwingen), `left` (Seitenumbruch erzwingen, und zwar so, dass die nächste Seite eine linke Seite ist), `right` (Seitenumbruch erzwingen, und zwar so, dass die nächste Seite eine rechte Seite ist) sowie `avoid` (Seitenumbruch verhindern). Bei `left` und `right` kann es vorkommen, dass zwei Seitenvorschübe eingefügt werden und eine Leerseite entsteht. Bei `page-break-inside` (Seitenumbruch innerhalb dieses Elements) ist neben den Standardzuweisungen nur `avoid` (Seitenumbruch innerhalb dieses Elements verhindern) erlaubt. Der Elementinhalt wird dann komplett auf die nächste Seite gedruckt. Auf der Seite davor bleibt unten je nach Umfang ein gewisser Leerraum.

Ebenfalls für seitenorientierte Ausgabemedien gedacht sind die beiden Eigenschaften `orphans` (alleinstehende Zeilen am Seitenende) und `widows` (alleinstehende Zeilen am Seitenanfang). Ein Beispiel:

```
@media print {
   p, li, blockquote {
      font-family:inherit;
      font-size:10pt;
      orphans:2;
      widows:2;
   }
}
```

In diesem Beispiel wird festgelegt, dass Fließtext der HTML-Elemente `p`, `li` und `blockquote`, falls innerhalb davon beim Ausdruck ein Seitenwechsel vorkommt, so umgebrochen werden sollte, dass wenigstens zwei Textzeilen am Ende der Seite und zwei Zeilen auf der neuen Seite stehen. Ist der Text kürzer, wird er komplett auf die neue Seite gedruckt. Der Wert für beide Eigenschaften ist also eine einfache Zahl, welche für die Anzahl Textzeilen steht.

6.3.4 Die @charset-Regel

Die @charset-Regel hat nichts mit dem HTML-Zeichensatz zu tun. Innerhalb von HTML-Dokumenten sollte die Angabe zum Zeichensatz deshalb stets wie üblich über die dafür vorgesehene Meta-Angabe erfolgen.

Die @charset-Regel ist nur für externe CSS-Dateien gedacht, damit ein auslesender Browser weiß, nach welchem Zeichensatz er die CSS-Datei dekodieren muss. Die @charset-Regel muss ganz zu Beginn der CSS-Datei notiert werden, z. B.:

```
@charset "UTF-8";
```

Innerhalb der Anführungszeichen muss ein Zeichenvorratsname aus dem IANA-Repertoire angegeben werden, also entweder eine Zeichenkodierungsform wie UTF-8 oder ein Zeichensatz wie beispielsweise ISO-8859-1. Die CSS-Datei muss im entsprechenden Kodierungsformat bzw. Zeichensatz abgespeichert sein.

6.4 CSS und die Browser

In Kapitel 5 haben Sie gelernt, wie moderne Webseitenlayouts ausschließlich über CSS erstellt werden. Nun gibt es ältere Browser, die vereinzelt noch zum Einsatz kommen, welche aber modernes CSS nicht so interpretieren wie spezifiziert oder nur einen Teil davon. Gemeint sind vor allem Netscape-Browser der 4er-Generation, Internet Explorer der 4er-Generation, teilweise auch noch der 5er-Generation, und Opera-Browser unter Version 6. Manchmal ist es sinnvoll, diese Browser mithilfe kleiner Tricks auszuschließen oder explizit anzusprechen. Diese Technik der so genannten *Browser-Weichen* versucht, Code bedingt ausführbar oder interpretierbar zu machen. Allein innerhalb von CSS gibt es einige Möglichkeiten dafür. In diesem Abschnitt werden wir einige wichtige davon vorstellen.

Schuld an der Misere sind übrigens nicht nur die (bösen, kommerziellen) Browser-Anbieter, wie von Hardlinern gerne behauptet wird. Auch die CSS-Spezifikation des W3-Konsortiums ist ein historisch gewachsenes Gebilde und war in früheren Jahren bei weitem nicht so präzise, wie sie es mittlerweile ist. Beschreibungen wie etwa das Boxmodell fehlten zunächst einfach. Der Interpretationsspielraum, den die Spezifikation ließ, führte bei den Browsern zu unterschiedlichen Implementierungen.

6.4.1 Netscape 4.x ausschließen

Wenn Sie überhaupt noch Wert darauf legen, auf Netscape-4-Browser Rücksicht zu nehmen, dann sollten Sie, falls Sie CSS massiv und vor allem als Layoutbasis einsetzen, diesen Browser völlig ausschließen und dafür sorgen, dass er nur nacktes HTML anzeigt, ebenso wie ein textbasierter Browser.

Dies geht recht einfach mithilfe der @import-Regel. Notieren Sie alle CSS-Formatdefinitionen in einer externen CSS-Datei. Binden Sie diese Datei in HTML nicht über das `<link>`-Tag ein (diese Syntax versteht Netscape 4 nämlich), sondern CSS-spezifisch (das ignoriert Netscape 4). Beispiel:

```
<style type="text/css">
   @import url(styles.css);
</style>
```

Diese Lösung ist in jedem Fall sauberer, als Netscape mit CSS-Definitionen zu konfrontieren, die er völlig fehlerhaft und teilweise mit grotesken Bugs darstellt. Vorausgesetzt, Ihre HTML-Dokumente enthalten ordentlich strukturiertes Markup, steht einer akzeptablen Präsentation in Netscape nichts im Wege.

Netscape-4-Browser mit eigenen Formaten bedienen

Wenn Sie sich die Mühe machen wollen, können Sie Netscape 4 doch noch mit etwas für ihn verdaulichem CSS versorgen. Binden Sie dazu zusätzlich zur Referenz via @import-Regel über das <link>-Tag eine andere CSS-Datei ein, die nur jene Formate enthält, die auch für Netscape unkritisch sind. Beispiel:

```
<link rel="stylesheet" type="text/css" href="simple.css">
<style type="text/css">
   @import url("advanced.css");
</style>
```

Mit diesem Code werden zwei CSS-Dateien eingebunden. Die Datei *simple.css* wird von allen Browsern interpretiert, die Datei *advanced.css* jedoch nicht von Netscape-4-Browsern. In der *simple.css* könnten beispielsweise Angaben zur Schriftformatierung stehen, die Netscape 4 noch leidlich umsetzt. Formatdefinitionen, die eine ordentliche Umsetzung des CSS-Boxmodells erfordern, wie Rahmen, Hintergrundgestaltung, Positionierung, Breiten, Höhen usw., sollten Netscape 4 dagegen unzugänglich bleiben und könnten in der *advanced.css* stehen.

6.4.2 Internet Explorer ausschließen und explizit ansprechen

Zurzeit der Drucklegung dieses Buchs ist der Internet Explorer der in der Szene am meisten verpönte Browser, leider aber auch noch der mit der größten Verbreitung. Verpönt ist er nicht nur wegen seiner engen Betriebssystemeinbindung, die ihn zu einem großen Sicherheitsrisiko für PCs werden ließ, sondern auch, weil er bei der Umsetzung der aktuellen HTML- und CSS-Standards des W3-Konsortiums noch einige ärgerliche Lücken und Fehler aufweist.

Wichtig ist auf jeden Fall, im HTML-Dokument eine gültige und vollständige Dokumenttyp-Deklaration zu notieren (inklusive DTD-URL-Angabe). Nur dadurch wird der Internet Explorer 6 dazu bewegt, vom so genannten *Quirks Mode* in den so genannten *Compliant Mode* umzuschalten. Im Quirks Mode benutzt der Internet Explorer ein älteres, microsoft-eigenes CSS-Boxmodell, das gewisse Abweichungen im Vergleich zu der CSS-Spezifikation aufweist. Die Ausdehnungssummen von Elementinhalt, Innenabstand, Rahmen und Außenabstand werden in diesem Modell anders berechnet und können daher bei Seitenlayouts zu unangenehmen Überraschungen führen. Der Compliant Mode ist dagegen der Schalter für eine standardkonforme Interpretation. Gesetzt wird der Schalter wie erwähnt, indem in HTML eine vollständige Angabe zum Dokumenttyp notiert wird.

Internet Explorer ausschließen

Einige neuere syntaktische Features bei Selektoren versteht der Internet Explorer bis einschließlich Version 6.0 nicht. Formatdefinitionen innerhalb von Selektoren mit solchen syntaktischen Elementen ignoriert er vollständig. Wenn Sie also Formate definieren möchten, die der Internet Explorer nicht interpretieren soll, dann müssen Sie einfach den Selektor so formulieren, dass dieser Browser ihn ignoriert. Ein Beispiel:

Selektor für alle Browser:

```
#kopf {
   position:fixed;
   top:0;
   left:0;
   width:100%;
   height:80px;
}
```

Selektor nicht für Internet Explorer:

```
body > #kopf {
   position:fixed;
   top:0;
   left:0;
   width:100%;
   height:80px;
}
```

Die #-Syntax, um via Selektor ein Element mit einem bestimmten id-Namen auszuwählen, kennt der Internet Explorer. Die Syntax, um mithilfe des >-Zeichens eine genaue Angabe zur Verschachtelungsregel zu notieren, kennt er hingegen nicht und er ignoriert sämtliche zugehörigen Formatdefinitionen. Befindet sich das Element mit dem id-Namen `#kopf` in der Elementhierarchie direkt unterhalb des body-Elements, also `<body> ... <element> </element> ... </body>`, so ist die zweitgenannte Selektor-»Formulierung« bedeutungsgleich mit der ersten. Im Gegensatz zur ersten wird sie jedoch vom Internet Explorer ignoriert, was sinnvoll ist, da dieser `position:fixed` nicht kennt.

Auch Selektoren mit attributbedingten Angaben wie `div[id]` schließen den Internet Explorer aus.

Eine weitere Möglichkeit, Internet Explorer mit einer Versionsnummer kleiner als 5 auszuschließen, besteht darin, alle zu verbergenden Formatdefinitionen in einen @media-Block zu packen. Ein Beispiel:

```
@media all {
   p:first-letter { font-size:250%; }
}
```

Da der Internet Explorer die @-Regeln bis Version 4 noch nicht kennt, ignoriert er sie. Die Angabe `@media all` (also »alle Medien«) hat für andere Browser jedoch den gleichen Effekt, wie wenn der @media-Block gar nicht da wäre.

Mit den beschriebenen Maßnahmen wird gleichzeitig auch Netscape 4.x ausgeschlossen.

Internet Explorer explizit ansprechen

Die zweifellos eleganteste Methode, Regelungen speziell für den Internet Explorer zu treffen, sind die genannten *Conditional Comments*. Dies sind HTML-Kommentare, deren Inhalt vom Internet Explorer auf Grund einer speziellen Syntax jedoch nicht ignoriert, sondern interpretiert wird. HTML-Kommentare beginnen normalerweise mit `<!--` und enden mit `-->`. Mit `<!--[if ...]>` und dem Gegenstück `<![endif]-->` können Sie jedoch HTML-Code speziell für den Internet Explorer oder sogar für spezielle Versionen davon notieren. Andere Browser behandeln den Inhalt dagegen als normalen HTML-Kommentar und ignorieren ihn. Die Syntax zur Formulierung der if-Bedingung erlaubt Ausprägungen wie `[if IE]` (wenn es irgendein Internet Explorer ist), `[if IE 6]` (wenn es ein Internet Explorer 6.0 ist), `[if gte IE 5]` (wenn es ein Internet Explorer Version größer gleich 5.0 ist) oder `[if lte IE 6]` (wenn es ein Internet Explorer mit einer Version unter 6.0 ist).

In Zusammenhang mit CSS erlauben es solche Kommentare, `<link>`-Tags zum Einbinden spezieller CSS-Dateien oder spezifische style-Bereiche zu notieren. Ein Beispiel:

```
<!--[if lte IE 6]>
   <link rel="stylesheet" type="text/css" href="ie-lte-6.css">
   <style type="text/css">
      div.kasten {
         width:60%;
         height:100px;
         background-color:yellow;
      }
   </style>
<![endif]-->
```

In diesem Beispiel wird sowohl eine CSS-Datei eingebunden als auch ein Style-Bereich definiert. Der Code steht jedoch innerhalb der Zeichenfolgen `<!-- -->` und wird deshalb eigentlich von allen Browsern ignoriert. Die spezielle Anschlusssyntax signalisiert dem

Internet Explorer jedoch, dass er den HTML-Code interpretieren soll, wenn es sich um eine Produktversion kleiner als 6.0 handelt.

Links zum Thema Browser-Weichen mit CSS:

Browser-Weichen und -Filter per CSS:
http://www.lipfert-malik.de/webdesign/tutorial/bsp/css-weiche-filter.html
Workshop Browser-Weiche:
http://www.css4you.de/wsbw/

Teil 3 – Dynamische Seiten mit JavaScript/DOM

JavaScript ist eine Programmiersprache, die sich ähnlich wie CSS in HTML integrieren lässt und wie CSS vom Browser ausgeführt wird. Sie erlaubt während der Anzeige einer Webseite beim Anwender dynamische Änderungen am Inhalt und Erscheinungsbild der Seite sowie einfache Benutzerinteraktion. Das Document Object Model (DOM) ist dabei der Standard für den dynamischen Zugriff auf HTML-Elemente, für die Steuerung von Anwenderereignissen usw.

7 Basiswissen JavaScript/DOM

Über JavaScript gibt es allein Regale voll Fachbücher. Was die tatsächliche Bedeutung dieser Sprache im professionellen Bereich der Website-Erstellung betrifft, ist jedoch ein deutlicher Rückgang in den letzten Jahren zu verspüren. Dafür gibt es verschiedene Gründe:

- Vieles, was einst mit JavaScript versucht wurde zu lösen, wird heute über ***Flash*** gelöst, wie bewegliche Newsticker, Intros und Trailer mit oder ohne Sound, oder besondere Effekte bei der Navigation. Nachdem die meisten modernen Browser mit Flash-Plug-In ausgeliefert werden, haben viele Webdesigner JavaScript in diesen Disziplinen den Rücken gekehrt.
- JavaScript ist uneinheitlich implementiert. Die Unterschiede vor allem bei etwas älteren Browsern sind so groß, dass in vielen Fällen einiger Programmieraufwand dafür erforderlich ist, Fehler in bestimmten Browsern oder Browser-Versionen zu vermeiden.
- Der W3C-Standard des DOM ist recht komplex und kommt für JavaScript zu spät. Es war zwar im Prinzip eine richtige Entscheidung, JavaScript so weit wie möglich an diesen Standard anzubinden. Das DOM ist jedoch auf XML ausgelegt und seine Entwicklung ist noch nicht abgeschlossen. Viele Webdesigner, die »mal eben« etwas »Dynamisches« programmieren wollen, scheitern zudem am Verständnis der Zusammenhänge zwischen herkömmlichem JavaScript-DOM, Kern-JavaScript und W3C-DOM.
- JavaScript hat keinen astreinen Ruf. In der Vergangenheit wurde die Sprache tausendfach missbraucht, um ungebetene Popups zu öffnen, die Browser-Statuszeile ohne Einverständnis des Benutzers mit Unsinn vollzuschreiben oder das Browser-Fenster einfach auf eine bestimmte, vom »Designer« gewünschte Größe zu ändern. JavaScript geriet in den Ruf, ein Mittel zu sein, mit dem unsensible Homepage-Künstler und Werbestrategen die Seitenbesucher quälen.

Die genannten Nachteile sind der Grund dafür, warum wir JavaScript im vorliegenden Buch nicht zu viel Platz einräumen wollen. Dennoch lohnt es sich auch heute noch, sich mit JavaScript und DOM zu beschäftigen. Für Webdesigner, die »aus allen Positionen schießen« können wollen, sind Kenntnisse in JavaScript und DOM nach wie vor Pflicht.

Es gibt auch durchaus genügend Anwendungsfälle, für die JavaScript bestens geeignet ist. Dazu gehören beispielsweise:

- Überprüfen von Formulareingaben vor dem Absenden des Formulars
- Komfort-Features für Besucher wie Style-Switching oder einblendbare Zusatzinfos

- Kleinanwendungen wie Umrechner, Zinsrechner oder Kalender, die den Anwender beim eigentlichen Inhalt einer Seite unterstützen
- Präsentation mehrteiliger Inhalte ohne langes Scrollen oder Aufruf anderer Seiten

Anwendungen solcher Art kommen auch auf professionellen Sites häufig zum Einsatz. Im Praxiskapitel 8 werden wir uns deshalb auf solche Anwendungen konzentrieren. Im vorliegenden Kapitel 7 wird das notwendige Handwerkszeug vermittelt. Wir konzentrieren uns dabei auf zeitgemäßes, DOM-kompatibles JavaScript ohne historischen Ballast.

Falls Sie bislang zwar HTML und CSS kennen, aber noch nie programmiert haben, werden Sie in diesem Kapitel mit neuen, ungewohnten Konzepten konfrontiert. Zusätzlich zu den syntaktischen Konzepten der Sprache sind auch Kombinatorik und die Fähigkeit, Algorithmen zur Lösung von Aufgaben zu finden, gefragt.

Link zum Thema JavaScript-Einsatz:

JavaScript – ja oder nein?:
http://www.frixon.de/docs/javascript.html

7.1 Implementierungen von JavaScript und DOM

JavaScript gehört zur Gattung der ***Scriptsprachen***. Der Programmierer notiert den Programmcode in einer High-Level-Sprache, die für Menschen recht gut lesbar ist, aber syntaktisch so präzise Regeln hat, dass der im Browser integrierte JavaScript-Interpreter den Code zur Laufzeit in ausführbaren Maschinencode umwandeln kann.

DOM ist dagegen keine bestimmte Sprache, sondern ein sprachunabhängiges Modell, das Objekte, Eigenschaften und Methoden definiert, mit deren Hilfe auf XML- und HTML-Dokumente zugegriffen werden kann, aber auch Anwenderereignisse wie Mausklicks und Tastatureingaben verarbeitet werden können.

7.1.1 JavaScript

JavaScript wurde ursprünglich von Netscape mit dessen Browser-Version 2.0 eingeführt. Das war 1995, als das W3-Konsortium noch in seinen Kinderschuhen steckte und weit entfernt davon war, von allen Seiten als das Standardisierungsgremium für Webtechnologien akzeptiert zu werden. Um der Sprache den Rang eines Standards zu verleihen, schaltete Netscape die European Computer Manufacturers Association (ECMA) ein. Unter der Bezeichnung ECMA-262 ist JavaScript seitdem als Industriestandard definiert.

Microsoft erwarb zwar frühzeitig eine JavaScript-Lizenz, durfte für die eigene Implementierung jedoch nicht den Namen »JavaScript« verwenden. Außerdem wollten die Redmonder eine Scriptsprache, die auch andere Aufgaben wahrnimmt und sich in den ***Windows Scripting Host*** integriert. Die Scriptsprache wurde ***JScript*** getauft. Außerdem interpretiert der Internet Explorer noch das ältere Visual Basic Script (VBScript). Während VBScript jedoch eine eigene Syntax hat, sind Syntax und Teile der Objektstruktur von

JScript so weit deckungsgleich mit JavaScript, dass es genügt, in HTML einen Script-Bereich als Bereich für JavaScript auszuzeichnen. Der Internet Explorer interpretiert den Code.

Die aktuelle Version von JavaScript ist 1.5. Eine Version 2.0 ist in Entwicklung. Es ist jedoch fraglich, ob die zahlreichen Neuerungen der Version 2, die vor allem das Konzept der Objektorientierung wesentlich konsequenter umsetzt als bislang der Fall, in der Praxis überhaupt Erfolg haben werden.

Die stabile Sprachstandard-Version 1.5 entstand im Jahr 2000 und wurde gemeinsam von Netscape und Sun für den völlig neuen, auf der Gecko-Engine basierenden Netscape-6-Browser entwickelt. Entscheidend in dieser Version war die Reduktion von JavaScript auf einen relativ überschaubaren JavaScript-Kern mit nur wenigen Objekten, Methoden und Eigenschaften. Für die übrigen Angelegenheiten vom Zugriff auf HTML-Inhalte bis hin zum Event-Handling sollten JavaScript-Interpreter einfach das vom W3-Konsortium spezifizierte DOM umsetzen.

Links zu JavaScript-Spezifikationen:

JavaScript 1.5 Spezifikation:
http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf
JavaScript 1.3 Reference:
http://docs.sun.com/app/docs/doc/816-6408-10
JavaScript 1.3 Guide:
http://docs.sun.com/app/docs/doc/816-6409-10

7.1.2 DOM (Document Object Model)

Das DOM ist wie HTML und CSS ein vom W3-Konsortium entwickelter Standard und hat damit eine hohe Bindungskraft im Web. Es ist sowohl für client- als auch für serverseitige Programmierung einsetzbar. Gedacht ist es als Ergänzung zum XML-Universum. Während Markup-Sprachen der XML-Familie Datenstrukturen auszeichnen, stellt das DOM eine Schnittstelle für Programmiersprachen zur Verfügung, um auswählbare Einzeldaten oder Datengruppen in XML-Dokumenten lesen, ändern und löschen zu können, um ihnen Style-Definitionen aus Sprachen wie CSS oder XSL zuordnen zu können und um Ereignisse zu behandeln, die mit diesen Daten geschehen können. Herkömmliches HTML (also nicht XHTML) ist zwar keine XML-basierte Sprache, doch auch hierauf ist der Zugriff über das DOM möglich. Fast alle modernen, im Web eingesetzten Programmiersprachen nutzen die Schnittstelle, die das DOM bietet.

Die aktuell als Empfehlung (***Recommendation***) vorliegende DOM-Version ist 3.0. Vieles davon ist jedoch in Zusammenhang mit JavaScript gar nicht von Bedeutung. Die wichtigsten Methoden und Eigenschaften für den Zugriff auf einzelne HTML-Elemente und -Attribute waren schon Bestandteil von DOM 1.0 und die DOM-Vorgaben für den Zugriff auf CSS-Definitionen und Event-Handling sind seit DOM-Version 2.0 standardisiert.

Die JavaScript-Interpreter der verschiedenen Browser unterstützen das DOM sicher nicht vollständig und einheitlich. Für die Praxis genügt das, was Internet Explorer 6.0 (mit Einschränkungen auch schon Internet Explorer 5.x), Gecko-Browser (wie Netscape ab Version 6.0, Mozilla, Firefox usw.) und Opera ab Version 7.0 bieten, jedoch in der Regel.

Das spezielle Problem bei JavaScript ist, dass es – historisch bedingt – mehrere Objektmodelle mit sich herumschleppt, die parallel implementiert sind. Da die klassischen Objektmodelle von Netscape und Microsoft in Dokumentationen und Büchern auch immer noch vielfach wie aktuelle Standards angepriesen werden, werden sie immer noch viel ernster genommen, als sie es eigentlich sind. Moderne Webseiten sollten bei Verwendung von JavaScript jedoch wenn irgend möglich nur noch DOM-kompatibel arbeiten. Deshalb werden wir in diesem Buch auch keine Rücksicht mehr auf die veralteten, seit mehreren Jahren überholten Objektmodelle nehmen. Alle Beispiele werden so weit möglich die DOM-gerechte Syntax verwenden.

Link zum Thema DOM:

W3C-Startseite zum Thema DOM:
http://www.w3.org/DOM/

7.2 JavaScript in HTML

Ähnlich wie bei CSS gibt es drei Stellen, an denen JavaScript-Code vorkommen kann:

- Innerhalb von HTML-Elementen, genauer, als Wertzuweisung an ein Event-Handler-Attribut
- In Script-Bereichen innerhalb des HTML-Dokuments, markiert durch das dafür vorgesehene script-Element
- In separaten Textdateien mit der Standardendung *.js*, die in HTML referenziert werden können

Anhand von einfachen Beispielen werden wir die drei Varianten vorstellen.

7.2.1 Event-Handler und JavaScript

Ein Event-Handler kann in einleitenden HTML-Tags in der Form eines Attributs notiert werden. Als Wert wird einem solchen Attribut JavaScript-Code zugewiesen. Im nachfolgenden Beispiel erhält ein p-Element zwei Event-Handler: `onmouseover` und `onmouseout`.

```
<p style="cursor:pointer"
   onmouseover="this.innerHTML='Hallo Maus!'"
   onmouseout="this.innerHTML='Komm zu mir, Maus!'">
     Komm zu mir, Maus!
</p>
```

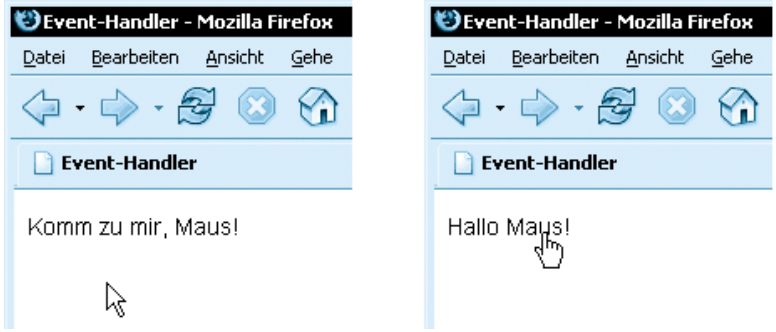


Abbildung 7.1: p-Element mit Text, der sich beim Überfahren mit der Maus ändert

Der Event-Handler `onmouseover=` bedeutet: »wenn und solange der Mauszeiger sich über der Elementbox befindet«. Diese Formulierung legt auch schon nahe, in welchen HTML-Elementen dieser Event-Handler notiert werden kann: in allen Elementen, die zumindest potenziell eine sichtbare Ausdehnung im Browser-Fenster haben. Es kann sich also um Überschriften, Textabsätze, Tabellenzellen, Listenpunkte, aber auch um Inline-Elemente wie Hyperlinks, betonten Text oder um Grafikreferenzen handeln. Das Gleiche gilt für den Event-Handler `onmouseout=` (»wenn die Maus aus der Elementbox herausbewegt wird«).

Im obigen Beispiel hat der Textabsatz in HTML als Elementinhalt den Text `Komm zu mir, Maus!`. Wird der Mauszeiger in den Bereich bewegt, den die Elementbox des Absatzes einnimmt, wird der JavaScript-Code ausgeführt, der beim Event-Handler `onmouseover=` notiert ist. Dieser Code besteht im Beispiel aus einer einzigen Anweisung, welche den Elementinhalt des Absatzes gegen einen anderen Text (`Hallo Maus!`) austauscht.

Der geänderte Text würde nun einfach so stehen bleiben. Wir möchten jedoch, dass der Ursprungstext wiederhergestellt wird, wenn der Anwender die Maus aus der Elementbox wieder herausbewegt. Das entsprechende Ereignis kann mit dem Event-Handler `onmouseout=` abgefangen werden. Der JavaScript-Code, der bei Eintritt dieses Ereignisses ausgeführt wird, tauscht den Elementinhalt abermals aus, diesmal jedoch mit dem ursprünglichen Text.

Da der ursprüngliche Text in JavaScript zwischenzeitlich nirgendwo gespeichert war, muss er bei `onmouseout=` explizit noch mal angegeben werden. Das ist der Grund, warum der Text `Komm zu mir, Maus!` insgesamt zweimal vorkommt: einmal als normaler Elementinhalt und einmal zur »Wiederherstellung« bei `onmouseout`.

Auf die Syntax des JavaScript-Codes gehen wir an dieser Stelle noch nicht detailliert ein. Nur so viel zum Verständnis: Über das Schlüsselwort `this` ist es in JavaScript möglich, auf ein »aktuelles Objekt« zuzugreifen, und `innerHTML` ist eine Objekteigenschaft, nämlich der Inhalt eines HTML-Elements. Die Objekteigenschaft `innerHTML` ist übrigens eine »schmutzige« Eigenschaft, die nicht ganz der DOM-Philosophie entspricht und nicht zum Standard gehört bzw. nur zum so genannten »DOM Level 0«, welches speziell für den Zugriff auf HTML entwickelt wurde. Die innerHTML-Eigenschaft ist jedoch sehr praktisch und in allen modernen Browsern implementiert, so dass es unsinnig wäre, sie nicht zu verwenden.

Im Beispiel wird der innerHTML-Eigenschaft durch das Gleichheitszeichen ein Wert zugewiesen. Das ganze Konstrukt ist eine einzelne JavaScript-Anweisung.

Das Beispiel enthält neben den Event-Handlern im style-Attribut noch eine CSS-Eigenschaft, die wir bislang nicht behandelt haben: `cursor`. Diese Eigenschaft erlaubt es, den Mauszeiger zu verändern. Sinnvoll ist dies in aller Regel aber nur in Verbindung mit Script-Reaktionen. Mit `cursor:pointer` wird eine Art Mauszeiger erzeugt, wie er beim Überfahren von Hyperlinks verwendet wird. Die Cursor-Definition gilt jedoch nur für das p-Element. Der Browser ändert den Mauszeiger also nur, solange dieser sich über dem Elementbereich des p-Elements befindet, und stellt anschließend automatisch den Normalzustand wieder her. Weitere mögliche Angaben zur Cursor-Eigenschaft finden Sie in der CSS-Referenz im hinteren Buchteil.

Event-Handler in HTML

Da die Event-Handler als Attribut in HTML notiert werden, regelt die HTML-Spezifikation auch, welche Event-Handler-Attribute es gibt und in welchen HTML-Elementen sie vorkommen dürfen. Einige Browser, allen voran der Internet Explorer, unterstützen daneben zahlreiche weitere Event-Handler, die jedoch an JScript orientiert und proprietär sind. Die nachfolgende Tabelle listet die offiziellen Event-Handler und ihre Einsatzmöglichkeiten auf.

Event-Handler	Bedeutung	Vorkommen
`onload=`	Ereignis tritt ein, wenn das Laden eines Dokuments oder Framesets abgeschlossen ist.	Im body-Element oder in einem frameset-Element.
`onunload=`	Ereignis tritt ein, wenn ein angezeigtes Dokument oder Frameset verlassen wird, z. B. durch Hyperlink oder Schließen des Fensters.	Im body-Element oder in einem frameset-Element.
`onclick=`	Ereignis tritt ein, wenn mit der Maus in den Elementbereich geklickt wird.	In allen Elementen innerhalb des body-Elements, außer in bdo, br, iframe, param und script.
`ondblclick=`	Ereignis tritt ein, wenn mit der Maus ein Doppelklick im Elementbereich erfolgt.	In allen Elementen innerhalb des body-Elements, außer in bdo, br, iframe, param und script.
`onmousedown=`	Ereignis tritt ein, wenn die Maustaste im Elementbereich länger gedrückt gehalten wird.	In allen Elementen innerhalb des body-Elements, außer in bdo, br, iframe, param und script.
`onmouseup=`	Ereignis tritt ein, wenn die Maustaste über dem Elementbereich gedrückt war und nun losgelassen wird.	In allen Elementen innerhalb des body-Elements, außer in bdo, br, iframe, param und script.
`onmouseover=`	Ereignis tritt ein, wenn der Mauszeiger in den Elementbereich hinein bewegt wird, ohne dass eine Maustaste gedrückt wird.	In allen Elementen innerhalb des body-Elements, außer in bdo, br, iframe, param und script.

Tabelle 7.1: Mögliche Event-Handler in HTML-Elementen

Event-Handler	Bedeutung	Vorkommen
onmousemove=	Ereignis tritt ein, wenn der Mauszeiger, während er sich über dem Elementbereich befindet, innerhalb des Elementbereichs bewegt wird.	In allen Elementen innerhalb des body-Elements, außer in bdo, br, iframe, param und script.
onmouseout=	Ereignis tritt ein, wenn der Mauszeiger aus dem Elementbereich herausbewegt wird.	In allen Elementen innerhalb des body-Elements, außer in bdo, br, iframe, param und script.
onfocus=	Ereignis tritt ein, wenn das Element durch Anklicken oder Tastaturzugriff den Fokus erhält.	Nur in den Elementen a, area, button, input, label, select und textarea.
onblur=	Ereignis tritt ein, wenn ein Element den Fokus verliert, weil sich der Fokus durch Klick oder Tastaturzugriff geändert hat.	Nur in den Elementen a, area, button, input, label, select und textarea.
onkeypress=	Ereignis tritt ein, wenn eine Tastaturtaste bei einem definierten Element gedrückt und wieder losgelassen wird.	In allen Elementen innerhalb des body-Elements, außer in bdo, br, iframe, param und script.
onkeydown=	Ereignis tritt ein, wenn eine Tastaturtaste bei einem definierten Element gedrückt, aber noch nicht wieder losgelassen wird.	In allen Elementen innerhalb des body-Elements, außer in bdo, br, iframe, param und script.
onkeyup=	Ereignis tritt ein, wenn eine Tastaturtaste bei einem definierten Element wieder losgelassen wird.	In allen Elementen innerhalb des body-Elements, außer in bdo, br, iframe, param und script.
onsubmit=	Ereignis tritt ein, wenn ein Formular abgesendet wird.	Nur im form-Element.
onreset=	Ereignis tritt ein, wenn ein Formular auf seine Ausgangswerte zurückgesetzt wird.	Nur im form-Element.
onselect=	Ereignis tritt ein, wenn in einem Eingabefeld Text selektiert wird.	Nur bei den Elementen input und textarea.
onchange=	Ereignis tritt ein, wenn ein Formularelement den Fokus verliert und einen anderen Wert hat als beim Erhalt des Fokus – kurz, wenn sich der Wert des Elements geändert hat.	Nur bei den Elementen input, select und textarea.

Tabelle 7.1: Mögliche Event-Handler in HTML-Elementen (Fortsetzung)

Event-Handler können also der Auslöser für die Ausführung von JavaScript-Code sein. Der Code wird ausgeführt, wenn das entsprechende Ereignis eintritt.

7.2.2 JavaScript-Bereiche in HTML

Es ist durchaus möglich, in der Wertzuweisung an einen Event-Handler mehrere JavaScript-Anweisungen auszuführen. Doch Wertzuweisungen an HTML-Attribute eignen sich nur begrenzt für längere Anweisungssequenzen. Dazu sind Script-Bereiche besser geeignet. In einem Script-Bereich können unter anderem auch Funktionen stehen, die mehrere Anweisungen ausführen. Bei der Wertzuweisung an einen Event-Handler genügt

es dann, als einzige Anweisung die gewünschte Funktion aufzurufen. Deren Anweisungen werden dann ausgeführt.

Ein Script-Bereich für JavaScript wird in HTML folgendermaßen ausgezeichnet:

```
<script type="text/javascript">
</script>
```

Der Script-Inhalt wird durch `<script>...</script>` eingeschlossen. Pflicht ist die Angabe des type-Attributs. Diesem Attribut wird als Wert der Mime-Type der verwendeten Scriptsprache zugewiesen. Für JavaScript lautet der Mime-Type `text/javascript`.

Wenn der Script-Inhalt keinen HTML-Output erzeugt und keine dynamischen Änderungen im HTML-Strukturbaum vornimmt, ist es sinnvoll, im einleitenden <script>-Tag zusätzlich das Standalone-Attribut `defer` zu notieren (Notation bei XHTML: `defer="defer"`). Dieses Attribut teilt dem Browser mit, dass er mit der Darstellung des gerade eingelesenen Dokuments fortfahren kann, ohne erst den Scriptinhalt interpretiert zu haben. Das kann in manchen Fällen dazu beitragen, dass eine Seite wahrnehmbar schneller angezeigt wird.

Die XHTML-Problematik bei Script-Bereichen

Falls Sie XHTML einsetzen, ist es zweckmäßig, den Inhalt des script-Elements mit der dafür vorgesehenen XML-Syntax als CDATA-Bereich auszuzeichnen. CDATA steht für *character data* und weist den Parser an, dass HTML-eigene Zeichen innerhalb eines solchen Bereichs nicht als HTML-Syntax interpretiert werden. Das Konstrukt lautet dann:

```
<script type="text/javascript">
//<![CDATA[
//]]>
</script>
```

Die Zeichenfolge `<![CDATA[` leitet in XML einen CDATA-Bereich ein. Alles bis zur Zeichenfolge `]]>` wird dann als »frei von Markup« behandelt. Die einleitende Zeichenfolge und die beendende Zeichenfolge sollten je in einer eigenen Zeile notiert werden. Beiden Zeichenfolgen sollten außerdem wie im Beispiel gezeigt je zwei Schrägstriche `//` vorangestellt werden. Dadurch wird verhindert, dass der JavaScript-Interpreter die XML-spezifischen Zeichenfolgen als JavaScript-Code interpretiert, was aus JavaScript-Sicht Fehler produzieren würde. Die beiden Schrägstriche bedeuten in JavaScript-Bereichen einen Kommentar. Alles bis zum Zeilenende wird vom JavaScript-Interpreter ignoriert.

Der Hintergrund ist, dass Script-Bereiche in HTML automatisch als CDATA gelten. In XHTML ist der Elementinhalt des script-Elements jedoch als PCDATA definiert, was aber wie bei Text innerhalb anderer Elemente bedeutet, dass Zeichen wie < und `&` maskiert werden müssen. Diese Zeichen kommen in JavaScript jedoch öfter vor. Mit der maskierten Form wie `<` könnte der JavaScript-Interpreter jedoch nichts anfangen und würde einen Fehler melden.

Leider ist die Problematik damit noch nicht zu Ende. Einige Browser verstehen die XML-Notation für CDATA-Bereiche nicht, was dann doch JavaScript-Fehler produziert. Das W3-Konsortium und auch andere Fachleute empfehlen daher, JavaScript-Code bei XHTML grundsätzlich in externe Dateien auszulagern und diese einzubinden. Ein Beispiel:

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

Die Einbindung erfolgt ebenfalls über das script-Element und zwar im src-Attribut. Angegeben werden kann ein beliebiger URI. Die JavaScript-Datei ist eine normale Textdatei mit JavaScript-Code und sollte die Standardendung *.js* haben. Das script-Element sollte selbst keinerlei Inhalt haben. Auf diese Weise lässt sich das Problem für alle Browser beheben.

Event-Handler und Script-Bereich

Mit dem nachfolgenden Beispiel soll das typische Zusammenspiel zwischen Event-Handler und Script-Bereich deutlicher werden.

Listing 7.1: HTML-Dokument mit Script-Bereich, Script und Event-Handler

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Script-Bereich</title>
<script type="text/javascript">
   function tell_result() {
      var population = 0;
      var city = document.getElementById('city');
      switch(city.options[city.options.selectedIndex].value) {
          case "Hamburg":
             population = "1,7 Mio. Einwohner"; break;
          case "Berlin":
             population = "3,4 Mio. Einwohner"; break;
          case "München":
             population = "1,2 Mio. Einwohner"; break;
          default: break;
      }
      document.getElementById('result').innerHTML = population;
   }
</script>
</head>
<body>
<form action="javascript:void(0)"
      style="background-color:#FFDDAA; padding:10px;">
<p>Wählen Sie eine Stadt aus:</p>
<select id="city" size="3" style="width:120px;">
<option value="Hamburg">Hamburg</option>
<option value="Berlin">Berlin</option>
```

```
<option value="München">München</option>
</select><br>
<button type="button" style="width:120px; border:solid 1px gray"
        onClick="tell_result()">Ergebnis</button>
<p id="result"> </p>
</form>
</body>
</html>
```

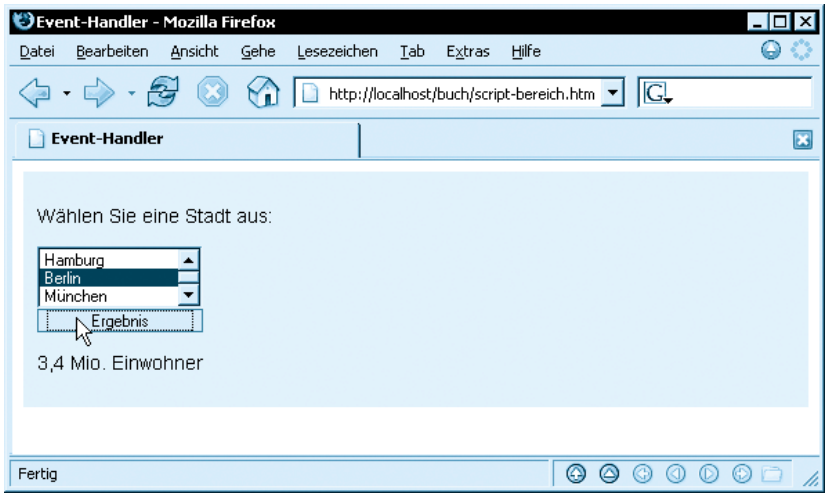


Abbildung 7.2: Städte-Einwohner-Script und seine Wirkung

Das Beispiel ist ein Anwendungsfall der Sorte »Information interaktiv und nicht linear vermitteln«. Im Beispieldokument kann der Anwender in einem Formular aus einer Auswahlliste eine Stadt auswählen. Klickt er auf den Button mit der Aufschrift ***Ergebnis***, wird die Einwohnerzahl der Stadt unterhalb des Buttons ausgegeben.

Betrachten wir dazu zunächst das HTML-Formular mit der Auswahlliste und dem Button ***Ergebnis***. Im einleitenden `<form>`-Tag fällt das action-Attribut mit der ungewöhnlichen Wertzuweisung `javascript:void(0)` auf. Eigentlich möchten wir bei diesem Formular überhaupt keine »Aktion«. Da das action-Attribut aber HTML-seitig ein Pflichtattribut ist, muss es, um das Dokument nicht syntaktisch fehlerhaft werden zu lassen, angegeben werden. Die Wertzuweisung ist zwar vom Typ her ein URI, doch `javascript:` ist kein offizielles Schema. Die Browser erkennen es jedoch und wissen, dass in diesem Fall keine andere Seite aufgerufen werden muss, sondern die aktuelle Seite geladen bleibt. Allerdings wird auch der JavaScript-Interpreter aktiv, denn hinter `javascript:` kann ähnlich wie bei der Wertzuweisung an einen Event-Handler JavaScript-Code notiert werden. Die Anweisung `void(0)` bewirkt einfach gar nichts. Das Konstrukt `action="javascript:void(0)"` ist also empfehlenswert für Formulare, die ausschließlich mit JavaScript zusammenarbeiten.

Weiterhin beachtenswert ist im HTML-Quelltext das p-Element mit dem id-Namen `result` im unteren Dokumentabschnitt. Das Element hat lediglich ein erzwungenes Leerzeichen (HTML-Entity ` `) als Inhalt. Der Grund ist, dass die Elementbox zwar schon in der

Höhe einer Textzeile angezeigt werden, zunächst jedoch noch keinen sichtbaren Inhalt enthalten soll. Der Inhalt wird mithilfe von JavaScript dynamisch in das Element geschrieben.

Die Schnittstelle zwischen dem Formular und JavaScript ist der Button mit der Aufschrift *Ergebnis* (markiert durch `<button> ... </button>`). In seinem einleitenden Tag enthält dieses Element ein Event-Handler-Attribut `onclick=`. Die Wertzuweisung lautet ganz schlicht `tell_result()`. An den runden Klammern ist erkennbar, dass es sich dabei um einen JavaScript-Funktionsaufruf handelt.

Auf die Einzelheiten der JavaScript-Syntax werden wir auch in diesem Beispiel noch nicht näher eingehen. Wichtig ist zunächst nur, dass Sie die Verknüpfung von Event-Handler und ausgeführtem JavaScript-Code erkennen.

Im Kopfbereich des HTML-Dokuments, also zwischen `<head>` und `</head>`, ist ein script-Element notiert. Als Inhalt enthält es JavaScript-Code.

Der Code besteht aus der Funktion `tell_result()`. Alles, was zwischen der öffnenden geschweiften Klammer hinter dem Funktionsnamen und der schließenden geschweiften Klammer in der letzten Zeile vor dem schließenden `</script>`-Tag steht, gehört zu dieser Funktion. JavaScript-Anweisungen, die in einem script-Bereich außerhalb jeder Funktion stehen, werden direkt beim Einlesen des Dokuments ausgeführt. Alles, was wie im Beispiel innerhalb einer Funktion steht, wird dagegen nur dann ausgeführt, wenn die Funktion explizit aufgerufen wird. Das ist im Beispiel der Funktion `tell_result()` bei Eintritt des Event `onClick=` beim Button *Ergebnis* der Fall, also dann, wenn der Anwender auf diesen Button klickt.

Die Funktion `tell_result()` wertet aus, welcher der drei Listeneinträge bei den Städtenamen ausgewählt ist. Abhängig vom ausgewählten Eintrag schreibt sie mit der bereits bekannten innerHTML-Eigenschaft die Einwohnerzahl der ausgewählten Stadt.

Der richtige Ort für JavaScript-Bereiche

In unserem Beispiel steht das script-Element innerhalb der Kopfdaten des HTML-Dokuments. Das ist auch der übliche Ort, wenn das Script vorwiegend aus Funktionen besteht, die erst bei Eintritt von Events oder von anderen Funktionen aufgerufen werden oder wenn es Anweisungen enthält, die zwar direkt beim Einlesen ausgeführt werden, aber keine Ausgaben ins Browser-Fenster schreiben.

Das script-Element darf jedoch auch innerhalb des body-Elements und beliebiger Block- und Inline-Elemente vorkommen. Innerhalb des body-Bereichs sollte ein script-Bereich jedoch nur dann notiert werden, wenn sein Code direkt beim Einlesen ausgeführt wird und wenn dieser entweder direkt an der betreffenden Stelle HTML-Code erzeugen soll oder wenn er sich auf ein Element im Dokumentbaum bezieht, das dem Browser zum Zeitpunkt der Scriptausführung bereits bekannt sein muss. Ein Beispiel soll dies verdeutlichen:

Listing 7.2: Script-Bereich im body-Bereich

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<meta id="author" name="author" content="Stefan Münz">
<title>Script-Bereiche</title>
</head>
<body style="background-color:#FFDDAA">
<h1>Überschrift</h1>
<p style="border-bottom:solid 1px black;
          padding-bottom:15px;">Text</p>
<p id="url_and_author" style="font-style:italic">
<script type="text/javascript">
   var url = document.URL;
   var author =
      document.getElementById('author').getAttribute('content');
   var now = new Date();
   var year = now.getFullYear();
   document.write(url + "<br>" + author + ", " + year);
</script>
</p>
</body>
</html>
```

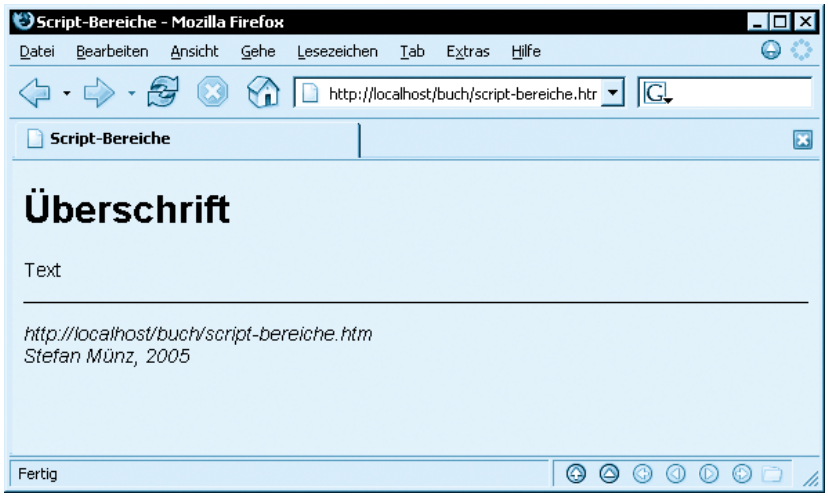


Abbildung 7.3: Vom Script generierter Inhalt im Fußbereich

Das HTML-Dokument enthält in den Kopfdaten unter anderem eine Meta-Angabe zum Autor der Seite. Diese Meta-Angabe erhält den id-Namen `author`. Im body-Teil enthält das Dokument zunächst etwas Text, bestehend aus einer Überschrift und einem Textabsatz.

Dann folgt ein zweiter Textabsatz, also ein p-Element. Dieses enthält jedoch keinen statischen Text als Elementinhalt, sondern einen Script-Bereich.

Der Code innerhalb dieses Bereichs steht nicht in einer Funktion, wird also direkt beim Einlesen des Dokuments vom JavaScript-Interpreter ausgeführt. Das Script ermittelt zunächst den aktuellen URI des Dokuments mit der Objekteigenschaft `document.URL`. Dann ermittelt es den Autorennamen aus dem Wert, der beim entsprechenden Meta-Tag dem content-Attribut zugewiesen wurde. Und schließlich ermittelt es noch das aktuelle Kalenderjahr. Mithilfe der Objektmethode `document.write()` schreibt das Script am Ende die ermittelten Daten ins Dokument an die aktuelle Stelle, also in den Elementinhalt des p-Elements, welches den script-Bereich umschließt.

7.2.3 JavaScript in separaten Dateien

Für XHTML ist es, wie bereits beschrieben, die sauberste Lösung, JavaScript-Code nicht direkt als Elementinhalt zwischen `<script>` und `</script>` zu notieren, sondern besser in separaten Dateien, die im einleitenden `<script>`-Tag referenziert werden können. Aber auch bei Verwendung von SGML-basiertem HTML ist es häufig besser, JavaScript-Code auszulagern. Das gilt vor allem für Code, der auf mehreren oder gar allen Seiten benötigt wird. Dazu ein Beispiel. Zunächst der Inhalt einer separaten JavaScript-Datei:

Listing 7.3: JavaScript-Datei mit Funktion für Datumsausgabe

```
function show_date(id_name) {
   var weekdays =  new Array("Sonntag","Montag","Dienstag",
                   "Mittwoch","Donnerstag","Freitag","Samstag");
   var now = new Date();
   var day = now.getDate();
   var month = now.getMonth() + 1;
   var year = now.getFullYear();
   var weekday = now.getDay();
   var day0  = ((day < 10) ? "0" : "");
   var month0  = ((month < 10) ? ".0" : ".");
   var today = day0 + day + month0 + month + "." + year;
   var date_to_show = weekdays[weekday] + ", " + today;
   document.getElementById(id_name).innerHTML = date_to_show;
}
```

Eine JavaScript-Datei ist eine einfache Textdatei, die nichts anderes als JavaScript-Code enthalten darf. In unserem Beispiel enthält sie nichts anderes als eine Funktion namens `show_date()`. Die Funktion erwartet beim Aufruf als Parameter `id_name` den id-Namen eines Elements. Sie ermittelt das aktuelle Datum inklusive Wochentag und schreibt das Ergebnis am Ende ordentlich formatiert mit innerHTML in das Element mit dem im Parameter übergebenen id-Namen.

Auch an dieser Stelle sind die genauen Details im Code noch nicht so wichtig. Wichtig ist zu erkennen, dass die Funktion `show_date()` so, wie sie in der Datei notiert ist, sehr allgemein gehalten ist. Sie kann also in verschiedensten Kontexten bemüht werden. In unserem Beispiel speichern wir die JavaScript-Datei unter dem Namen ***functions.js*** ab. Nachfol-

gend das Listing eines HTML-Dokuments, welches diese JavaScript-Datei einbindet und mit der darin enthaltenen Funktion `show_date()` arbeitet.

Listing 7.4: HTML-Datei mit eingebundenem Script für Datumsausgabe

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Externe JavaScript-Datei</title>
<script type="text/javascript" src="functions.js"></script>
</head>
<body style="background-color:#FFDDAA"
      onLoad="show_date('date')">
<p id="date" style="font-weight:bold"> </p>
<h1 style="border-top:solid 1px black;
           padding-top:15px">News</h1>
<p><strong>+++ Sack Reis in China umgefallen</strong><br>
Wie aus zuverlässigen Quellen bekannt wurde, ist gestern in
China ein Sack Reis umgefallen.</p>
</body>
</html>
```



Abbildung 7.4: Aktuelle Datumsanzeige mit Funktion aus externer JavaScript-Datei

Im HTML-Kopfbereich wird ein Script-Bereich notiert. Dieser hat selber keinen Inhalt, bindet jedoch über das src-Attribut im einleitenden `<script>`-Tag die externe Datei *functions.js* ein, welche die Funktion `show_date()` enthält.

Es ist selbstverständlich zulässig, innerhalb eines Script-Bereichs, der eine externe JavaScript-Datei einbindet, zusätzlich JavaScript-Code zu notieren. Der JavaScript-Interpreter

behandelt in diesem Fall den Code aus der externen Datei und den Code des Elementinhalts wie ein zusammenhängendes Script.

Das src-Attribut erwartet als Wertzuweisung einen URI. Die eingebundene JavaScript-Datei kann also lokal oder auch über eine vollständige Internetadresse referenziert werden. Beachten Sie aber, dass bei der Einbindung von JavaScript-Dateien von einer anderen als der aktuellen Domain je nach Serverkonfiguration Probleme auftreten können.

Im obigen Beispiel wird die externe Scriptdatei zunächst nur eingebunden. Die Funktion `show_date()` wird erst im einleitenden <body>-Tag aufgerufen, und zwar über den Event-Handler `onLoad=`. Die Funktion wird dadurch also erst dann aufgerufen, wenn das Dokument vollständig geladen ist.

Übergeben wird der Funktion als erwarteter Parameter der Wert `'date'`. Wie weiter oben bereits erklärt, interpretiert die Funktion `show_date()` diesen Wert als id-Namen eines im Dokument enthaltenen Elements. Das von ihr erzeugte aktuelle Datum schreibt sie als Elementinhalt in dieses Element.

Innerhalb des body-Bereichs ist deshalb ein entsprechendes p-Element mit dem id-Namen `date` notiert. Als Elementinhalt hat es zunächst nur ein erzwungenes Leerzeichen (` `). Damit dehnt der Browser das Element bereits beim Einlesen auf die gewünschte Größe einer Textzeile aus und braucht später nichts mehr an der Position der Seiteninhalte zu korrigieren.

Zum Zeitpunkt des Aufrufs von `show_date()` ist das Dokument vollständig geladen, d.h., auch das p-Element `date` ist bekannt. Die Funktion kann nun problemlos den Inhalt dieses Elements mit dem aktuellen Datum füllen.

7.3 Sprachkonzepte von JavaScript

Im vorherigen Abschnitt konzentrierten wir uns auf die möglichen Orte von JavaScript-Code und auf das Zusammenspiel zwischen HTML und JavaScript. In diesem Abschnitt geht es nun darum, die syntaktischen Details und Potenziale von JavaScript zu verstehen und anzuwenden.

7.3.1 Allgemeines zur Code-Erstellung

Bevor wir auf die einzelnen syntaktischen Bestandteile von JavaScript eingehen, seien noch ein paar Bemerkungen zur »Philosophie« beim Erstellen von Script-Code gemacht.

Die meisten Programmierer legen sich im Laufe der Zeit einen eigenen Programmierstil zu. Einige bemühen sich, ihren Code so schwer wie möglich nachvollziehbar zu machen, in der meist uneingestandenen und leider oftmals erfüllten Hoffnung, von Anfängern dafür bewundert zu werden und als Guru zu gelten.

Professionell ist das jedoch nicht. Bei professionellem Programmcode steht die Forderung nach Transparenz an oberster Stelle. Dafür gibt es Gründe:

- Wenn ein Programmierer selbst nach Monaten oder gar Jahren ein Script ändern, korrigieren oder optimieren soll, kann er sich in der Regel nicht mehr an alle Einzelheiten des Scriptaufbaus, der gewählten Problemlösungsstrategien usw. erinnern. Je transparenter das Script diese Aspekte jedoch offenbart, desto leichter ist es für den Programmierer, schnell wieder in das Script »hineinzufinden«.
- Scripts werden bei größeren Projekten oftmals auch von anderen Programmierern weiter gepflegt. Jeder Programmierer weiß, dass es nicht einfach ist, sich in fremde Konzepte, Denkstrategien usw. hineinzudenken. Je transparenter das Script in dieser Hinsicht jedoch ist, desto einfacher ist es für andere, Code nachzuvollziehen.
- Teile von Scripts, insbesondere Funktionen, aber auch Datenstrukturen, werden oftmals in anderen Projekten wiederverwendet. Deshalb ist es beim Erstellen von Code sinnvoll, gleich darauf zu achten, dass Code-Teile geschlossen wiederverwendbar sind.

Lesbarkeit

Zur guten Lesbarkeit von Code gehört eine ordentliche Formatierung. Trennen Sie beispielsweise mehrere Funktionen durch eine oder mehrere Leerzeilen voneinander, damit leichter erkennbar ist, wo eine Funktion beginnt und wo sie aufhört. Dem JavaScript-Interpreter ist dies egal – er findet beispielsweise blitzschnell Funktionsblöcke. Der menschliche »Code-Leser« tut sich dabei jedoch deutlich schwerer.

Notieren Sie jede neue Anweisung so weit sinnvoll in einer neuen Zeile. Auch wenn der JavaScript-Interpreter Anweisungen am schließenden Semikolon voneinander trennt, so ist es einfach besser lesbar, wenn Anweisungen wie in klassischen, zeilenorientierten Programmiersprachen zeilenweise notiert sind.

Arbeiten Sie mit Einrückungen. Der Inhalt etwa einer Funktion sollte gegenüber der Funktionsdeklaration um zwei bis vier Zeichen eingerückt sein. Ebenso verhält es sich beim Arbeiten mit Kontrollstrukturen.

Einheitlichkeit

Gewöhnen Sie sich beispielsweise bei der Vergabe von Variablen- und Funktionsnamen ein einheitliches Schema an. Im vorliegenden Buch verwenden wir beispielsweise nur englischsprachige Namen, die ausschließlich klein geschrieben sind und bei denen im Fall mehrerer Namensteile Unterstriche das Leseverständnis erhöhen. Wir verwenden auch keine schwer zu erratenden Abkürzungen.

Es spricht nichts gegen andere Prinzipien, z.B. die Verwendung deutschsprachiger Namen mit anderen Regeln für Klein- und Großschreibung. Kryptisch wird der Code jedoch dann, wenn mal so, mal so gearbeitet wird.

Kommentare

Sparen Sie nicht damit, Code in Kommentaren zu dokumentieren. Auch für die optische Trennung und Einleitung von Code-Abschnitten werden Kommentare gern genutzt.

Dazu als Beispiel noch mal unsere Funktion `show_date()` aus ***Abschnitt 7.2.3***. In diesem Fall wird die Funktion jedoch um verschiedene Kommentare ergänzt. Die Kommentare werden vom JavaScript-Interpreter ignoriert, erleichtern Menschen jedoch das Verständnis des Codes:

Listing 7.5: Funktion mit JavaScript-Kommentaren

```
// ================================================
// Funktion show_date()
// schreibt "Wochentag, TT.MM.JJJJ" in ein Element
// Parameter id_name: id-Name des Elements, in das
// geschrieben werden soll
// ================================================
function show_date(id_name) {
   var weekdays =  new Array("Sonntag","Montag","Dienstag",
                   "Mittwoch","Donnerstag","Freitag","Samstag");
   var now = new Date();
   var day = now.getDate();
   // getMonth liefert Werte zwischen 0 und 11:
   var month = now.getMonth() + 1;
   var year = now.getFullYear();
   var weekday = now.getDay();
   // führende 0 für Tage 1 bis 9:
   var day0  = ((day < 10) ? "0" : "");
   // führende 0 für Monate 1 bis 9:
   var month0  = ((month < 10) ? ".0" : ".");
   var today = day0 + day + month0 + month + "." + year;
   var date_to_show = weekdays[weekday] + ", " + today;
   document.getElementById(id_name).innerHTML = date_to_show;
   /* Alternative zu innerHTML:
   document.getElementById(id_name).firstChild.nodeValue =
      date_to_show; */
}
```

JavaScript-Kommentare beginnen entweder mit zwei Schrägstrichen `//`. In diesem Fall gilt alles, was danach bis zum Zeilenende folgt, als Kommentar. Oder ein Kommentar wird in der Form `/* ... */` notiert. Alles, was zwischen `/*` und `*/` notiert wird, gilt dann als Kommentar.

Im Beispiel werden Kommentare für unterschiedliche Zwecke eingesetzt:

- Oberhalb der Funktion wird durch Kommentare ein optisch hervorgehobener Bereich erzeugt, der die Aufgabe und den korrekten Aufruf der Funktion dokumentiert.
- Innerhalb der Funktion werden einzelnen, sich möglicherweise nicht unmittelbar erschließenden Anweisungen erläuternde Kommentare vorangestellt.
- Unten in der Funktion wird ein alternatives Code-Fragment durch Kommentar ausgeklammert.

Da JavaScripts selbst in Großprojekten nicht so strikt dokumentiert werden wie Softwareentwicklung, bleiben die Kommentare im Script meist die einzige zum Script existierende Dokumentation. Entsprechend ernst sollte dieses Feature vor allem bei nicht trivialen Scripts genommen werden.

7.3.2 Anweisungen, Ausdrücke und Blöcke

JavaScript besteht wie jede Programmiersprache aus einer kontrollierten Abfolge von Anweisungen. Gewöhnen Sie sich am besten gleich an, jede Anweisung durch ein Semikolon abzuschließen. Das ursprüngliche Netscape-JavaScript erlaubt es zwar auch, Anweisungen einfach durch Zeilenumbruch zu trennen. Doch um Fehler zu vermeiden, ist dringend zu empfehlen, davon keinen Gebrauch zu machen. Nur in einem Fall kann das Semikolon bedenkenlos weggelassen werden: wenn in der Wertzuweisung an einen Event-Handler nur eine einzige Anweisung notiert wird, wie z.B. `onLoad="show_date()"`.

Anweisungen und Ausdrücke

»Anweisung« ist ein Oberbegriff. Bei Anweisungen, die als Ergebnis einen Wert produziert, spricht man auch von »Ausdruck«. Dazu folgendes Beispiel:

```
x = 3 + 5;
document.write(x);
```

Das Beispiel besteht aus zwei Anweisungen. Die erste davon ist jedoch ein Ausdruck, da in diesem Fall ein Wert produziert wird. Die zweite Anweisung ist dagegen kein Ausdruck, weil in diesem Fall nur etwas geschieht, was aber keinen neuen Wert schafft.

Anweisungsblöcke

Blöcke aus mehreren Anweisungen und Ausdrücken werden in JavaScript in geschweifte Klammern { bzw. } eingeschlossen. Blöcke entstehen beispielsweise bei Funktionen. Alle Anweisungen einer Funktion werden deshalb von geschweiften Klammern umgeben. Aber auch, wenn mehrere Anweisungen von einer erfüllten Bedingung abhängen sollen, werden diese als Block mit geschweiften Klammern markiert. Es gibt noch mehr Situationen, in denen Blöcke vorkommen können. Wichtig ist zunächst nur zu wissen, dass Blöcke eine strukturelle Einheit aus mehreren Anweisungen bilden, markiert durch geschweifte Klammern. Anweisungsblöcke können auch ineinander verschachtelt sein.

7.3.3 Variablen und Datentypen in JavaScript

Werte, die sich ein Script zur späteren Verwendung merken muss, werden in Variablen gespeichert. Variablen sind benannte, reservierte Speicherplätze für bestimmte Daten.

Variablendeklaration ohne und mit Initialisierung

Es gehört zum sauberen Programmierstil, Variablen bei der ersten Erwähnung im Script zu deklarieren. Dazu ein paar Beispiele:

```
var given_name;
var family_name = "Müller";
var x = 3 * (7 + 5);
var y = x / 2;
var datetime = get_datetime();
```

Variablen werden durch das Schlüsselwort `var`, gefolgt vom gewünschten Variablennamen, deklariert. Es genügt, die Variable erst einmal zu deklarieren, ohne ihr einen Wert zuzuweisen. In der ersten Beispielzeile wird eine solche Variable deklariert (die Variable `given_name`).

Alle anderen Beispielvariablen bekommen bei der Deklaration nach einem Gleichheitszeichen einen Wert zugewiesen. Diese anfängliche Wertzuweisung bezeichnet man als Initialisierung einer Variable. Die Wertzuweisung kann ein schlichter Wert sein, wie im Beispiel der Variablen `family_name`. Sie kann aber auch ein errechneter Wert sein, wie im Beispiel der Variablen `x`. In solchen Berechnungen können auch andere, bereits deklarierte Variablen vorkommen. So wird im Beispiel der Initialisierung von y mit x gerechnet. An die Stelle von x wird dabei deren aktueller Wert eingesetzt. Die Initialisierung kann aber auch wie im letzten Beispiel ein Funktionsaufruf sein. Voraussetzung ist, dass die Funktion einen Wert ermittelt und zurückgibt.

Bei der Initialisierung und auch bei weiteren Wertänderungen von Variablen im Verlauf des Scripts brauchen Sie sich nicht um die Arbeitsspeicherreservierung zu kümmern. Sie haben aus Sicherheitsgründen auch gar keine Einflussmöglichkeit darauf. Die gesamte Speicherverwaltung übernimmt der JavaScript-Interpreter selbst.

Globale und lokale Variablen

Variablen, die in einem script-Bereich oder in einer JavaScript-Datei außerhalb einer Funktion deklariert werden, sind globale Variablen. Auf sie kann auch innerhalb von Funktionen zugegriffen werden. Variablen, die innerhalb einer Funktion deklariert werden, sind dagegen lokale Variablen, deren Gültigkeit und Bekanntheit sich auf die Funktion beschränkt.

Konventionen bei Variablennamen

Variablennamen in JavaScript dürfen aus den Großbuchstaben A bis Z, den Kleinbuchstaben a bis z, den Ziffern 0 bis 9 und dem Unterstrich (_) bestehen. Das erste Zeichen darf keine Ziffer sein. Buchstaben wie ä, ö, ü oder ß sind nicht erlaubt. Leerzeichen auch nicht. Die erlaubte Länge der Namen ist nicht explizit festgelegt, sollte aber aus praktischen Gründen die Größenordnung von ein paar Dutzend Zeichen nicht überschreiten. Variablennamen dürfen nicht mit einem reservierten Schlüsselwort von JavaScript identisch sein. Das betrifft folgende Wörter:

`abstract`, `boolean`, `break`, `byte`, `case`, `catch`, `char`, `class`, `const`, `continue`, `debugger`, `default`, `delete`, `delete`, `do`, `double`, `else`, `enum`, `export`, `export`, `extends`, `final`, `finally`, `float`, `for`, `function`, `goto`, `if`, `implements`, `import`, `import`, `in`, `instanceof`, `int`, `interface`, `long`, `native`,

`new`, `null`, `package`, `private`, `protected`, `public`, `public`, `return`, `short`, `static`, `super`, `switch`, `synchronized`, `this`, `throw`, `throws`, `transient`, `true`, `try`, `typeof`, `var`, `void`, `volatile`, `while`, `with`.

Einige der Schlüsselwörter werden gegenwärtig von JavaScript gar nicht verwendet, sind jedoch für künftige Zwecke reserviert und gehören daher mit zu den zu vermeidenden Variablennamen.

Gültige Variablennamen sind also beispielsweise:

- `Name`
- `nick_name`
- `name2`
- `a`
- `_a`
- `p101260`

Ungültige Variablennamen (führen zu Fehlern) sind beispielsweise:

- `Übergröße` (nicht erlaubte Buchstaben)
- `1a` (Ziffer als erstes Zeichen)
- `family name` (Leerzeichen)
- `true` (reserviertes Wort)

Datentypen und Typumwandlung

Wie praktisch alle Programmiersprachen unterscheidet auch JavaScript zwischen verschiedenen Datentypen. Der Grund ist, dass etwa Zeichenketten und Zahlen intern im Arbeitsspeicher unterschiedlich gespeichert werden. Damit der JavaScript-Interpreter einen gespeicherten Variablenwert korrekt aus dem Speicher ausgelesen kann, muss er wissen, nach welchem Datentyp der Wert zu behandeln ist.

Bei der Wertzuweisung an eine Variable entscheiden Sie durch die Notationsweise, welchen Datentyp Sie meinen. Das folgende Beispiel zeigt den Unterschied:

```
var x = 36;
var y = "36";
```

Zeichenkettenwerte werden in hohe Anführungszeichen gesetzt, Zahlen dagegen nicht. Bei den Anführungszeichen von Zeichenketten sind sowohl doppelte als auch einfache Anführungszeichen erlaubt. Beide die Zeichenkette einschließenden Zeichen müssen jedoch gleich sein.

In der obigen Beispielvariable `x` ist also nach der Initialisierung der numerische Wert 36 gespeichert und in `y` die Zeichenkette 36.

Mit `x` können Sie anschließend rechnen, z.B. `z = Math.sqrt(x)`, was in Variable `z` die Quadratwurzel aus 36 speichert. Wenn Sie `z = Math.sqrt(y)` notieren, wird jedoch ebenfalls 6

errechnet. Anstatt einen Typfehler zu produzieren, nimmt der JavaScript-Interpreter in lösbaren Konfliktsituationen eine ***automatische Typumwandlung*** vor. Da im Beispiel das Rechnen mit einer Zeichenkette verlangt wird und deren Wert sich als gültige Zahl interpretieren lässt, interpretiert es der JavaScript-Interpreter ohne Rückfrage tatsächlich so.

Mit `y` können Sie dagegen Zeichenkettenoperationen durchführen. Auf jede Zeichenkettenvariable in JavaScript ist unter anderem die Eigenschaft `length` anwendbar, welche die Anzahl Zeichen der Zeichenkette ermittelt. Wenn Sie `document.write(y.length)` notieren, schreibt der Browser den Wert 2 ins Dokument, weil in `y` der Wert 36 als Zeichenkette gespeichert ist und diese Zeichenkette aus zwei Zeichen besteht.

Wenn Sie `document.write(x.length)` notieren, wenden Sie eine Zeichenketteneigenschaft auf einen numerisch gespeicherten Wert an. In diesem Fall nimmt der JavaScript-Interpreter jedoch keine automatische Typkonvertierung vor, sondern schreibt »undefined« ins Dokument.

Um sich nicht auf die automatische Typumwandlung zu verlassen, können Sie eine Typumwandlung auch explizit erzwingen. Wichtig ist dabei natürlich, dass die Umwandlung lösbar ist. Wenn Sie etwa `document.write(String(x).length)` notieren und in `x` der numerische Wert 36 gespeichert ist, schreibt JavaScript den Wert 2 ins Dokument. Grund dafür ist die globale JavaScript-Funktion `String()`, die dafür sorgt, dass ein Wert auf jeden Fall als Zeichenkette behandelt wird. Das Konstrukt `String(x)` bedeutet so viel wie »behandle den Wert von `x` als Zeichenkette«. Kombiniert mit der Zeichenketteneigenschaft `length` ergibt sich ein Wert von 2, da 36 als Zeichenkette betrachtet aus zwei Zeichen besteht.

Nachfolgende Tabelle listet globale JavaScript-Funktionen zur Typumwandlung auf.

Funktion	Bedeutung
`String(Wert)`	Wandelt beliebige Datentypen, die in `Wert` gespeichert sind, in eine Zeichenkette um.
`Number(Wert)`	Interpretiert eine Zeichenkette, die in `Wert` gespeichert ist, als Zahl. Ergibt die Prüfung eine gültige Zahl, wird die Typumwandlung vorgenommen. Dabei entscheidet der Script-Interpreter selbständig, ob es sich um eine Ganzzahl (Integer) oder um eine Fließkommazahl (Float) handelt. Ergibt die Prüfung keine gültige Zahl, wird der Wert `NaN` erzeugt (not a number).
`parseInt(Wert)`	Interpretiert eine Zeichenkette, die in `Wert` gespeichert ist, als Ganzzahl (Integer). Ergibt die Prüfung eine gültige Zahl, wird die Typumwandlung vorgenommen. Ergibt die Prüfung keine gültige Zahl, wird der Wert `NaN` erzeugt (not a number). Falls ein Wert wie 35.9 übergeben wird, wird bis zum ersten nicht mehr als Integer interpretierbaren Zeichen konvertiert – es wird also der numerische Wert 35 erzeugt.
`parse-Float(Wert)`	Interpretiert eine Zeichenkette, die in `Wert` gespeichert ist, als Fließkommazahl (Float). Ergibt die Prüfung eine gültige Zahl, wird die Typumwandlung vorgenommen. Ergibt die Prüfung keine gültige Zahl, wird der Wert `NaN` erzeugt (not a number).

Tabelle 7.2: Global verfügbare JavaScript-Funktionen zur expliziten Typumwandlung

Die Notwendigkeit expliziter Typumwandlungen mag zunächst etwas abstrakt erscheinen. Ein gutes Beispiel aus der Praxis ist das Einlesen von Inhalten aus Formularfeldern.

Greift ein JavaScript auf eine Zahl zu, die ein Anwender in einem HTML-Formular eingegeben hat, so liegt der ausgelesene Wert zunächst als Zeichenkette vor. Soll mit dem Wert gerechnet werden, ist eine Typumwandlung in vielen Fällen erforderlich.

Ganzzahlen (Integer) und Fließkommazahlen (Float)

Bei numerischen Werten wird zwischen Ganzzahlen und Fließkommazahlen unterschieden. Der Grund ist auch hier ein Unterschied in der internen Speicherung im Arbeitsspeicher.

Integer-Werte belegen 4 »Octets«, also 4 Einheiten zu je 8 Bit im Arbeitsspeicher. Da es positive und negative Ganzzahlen gibt, wird ein Bit für die Speicherung des Vorzeichens (+ oder –) benötigt. Bleiben 31 Bit für die Zahl. Damit lassen sich 2.147.483.647 (ca. 2,1 Milliarden) unterschiedliche Werte speichern. Der Wertebereich von Integer-Ganzzahlen reicht demnach von –2.147.483.647 bis 2.147.483.647. Das sind 4.294.967.295 unterschiedliche Zahlen.

Bei Zahlen, die größer oder kleiner sind, verwendet der JavaScript-Interpreter automatisch eine Speicherung als Float-Zahl. Float-Zahlen können Sie jedoch auch explizit notieren, indem Sie »Kommazahlen« notieren. Das »Komma« ist in JavaScript allerdings der Punkt.

Falsch ist beispielsweise:

```
var x = 9,81;
```

Richtig ist:

```
var x = 9.81;
```

Der Speicheralgorithmus von Fließkommazahlen ist komplexer (er wird im Standard IEEE754 beschrieben). Der Ausdruck »Fließkomma« (»Gleitpunkt« meint das Gleiche) drückt jedoch gut aus, was passiert: Je kleiner der Ganzzahlanteil einer Float-Zahl ist, desto mehr Nachkommastellen können gespeichert werden. Je größer dagegen der Ganzzahlanteil, desto weniger Nachkommastellen können gespeichert werden. Fließkommawerte können also »unscharf« werden.

Negative Zahlen bekommen in JavaScript wie üblich ein Minuszeichen vorangestellt. Das gilt sowohl für Integer-Zahlen als auch für Float-Zahlen.

Boolean-Werte

Ein Boolean-Wert ist ein ja/nein-Wert. Er kann nur die Werte 1 oder 0 haben und lässt sich auch durch die reservierten Wörter `true` (wahr) und `false` (falsch) ausdrücken. Diese Werte können Sie beispielsweise bei der Variableninitialisierung zuweisen:

```
var play_the_game = false;
var show_help = true;
```

Sehr zweckmäßig sind Boolean-Werte bei Abfragen wie:

```
if(show_help == true)
   document.getElementById("help").innerHTML = help_html;
```

Oder noch einfacher:

```
if(show_help)
   document.getElementById("help").innerHTML = help_html;
```

Entsprechend bei false-Werten:

```
if(play_the_gabe == false)
   highscore = 0;
```

Oder noch einfacher:

```
if(! play_the_game)
   highscore = 0;
```

Auf die Syntax von if-Abfragen und Operatoren wie den Verneinungsoperator `!` gehen wir später noch ein.

Auch Checkboxen und Radiobuttons aus HTML, die ja typische ja/nein-Schalter darstellen, stellen in JavaScript Boolean-Werte dar.

Rein technisch gesehen ist ein Boolean einfach eine Zahl. Jede numerische Variable, welche die Werte 1 oder 0 enthält, enthält damit automatisch auch die Boolean-Werte true bzw. false. Umgekehrt ist eine als Boolean-Wert initialisierte Variable immer auch eine normale Zahl, wie das folgende Beispiel zeigt:

```
var x = true;
x = x + 19;
document.write(x);
```

Das Beispiel schreibt den Wert 20 ins Browser-Fenster. Die Variable `x` wird mit `true`, also 1, initialisiert. Anschließend wird ihrem bisherigen Wert 19 hinzuaddiert.

Boolean-Variablen können auch mithilfe des Boolean-Objekts von JavaScript initialisiert werden. Beispiele:

```
var do_it = new Boolean(true);
var do_not_do_it = new Boolean(false);
```

Es werden zwei Variablen definiert, die im Beispiel die namen `do_it` und `do_not_do_it` erhalten. Beide Variablen werden mit einem Aufruf der Funktion zum Erstellen einer neuen Instanz des Boolean-Objekts aufgerufen (`new Boolean()`). Dieser Funktion wird der gewünschte Initialwert `true` oder `false` übergeben.

Escape-Sequenzen für Steuerzeichen, Sonderzeichen und Maskierung

Bei Zeichenketten können Zeichen in Form von so genannten Escape-Sequenzen notiert werden. Das betrifft vor allem Steuerzeichen wie Zeilenvorschub oder Tabulator, aber auch Zeichen aus erweiterten Zeichenvorräten wie dem Latin-1-Zeichensatz oder dem Unicode-System, und schließlich jene Zeichen, die in JavaScript innerhalb von Zeichenketten besondere Bedeutung haben. Die nachfolgende Tabelle listet mögliche Escape-Sequenzen auf.

Zeichenfolge	Bedeutung
`\b`	Backspace-Zeichen
`\f`	Formfeed-Zeichen (Steuerzeichen für Seitenvorschub)
`\n`	Newline-Zeichen (Steuerzeichen für Zeilenvorschub)
`\r`	Rewind-Zeichen (Steuerzeichen für Wagenrücklauf)
`\t`	Tabulator-Zeichen
`\\`	Backslash-Zeichen (um das Zeichen zu maskieren)
`\'`	Einfaches Anführungszeichen (um das Zeichen zu maskieren)
`\"`	Doppeltes Anführungszeichen (um das Zeichen zu maskieren)
`\###`	Zeichen aus ISO-8859-1 als Oktalwert mit drei Ziffern (#) dargestellt, also mit Ziffern von 0 bis 7, z. B. `"M\374nz"` (`\374` erzeugt »ü«)
`\x##`	Zeichen aus ISO-8859-1 als Hexadezimalwert mit zwei Ziffern (#) dargestellt, also mit Ziffern von 0 bis 9 und A bis F, z. B. `"M\xFCnz"` (`\xFC` erzeugt »ü«)
`\u####`	Zeichen aus dem Unicode-System als Hexadezimalwert mit vier Ziffern (#) dargestellt, also mit Ziffern von 0 bis 9 und A bis F, z. B. `"\u20AC 10,00"` (`\u20AC` erzeugt »€«)

Tabelle 7.3: Escape-Sequenzen in JavaScript

Für Zeichenketten, die als HTML ausgegeben werden sollen, sind einige dieser Sequenzen uninteressant. Sonderzeichen sollten in solchen Zeichenketten in den Notationsformen erscheinen, die in HTML vorgesehen sind. Sehr häufig werden Sie dagegen die Escape-Sequenzen zum Maskieren benötigen. Dazu ein Beispiel:

```
document.write
   ("<a href=\"http://www.example.org/\">Beispiel-Link</a>");
```

Wenn Sie direkt mit `document.write()` HTML-Code in ein HTML-Dokument schreiben, müssen Sie die Zeichenkette, die der HTML-Code darstellt, wahlweise in doppelte oder in einfache hohe Anführungszeichen einschließen. Enthält die Zeichenkette eines der Zeichen, die zum Einschließen der Zeichenkette verwendet werden, muss es maskiert werden. Andernfalls würde der JavaScript-Interpreter das Ende der Zeichenkette vermuten und den Rest als ungültiges JavaScript mit einem Fehler quittieren. Im obigen Beispiel sind daher die Anführungszeichen der Wertzuweisung an das href-Attribut maskiert.

Nicht maskieren müssen Sie die "-Zeichen im folgenden Fall:

```
document.write
   ('<a href=\"http://www.example.org/\">Beispiel-Link</a>');
```

In diesem Fall wird die Zeichenkette in einfache hohe Anführungszeichen eingeschlossen. Darin vorkommende doppelte Anführungszeichen sind dann nicht maskierungspflichtig.

Oktalnotation, Hexadezimalnotation und Potenzschreibweise von Zahlen

Ebenso wie Zeichen können auch Zahlen anders geschrieben werden als in der für Menschen am besten lesbaren Dezimalschreibweise. Dazu ein Beispiel:

```
var x = 0x80 + 040 + 1.7E2;
document.write(x);
```

Bei diesem Beispiel schreibt der Browser am Ende den Wert 330 ins Dokument. Die Variable x bekommt bei der Initialisierung eine Addition von drei Zahlen zugewiesen. Die erste Zahl ist in *Hexadezimalschreibweise* notiert. Solche Notationen beginnen mit `0x`, gefolgt von den Ziffern der Zahl. Erlaubt sind als Ziffern 0 bis 9 und A bis F (oder a bis f), wobei die Buchstaben den Dezimalwerten 10 bis 15 entsprechen. `0x10` bedeutet dann so viel wie 16 usw.

Die zweite Zahl im Beispiel ist eine Oktalzahl (System zur Basis 8). Solche Zahlen dürfen nur aus den Ziffern 0 bis 7 bestehen. Markiert werden Oktalzahlen durch eine führende `0`. Diese Eigenheit kann in der Praxis übrigens für Verwirrung sorgen: Jede Ganzzahl, die aus mehr als einer Ziffer besteht und mit einer Ziffer 0 beginnt, wird von JavaScript oktal interpretiert. Das kann zu unerwarteten Ergebnissen führen, wenn JavaScript als Input eine »formatierte Zahl« mit führenden Nullen erhält und dann mit dieser Zahl gerechnet wird.

Die Potenzschreibweise schließlich ist vor allem bei sehr großen und sehr kleinen Gleitkommazahlen verbreitet. Mit e oder E bestimmen Sie die Zehnerpotenz bzw. die Anzahl Nullen, die hinter der Zahl vor dem e bzw. E stehen. `1.2e2` ist so viel wie 120, weil der Dezimalpunkt in Gedanken um zwei Stellen nach rechts verschoben wird, oder anders ausgedrückt, weil der Wert vor dem e mit 10^2, also mit 100, multipliziert wird. Durch Minuszeichen hinter dem e bzw. E wird der Dezimalpunkt dagegen nach links verschoben. `1.2e-2` entspricht 0.012.

7.3.4 Operatoren in JavaScript

Nachdem wir die etwas trockene Materie von Variablen und Werten hinter uns gebracht haben, steigen wir nun in den Bereich ein, der »typisch« ist für jedes Programmieren. Dazu gehören vor allem if-Abfragen, aber auch Schleifen und Fallunterscheidungen. Um sie zu realisieren, sind in vielen Fällen Operatoren erforderlich.

Ein Beispiel

Zum Warmwerden stellen wir ein typisches Praxisbeispiel vor. Darin soll der Anwender Text für eine SMS-Nachricht in ein Feld eingeben können. Die erforderliche Technik zum Versenden einer SMS über das Web muss natürlich serverseitig implementiert werden und soll hier nicht interessieren. Ein nützlicher Dienst, den JavaScript verrichten kann, ist jedoch, den Anwender während der Texteingabe jederzeit darüber zu informieren, wie viele Zeichen er noch eingeben kann. Denn die Länge von SMS-Nachrichten ist ja auf 160 Zeichen begrenzt.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Externe JavaScript-Datei</title>
<script type="text/javascript">
function check_input(id_name, max_value) {
  if(document.getElementById(id_name).value.length >
     max_value)
     document.getElementById(id_name).value =
       document.getElementById(id_name).value.substr(0,
         max_value);
  document.getElementById('max').innerHTML =
     max_value - document.getElementById(id_name).value.length;
}
</script>
</head>
<body>
<form action="javascript:void(0)"
      style="background-color:#FFDDAA; padding:10px">
<p>SMS-Text:
<input type="text" value="" id="username"
       style="width:600px; border:solid 1px black"
       onkeyup="check_input('username', 160)"><br>
noch max. <span id="max" style="font-weight:bold">160</span>
Zeichen</p>
</form>
</body>
</html>
```

Das Beispiel ist so programmiert, dass bei jedem Tastendruck die fett angezeigte Zahl unterhalb des Eingabefelds aktualisiert wird. Beträgt der Restwert weiterer Zeichen 0, werden weitere eingegebene Zeichen einfach sofort wieder gelöscht und im Feld bleiben nur die ersten 160 Zeichen stehen.

Das input-Element im HTML-Formular enthält zu diesem Zweck einen Event-Handler `onkeyup=`. Dieser wird bei jedem Loslassen einer Taste ausgelöst. Dabei wird jedes Mal die Funktion `check_input()` aufgerufen, die in einem script-Bereich im Kopfteil des HTML-Dokuments notiert ist.

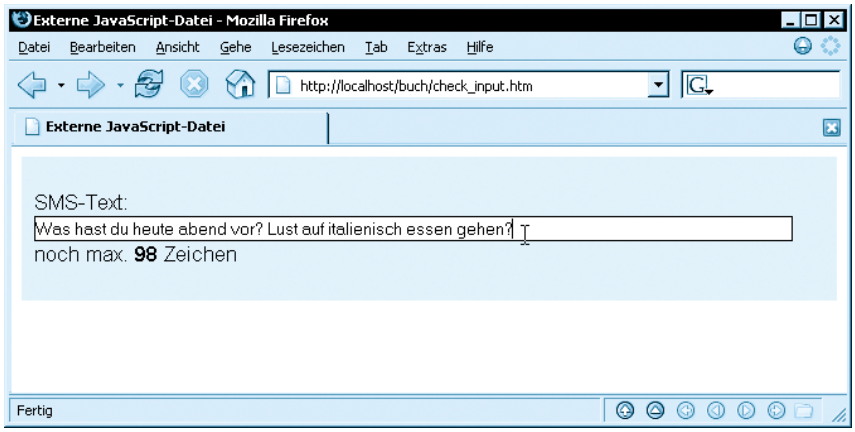


Abbildung 7.5: JavaScript-Überprüfung bei Benutzereingabe

In dieser Funktion kommen einige Operatoren und eine Kontrollstruktur vor. Der Funktionsinhalt beginnt mit einer if-Abfrage. Eine solche Abfrage gehört zu den Kontrollstrukturen. Bei jeder if-Abfrage muss eine Bedingung formuliert werden. Der JavaScript-Interpreter prüft die Bedingung und entscheidet, ob sie wahr oder falsch ist. Wenn sie wahr ist, werden die Anweisungen ausgeführt, die im Zweig unterhalb der if-Abfrage notiert sind. Sollen mehrere Anweisungen abhängig davon ausgeführt werden, müssen diese in geschweifte Klammern eingeschlossen werden. In unserem Beispiel soll bei erfüllter if-Bedingung jedoch nur eine Anweisung ausgeführt werden. In diesem Fall dürfen die geschweiften Klammern fehlen und es wird einfach die nächstnotierte Anweisung ausgeführt.

Im Beispiel wird folgende if-Abfrage formuliert:

```
if(document.getElementById(id_name).value.length > max_value)
```

Die zu prüfende Bedingung ist das, was innerhalb der Klammern von `if()` steht. Ein typischer Fall für das Formulieren einer Bedingung ist das Vergleichen von Werten. Im Beispiel besteht die Bedingungsfrage darin, ob die Zeichenanzahl (`length`) des Inhalt des Formularfelds (`document.getElementById(id_name).value`) größer ist als der Wert der Variablen `max_value`. Ein Wert für `max_value` wird der Funktion `check_input()` beim Aufruf übergeben.

In unserem Beispiel wird beim Funktionsaufruf im Event-Handler `onkeyup=` als Wert für `max_value` 160 übergeben. Angenommen, der Anwender hat bislang 10 Zeichen eingegeben, so dass als aktuelle Länge im Eingabefeld der Wert 10 ermittelt wird. Die if-Abfrage liest sich dann so:

```
if(10 > 160)
```

Was so viel bedeutet wie: »Wenn 10 größer als 160 ist, dann führe die nachfolgende Anweisung aus.« Der JavaScript-Interpreter kann diese Formulierung in jedem denkbaren

Zustand eindeutig mit ja oder nein (»trifft zu« oder »trifft nicht zu«) beantworten. Und so führt er die bedingungsabhängige Anweisung entweder aus oder nicht.

Bei der Formulierung von Bedingungen kommen sehr häufig Operatoren zum Einsatz. Das >-Zeichen für »größer als« ist ein solcher Operator, ein Vergleichsoperator. Im Beispiel kommen aber auch noch andere Operatoren vor. Der gängigste davon ist sicherlich der bereits bekannte Zuweisungsoperator, ausgedrückt durch das Gleichheitszeichen =. Aber auch Rechenoperatoren gehören dazu. Die Funktion im Beispiel benutzt einen solchen Operator (den Subtraktionsoperator »-«), um den Inhalt des span-Elements mit dem id-Namen `max` zu aktualisieren. Der Rechenausdruck lautet:

```
max_value - document.getElementById(id_name).value.length
```

Vom maximal erlaubten Wert werden dabei die tatsächlich aktuell im Eingabefeld enthaltenen Zeichen abgezogen. Das Ergebnis ist die Anzahl der noch erlaubten Zeichen.

Ein weiterer Operator, der im Beispiel auftaucht, ist der void()-Operator im einleitenden `<form>`-Tag. Wir haben bereits früher erläutert, welchen Zweck diese Wertzuweisung an das action-Attribut hat. Vom Typ her ist `void()`, obwohl es wie eine Funktion aussieht, ein Operator. Seine Aufgabe besteht darin, zu verhindern, dass eine Operation einen Wert erzeugt.

Kontrollstrukturen und Operatoren sind also allgegenwärtig beim Programmieren. Insofern ist es wichtig, ihre Syntax und Eigenheiten genau zu kennen.

Vergleichsoperatoren

Vergleichsoperatoren werden vor allem zur Formulierung von Bedingungen in if-Abfragen oder Schleifen genutzt. Sie dienen dazu, zwei Werte zu vergleichen.

Zeichenfolge	Bedeutung
==	Prüfung auf Gleichheit zweier Werte
===	Prüfung auf Gleichheit zweier Werte inklusive Typübereinstimmung
!=	Prüfung auf Ungleichheit zweier Werte
<	Prüfung, ob linker Wert kleiner als rechter Wert
<=	Prüfung, ob linker Wert kleiner als oder gleich rechtem Wert
>	Prüfung, ob linker Wert größer als rechter Wert
>=	Prüfung, ob linker Wert größer als oder gleich rechtem Wert

Tabelle 7.4: Vergleichsoperatoren in JavaScript

Ein kleines Beispiel:

```
var browser = navigator.appName;
if(browser == "Netscape")
   document.write("wirklich Netscape?");
```

Der Operator zur Prüfung auf Gleichheit zweier Werte darf nicht mit dem Zuweisungsoperator verwechselt werden. Hin und wieder unterläuft Programmierern der Fehler, dass sie ein Gleichheitszeichen vergessen und stattdessen eine Zuweisung formulieren anstelle eines Vergleichs. Wenn im obigen Beispiel etwa `if(browser = "Netscape")` notiert wäre, so wäre das aus syntaktischer Sicht nicht falsch, würde aber überraschende Ergebnisse liefern. Da die einfache Zuweisung immer wahr ist, ist das, was da als »Bedingung« formuliert wird, folglich auch immer wahr. Als Folge würde im Beispiel jeder Browser behaupten, er sei ein Netscape-Browser.

Auch bei einfachen Vergleichen von Werten greift in JavaScript die automatische Typumwandlung. Ein Beispiel:

```
var x = 36;
var y = "36";
if(x == y)
   document.write("beide Werte sind gleich");
```

Die Browser geben »beide Werte sind gleich« aus. Um genauer zu vergleichen, können Sie jedoch auch `if(x === y)` notieren. Dabei würde nicht auf Gleichheit erkannt, weil beide Variablen unterschiedlichen Typs sind.

Beim Operator `!=` ist zu beachten, dass eine damit formulierte Bedingung dann wahr ist, wenn die Werte ungleich sind, und unwahr, wenn die Werte gleich sind.

Rechenoperatoren

Für die vier Grundrechenarten stehen in JavaScript die üblichen Operatoren zur Verfügung. Daneben gibt es noch den in vielen Programmiersprachen üblichen Modulo-Operator für Restwertdivision sowie einige verkürzte Notationsformen.

Zeichenfolge	Bedeutung
+	Addiert den Wert rechts zum Wert links
-	Subtrahiert den Wert rechts vom Wert links
*	Multipliziert den Wert links mit dem Wert rechts
/	Dividiert den Wert links durch den Wert rechts
%	Dividiert den Wert links durch den Wert rechts ganzzahlig und liefert den Divisionsrest
+=	Verkürzte Schreibweise für Addition eines Werts zum Wert einer Variablen
-=	Verkürzte Schreibweise für Subtraktion eines Werts vom Wert einer Variablen
*=	Verkürzte Schreibweise für Multiplikation eines Variablenwerts mit einem Wert
/=	Verkürzte Schreibweise für Division eines Variablenwerts durch einen Wert
++	Schreibweise für Variablen-Inkrementierung (1 addieren)
--	Schreibweise für Variablen-Dekrementierung (1 subtrahieren)

Tabelle 7.5: Berechnungsoperatoren in JavaScript

Bei Rechenoperationen gilt die mathematisch übliche Punkt-vor-Strich-Regel, d.h., Multiplikation und Division haben Vorrang vor Addition und Subtraktion. Um die Regel zu beeinflussen, können Sie Klammern verwenden. `3 + 5 * 4` ergibt 23, weil zuerst multipliziert, dann addiert wird. Durch Notation von `(3 + 5) * 4` wird 32 ermittelt, da die Klammer zuerst berechnet wird. Klammern können beliebig verschachtelt sein.

Der Modulo-Operator funktioniert folgendermaßen: `13 % 5` ergibt als Ergebnis 3, weil 13 / 5 ein Ganzzahlergebnis von 2 ergibt (10 / 5) und 3 übrig bleiben.

Die verkürzten Schreibweisen `+=`, `-=`, `*=` und `/=` haben sich in vielen Programmiersprachen etabliert, weil es häufig vorkommt, dass mit Variablen, ausgehend von ihrem aktuellen Wert, eine Rechenoperation durchgeführt werden soll. Beispiel:

```
x = x + 3;
```

Dies kann verkürzt werden zu:

```
x += 3;
```

Wichtig ist, dass zwischen Operator und Gleichheitszeichen kein Leerzeichen steht.

Die beiden Operatoren für die so genannte Inkrementierung und Dekrementierung schließlich treten vor allem in Schleifen auf. Wir werden in Zusammenhang damit darauf zurückkommen.

Beim Rechnen mit JavaScript werden Sie möglicherweise einige Überraschungen erleben. Ein Beispiel:

```
document.write(0.3 + 0.6);
```

Als Ergebnis schreibt der Browser beispielsweise 0.8999999999999999 ins Dokument. Dies sind jedoch keine Rechenfehler im engeren Sinn, sondern »Darstellungsfehler«. Intern wird mit Zahlen stets binär gerechnet. Bei Ausgaben von Zahlen, wie es etwa `document.write()` tut, muss der binär gespeicherte Wert jedoch in Dezimalschreibweise übertragen werden. Da aber, wie bereits erläutert wurde, gerade Float-Zahlen stets mit einer wertabhängigen Unschärfe gespeichert werden, überträgt sich diese Unschärfe auch bei der Übertragung in die Dezimaldarstellung.

In ***Abschnitt 7.3.9*** gehen wir auf Lösungsmöglichkeiten für diese Probleme ein.

Logische Operatoren

In JavaScript gibt es zwei logische Verknüpfungen: »und« und »oder«. Verwendet werden sie beispielsweise zur Formulierung komplexer Bedingungen.

Zeichenfolge	Bedeutung
&&	Logische Und-Verknüpfung von zwei Ausdrücken. Beide Ausdrücke müssen wahr sein, damit die gesamte Bedingung wahr ist. In allen anderen Fällen ist die gesamte Bedingung nicht erfüllt.
||	Logische Oder-Verknüpfung von zwei Ausdrücken. Mindestens einer der Ausdrücke muss wahr sein, damit die gesamte Bedingung wahr ist. Nur wenn keiner der Ausdrücke wahr ist, ist die gesamte Bedingung nicht erfüllt.

Tabelle 7.6: Logische Operatoren in JavaScript

Ein Beispiel:

```
if(document.getElementById('city') == "München" &&
   parseInt(document.getElementById('year_income')) < 20000)
      comment = "Sie müssen wirklich sehr arm sein!";
```

Die Bedingung, die erfüllt sein muss, damit die angegebene Wertzuweisung an die Variable `comment` erfolgt, besteht aus zwei Bedingungen: Wenn in einem gedachten Formularelement mit id-Namen `city` der Wert `München` eingetragen wurde und wenn in einem anderen Formularelement als Jahresgehalt ein Wert von unter 20000 angegeben wurde, dann (und nur dann) ist die gesamte Bedingung erfüllt. Die beiden Einzelbedingungen bestehen darin, dass ein aus einem gedachten Formularelement gewonnener Wert mit einem bestimmten Wert durch Vergleichsoperatoren verglichen wird. Die logische Verknüpfung der Einzelbedingungen wird durch `&&` erreicht.

Es lassen sich auch mehr als zwei Einzelbedingungen verknüpfen. Dabei dürfen logisches Und und logisches Oder auch gemischt werden. Dazu ist es wichtig zu wissen, dass von links nach rechts aufgelöst wird und im Zweifelsfall der Operator `||` Vorrang vor dem Operator `&&` hat. Dazu noch ein Beispiel:

```
var x = true;
var y = false;
var z = true;
if(x && y || x && z || y && z)
   document.write("<p>das Ganze ist wahr</p>");
if(x || y && x || z && y || z)
   document.write("<p>das Ganze ist wahr</p>");
```

In dem Beispiel werden drei Boolean-Variablen initialisiert. Dann folgen zwei komplexe Bedingungen in if-Abfragen.

Die erste zusammengesetzte Bedingung lautet im Klartext: »Wenn `x` und `y` beide wahr sind oder wenn `x` und `z` beide wahr sind oder wenn `y` und `z` beide wahr sind«. In diesem Fall ist das Ganze wahr. Da die Einzelbedingungen durch logisches Oder verknüpft werden, genügt es, wenn eine davon wahr ist, damit die gesamte Bedingung als wahr bewertet wird. Da die Einzelbedingung `x && z` wahr ist, weil beide Variablen den Boolean-Wert `true` haben und bei logischem Und bei beiden wahren Bedingungen die Gesamtbedin-

gung wahr ist, ist auch die zusammengesetzte Bedingung der if-Abfrage wahr. Die document.write()-Anweisung wird daher ausgeführt.

Ebenso verhält es sich im zweiten Fall. Bewertet wird: »Wenn von `x` oder `y` eins wahr ist und wenn von `x` oder `z` eins wahr ist und wenn von `y` oder `z` eins wahr ist«. In diesem Fall ist das Ganze wahr. Da die Einzelbedingungen durch logisches Und miteinander verknüpft sind, müssen alle drei Einzelbedingungen wahr sein, damit das Ganze wahr ist. Von `x` oder `y` ist eins wahr, weil `x = true`. Von `x` oder `z` sind sogar beide wahr, weil `x = true` und `z = true`. Von `y` oder `z` ist eins wahr, weil `z = true`. Alle drei Einzelbedingungen sind wahr, also ist auch die gesamte Bedingung wahr.

Um die Bewertung zu beeinflussen, können Sie auch mit Klammern arbeiten.

Weitere Operatoren

JavaScript bietet noch eine ganze Reihe weiterer Operatoren an, darunter auch wie Funktionen aussehende Operatoren wie `void()`, `typeof()`, `new()`, `delete()` usw. Diese werden wir jedoch an anderen Stellen behandeln.

Wichtig ist noch der ***Negationsoperator***, ausgedrückt durch ein einfaches `!`-Zeichen. Ein Beispiel:

```
if(!document.getElementById)
   return;
```

Das Konstrukt, wie es dort steht, werden wir später noch häufiger verwenden, um Browser auszuschließen, die kein DOM-fähiges JavaScript interpretieren. Im Beispiel wird abgefragt, ob der Browser ein bestimmtes Objekt, eine bestimmte Objektmethode oder Objekteigenschaft ***nicht*** kennt, in diesem Fall `document.getElementById`. Das ***Nicht*** wird dabei durch das voranstehende `!` signalisiert.

Ebenfalls wissen sollten Sie, dass der Operator + nicht nur Zahlen, sondern auch Zeichenketten verknüpft. Ein Beispiel:

```
var given_name = "Stefan";
var family_name = "Münz";
var name = given_name + " " + family_name;
document.write(name);
```

Im Beispiel werden bei der Initialisierung der Variablen `name` der Wert von `given_name`, ein Leerzeichen und der Wert von `family_name` über den Zeichenkettenoperator `+` zusammengesetzt.

Eine weitere Gruppe von Operatoren sind die so genannten Bit-Operatoren. Diese eignen sich zur Manipulation von einzelnen Zeichen oder Zahlen. Wegen der seltenen Verwendung und des umfangreichen nötigen Hintergrundwissens über binäre Speicherung gehen wir jedoch nicht näher darauf ein.

7.3.5 Kontrollstrukturen in JavaScript

Beispiele für einfache if-Abfragen haben wir bereits kennen gelernt. Solche Abfragen gehören zu den so genannten Kontrollstrukturen. Diese heißen so, weil sie Verzweigungsmöglichkeiten im Programmcode schaffen und weil sie es erlauben, Programmcode zu wiederholen.

if-Abfragen, if-else- und else-if-Konstrukte

Wenn die Bedingung einer if-Abfrage vom Scriptinterpreter als wahr bewertet wird, werden eine oder mehrere Anweisungen ausgeführt, die abhängig davon notiert sind. Mehrere Anweisungen müssen dabei als Block in geschweifte Klammern eingeschlossen werden. Ein Beispiel:

```
if(attempt_count == 10) {
   reset_game();
   ask_for_new_game();
   attempt_count = 0;
}
```

Das Beispiel könnte aus einem mit JavaScript realisierten Online-Spiel stammen. Der Anwender hat zehn Versuche, um ein Ziel zu erreichen. Bei jedem erfolglosen Versuch wird die Anzahl der Versuche um 1 hochgezählt. Bei zehn Versuchen soll abgebrochen werden. Die Variable zum Merken der Anzahl Versuche ist `attempt_count`. Die if-Abfrage prüft, ob die Variable den Wert 10 hat. Ist dies der Fall, werden drei durch geschweifte Klammern eingeschlossene Anweisungen ausgeführt, nämlich die Funktionsaufrufe `reset_game()` und `ask_for_new_game()` sowie das Zurücksetzen der Variablen `attempt_count` auf 0.

Oft soll aber nicht nur Code ausgeführt werden, wenn eine if-Bedingung wahr ist, sondern auch dann, wenn sie nicht wahr ist. Dazu gibt es das if-else-Konstrukt. Wir erweitern unser Beispiel:

```
if(attempt_count == 10) {
   reset_game();
   ask_for_new_game();
   attempt_count = 0;
}
else {
   next_attempt();
   attempt_count += 1;
}
```

Nun werden im Fall, dass `attempt_count` noch nicht den Wert 10 erreicht hat, ebenfalls zwei Anweisungen ausgeführt. Eine Funktion namens `next_attempt()` wird aufgerufen und der Zähler `attempt_count` wird um 1 erhöht. Der else-Zweig des if-else-Konstrukts wird also ausgeführt, egal ob `attempt_count` aktuell gerade bei 1 oder bei 4 oder 9 steht.

Noch einen Schritt weiter gehen Erweiterungen mit `else if`. Damit können weitere Bedingungen formuliert werden für den Fall, dass die Bedingungen zuvor nicht erfüllt sind. Wir erweitern unser Beispiel abermals:

```
if(attempt_count == 10) {
   reset_game();
   ask_for_new_game();
   attempt_count = 0;
}
else if(attempt_count == 9) {
   warning_last_attempt();
   next_attempt();
   attempt_count += 1;
}
else {
   next_attempt();
   attempt_count += 1;
}
```

Zwischen if- und else-Zweig wird noch ein else-if-Zweig geschoben. Angenommen, der aktuelle Wert von `attempt_count` beträgt 9. In diesem Fall ist die Bedingung bei `if` nicht erfüllt. Die Bedingung, die hinter `else if` steht, ist jedoch erfüllt. Deshalb gelangt das Script in diesen Zweig und führt dessen Anweisungen aus. Im Beispiel unseres gedachten Spiels wird am Ende das Gleiche ausgeführt wie im else-Zweig. Zuvor wird jedoch noch eine Funktion namens `warning_last_attempt()` aufgerufen.

If- und else-Konstrukte können auch verschachtelt sein. Dazu folgendes Beispiel:

```
if(price >= 30000) {
   if(year_income < 30000)
      creditworthiness = "low";
   else if(year_income < 50000)
      creditworthiness = "medium";
   else
      creditworthiness = "high";
}
```

Dieses fiktive Beispiel nimmt an, dass ein Script einen vom Anwender gewünschten Kaufvorgang auswertet. Dabei werden der Kaufpreis der gewünschten Ware und das vom Anwender erfragte Jahreseinkommen in Beziehung gesetzt, um die Kreditwürdigkeit des Anwenders zu bewerten.

Die »äußere« if-Abfrage stellt eine Bedingung für den Kaufpreis auf. Wenn dieser mehr als 30000 beträgt, wird der Code innerhalb der geschweiften Klammern ausgeführt. In diesem Fall gelangt der Script-Interpreter in die »innere« if-Abfrage. Deren Bedingung besteht darin, dass das angegebene Jahreseinkommen weniger als 30000 beträgt. Dann wird die Variable `creditworthiness` auf `low` gesetzt. Ein else-if-Zweig und ein else-Zweig sorgen für weitere mögliche Wertzuweisungen.

Fallunterscheidung mit switch und case

Wenn Sie folgendes Konstrukt haben:

```
if(x == 1)
   y = 10;
else if(x == 2)
   y = 100;
else if(x == 3)
   y = 1000;
else if(x == 4)
   y = 10000;
else
   y = 1;
```

Dann ist es sauberer, die dafür vorgesehene Fallunterscheidungssyntax zu verwenden.

```
switch(x) {
   case 1:
      y = 10;
      break;
   case 2:
      y = 100;
      break;
   case 3:
      y = 1000;
      break;
   case 4:
      y = 10000;
      break;
   default:
      y = 1;
      break;
}
```

Fallunterscheidungen mit switch-case sind also immer dann sinnvoll, wenn Sie verschiedene mögliche Zustände einer Variablen unterscheiden und für jeden unterschiedenen Zustand individuellen Code ausführen möchten.

Die Syntax beginnt mit `switch(Variablenname)`. Innerhalb geschweifter Klammern können beliebig viele Abarbeitungsmarken des Typs `case Wert:` gesetzt werden. Am Ende sollte eine Marke des Typs `default:` gesetzt werden, die alle Variablenzustände behandelt, welche nicht durch zuvor unterschiedene Fälle abgedeckt wurden. Abhängig von `case:` und `default:` können ein oder mehrere Anweisungen notiert werden. Zum sauberen Programmierstil gehört es, eine `break`-Anweisung am Ende jedes unterschiedenen Falls zu notieren. Dadurch wird der Scriptinterpreter angewiesen, das gesamte switch-Konstrukt zu verlassen, sobald ein Fall auf den aktuellen Variablenzustand zutrifft und die entsprechenden Anweisungen abgearbeitet wurden.

for-Schleifen und for-in-Schleifen

Schleifen mit dem Schlüsselwort `for` eignen sich, um Anweisungen in einer von vorneherein feststehenden Anzahl von Wiederholungen auszuführen. Das folgende Beispiel zeigt einen typischen Anwendungsfall:

```
<script type="text/javascript">
var html = "<table border=\"1\"><tr>";
for(i = 1; i <= 10; i++)
  html += "<td>" + i + "</td>";
html += "</tr></table>";
document.write(html);
</script>
```

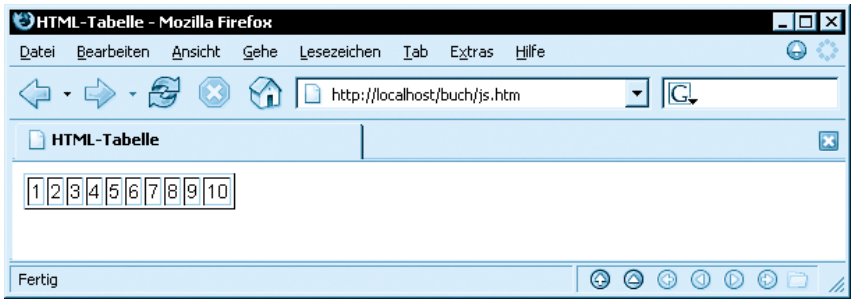


Abbildung 7.6: Mit for-Schleife erzeugte HTML-Tabelle

Das Script in diesem Bereich errichtet eine kleine HTML-Tabelle, bestehend aus einer Zeile mit zehn Zellen. Jede Zelle erhält als Textinhalt den aktuellen Zählerwert. Das Beispiel ist auch insofern praxistypisch, als es zeigt, wie größere Mengen von HTML-Code nach und nach in einer Variablen gesammelt werden. Die Variable wird dann am Ende in einem einzigen Schritt ausgegeben. Das ist wesentlich sauberer, als mit etlichen `document.write()`-Anweisungen zu arbeiten.

Deklariert wird eine Variable namens `html`. Zur Initialisierung wird ihr gleich der HTML-Code für das einleitende `<table>`-Tag und das `<tr>`-Tag der Tabellenzeile als Zeichenkette zugewiesen. Dann folgt die for-Schleife, welche die zehn Zellen erzeugt. In den runden Klammern der for-Schleife sind drei Anweisungen notiert. Die erste initialisiert eine Variable; die zweite formuliert eine Durchlaufbedingung für die Schleife; die dritte verändert die Variable so, dass die Durchlaufbedingung irgendwann nicht mehr erfüllt ist und die Schleife abbricht.

Seit Generationen von Programmierern hat es sich eingebürgert, für eine typische Zählervariable den Namen `i` zu vergeben. In der ersten Anweisung des for-Schleifenkopfs wird im Beispiel eine Variable dieses Namens deklariert und mit dem Wert 1 initialisiert. In der mittleren Anweisung wird als Durchlaufbedingung für die Schleife `i <= 10` formuliert. Die Schleife wird also durchlaufen, wenn `i` einen Wert kleiner oder gleich 10 hat. In der letzten Anweisung wird `i` inkrementiert, also um 1 hochgezählt.

Im Beispiel ist nur eine Anweisung vom Schleifenkopf abhängig. Wären es mehrere Anweisungen, müssten sie als Block in geschweifte Klammern eingeschlossen werden. Die abhängige Anweisung erweitert den Inhalt der Variablen `html` um den HTML-Code einer Tabellenzelle, als deren Textinhalt der aktuelle Wert von `i` eingefügt wird.

Bei jedem Schleifendurchlauf wird die Durchlaufbedingung erneut geprüft. Durch die Inkrementierung nimmt `i` nach und nach die Werte 2, 3, 4 usw. bis 10 ein, und schließlich auch 11. Bei diesem Wert ist die Durchlaufbedingung »kleiner oder gleich 10« jedoch nicht mehr erfüllt und die Schleife wird nicht mehr weiter durchlaufen.

Am Ende erhält die Variable `html` im Beispiel noch den HTML-Code zum Schließen der noch offenen Elemente. Damit ist der gesamte Code der Tabelle in der Variablen gespeichert und wird mit `document.write()` ausgegeben.

for-Schleifen können selbstverständlich auch verschachtelt sein. Das wäre im Beispiel nötig, wollte man mehrere Tabellenzeilen erzeugen. Die äußere for-Schleife würde dann so viele Zeilen erzeugen wie nötig und die innere pro Zeile die gewünschte Anzahl Zellen.

Eine Variante der for-Schleife ist die for-in-Schleife. Diese Variante eignet sich besonders gut zum Traversieren von Arrays. Ein Beispiel:

```
var names = new Array("Anna", "Berta", "Cäsar", "Dora", "Emil");
for(name in names)
   document.write(names[name] + "<br>");
```

Das Beispiel deklariert einen Array `names`, bestehend aus fünf Zeichenkettenwerten. Mithilfe der for-in-Schleife werden die Namen untereinander ins Browser-Fenster geschrieben. Dazu wird eine Traversionsvariable verwendet – im Beispiel heißt sie `name`. Der Ausdruck `name in names` bedeutet dann so viel wie »der Reihe nach alle Elemente aus dem Array `names`, wobei in `name` jeweils das aktuelle Element gespeichert ist«.

Auf die Syntax und Möglichkeiten von Arrays gehen wir in ***Abschnitt 7.3.11*** näher ein.

while-Schleifen und do-while-Schleifen

Schleifen mit `while` und `do-while` eignen sich für Fälle, in denen nicht bekannt oder errechenbar ist, wie oft die Schleife durchlaufen werden soll. Im Schleifenkopf, der bei der while-Schleife oberhalb des Schleifeninhalts und bei der do-while-Schleife unterhalb des Schleifeninhalts notiert wird, wird lediglich eine Durchlaufbedingung formuliert. Dafür, dass diese Bedingung irgendwann nicht mehr erfüllt ist und die Schleife abgebrochen wird, muss der Schleifeninhalt sorgen. Funktioniert das nicht, so entsteht eine so genannte Endlosschleife. Beginnen wir mit einem solchen Beispiel, um die Problematik zu verdeutlichen:

```
var x = 1;
while(x == 1)
   document.write(x + " ");
```

Mit so wenig JavaScript-Code können Sie jeden Browser ärgern. Im günstigsten Fall reagiert er nach wenigen Sekunden mit einem Dialogfenster, welches nachfragt, ob das Script abgebrochen werden soll. Es kann aber auch sein, dass die gesamte Browser-Anwendung nicht mehr reagiert und abstürzt.

Deklariert wird eine Variable x mit dem Initialwert 1. Dann wird als Durchlaufbedingung für eine while-Schleife geprüft, ob x den Wert 1 hat. Das ist der Fall, also wird die abhängige document.write()-Anweisung ausgeführt. Der Wert von x wird aber nirgends geändert. Daher bleibt die Schleifendurchlaufbedingung immer erfüllt und die Schleife bricht nie ab.

Ein ordentlicheres Beispiel für eine while-Schleife ist dieses:

```
x = 1;
while(x != 5) {
   x = parseInt(Math.random() * 10);
   document.write(x + " ");
}
```

In diesem Beispiel wird ebenfalls `x` mit 1 initialisiert. Die Durchlaufbedingung ist, dass `x` nicht den Wert 5 hat. Da `x` zunächst 1 ist, ist die Bedingung also erfüllt und die Programmausführung gerät in die Schleife. Innerhalb der Schleife wird jedoch der Wert für `x` neu ermittelt. Allerdings ist das Ergebnis nicht bekannt. Denn es handelt sich um eine Zufallszahl zwischen 0 und 9 (auf die Methode `Math.random()` gehen wir später noch ein). Anschließend wird der Wert von x ins Browser-Fenster geschrieben. Da `x` in jedem Schleifendurchgang einen neuen zufälligen Wert zwischen 0 und 9 erhält, ist einigermaßen sicher, dass nach einer Weile auch mal der Wert 5 vorkommt. Dann ist die Durchlaufbedingung nicht mehr erfüllt und die Schleife bricht ab.

Bei do-while-Schleifen wird zuerst der Schleifeninhalt ausgeführt und erst anschließend wird die Durchlaufbedingung geprüft. Dadurch ist sichergestellt, dass die Schleife mindestens einmal durchlaufen wird. In manchen Fällen kann dies wichtig sein. Ein Beispiel:

```
var x = 1;
do {
  document.write("ein Satz mit x<br>");
  x += 1;
} while(x % 2 == 1)
```

Im Beispiel wird x mit 1 initialisiert. Anschließend tritt die Programmausführung in den Schleifeninhalt ein, markiert durch `do`. Die Anweisungen innerhalb des Schleifenkörpers sind als Block in geschweifte Klammern gesetzt, da es mehr als eine Anweisung ist. Die Schleifendurchlaufbedingung wird am Ende nach der schließenden geschweiften Klammer formuliert. Die Syntax ist dabei die gleiche wie bei while-Schleifen. Im Beispiel lautet die Schleifendurchlaufbedingung »wenn x geteilt durch 2 einen Rest von 1 ergibt«. Da x innerhalb des ersten Schleifendurchlaufs durch x += 1 auf 2 erhöht wird, ist die Bedingung, wenn sie zum ersten Mal geprüft wird, bereits nicht erfüllt. Die Schleife bricht also nach dem einen Durchlauf ab.

Schleifenkontrolle

In manchen Fällen kann es erforderlich sein, zusätzlich zur Schleifendurchlaufbedingung innerhalb des Schleifeninhalts Abbruchbedingungen zu formulieren oder auch den nächsten Schleifendurchlauf zu erzwingen, bevor der gesamte Schleifeninhalt abgearbeitet wurde. Für solche Zwecke stehen die Statements `break` und `continue` zur Verfügung: `break` bricht eine Schleifenabarbeitung sofort ab, `continue` erzwingt an der Stelle, wo es notiert ist, den nächsten Schleifendurchlauf. Beide Statements hängen typischerweise von einer if-Abfrage ab. Dazu folgendes Beispiel:

```
x = 1;
while(x != 5) {
   x = parseInt(Math.random() * 10);
   if(x == 0)
      break;
   y = 10 / x;
   document.write(y + "<br>");
}
```

Das Beispiel erweitert jenes zu while-Schleifen. In diesem Fall ist jedoch zusätzlich zur Schleifendurchlaufbedingung `x != 5` noch eine Bedingung `x == 0` formuliert, bei der, falls sie erfüllt ist, die Schleifenverarbeitung abgebrochen wird. Grund ist in unserem Beispielfall zu verhindern, dass anschließend eine Division durch 0 durchgeführt wird, was zu einem ungültigen Ergebnis führen würde. Das Statement `break` wird wie eine Anweisung notiert und mit Semikolon abgeschlossen. Das Gleiche gilt für `continue`.

7.3.6 Funktionen, Parameter und Return-Werte

Funktionen haben wir in einigen Beispielen bereits kennen gelernt. Das erste Charakteristische daran ist: Anweisungen, die in Funktionen stehen, werden nicht gleich beim Einlesen des Scripts ausgeführt, sondern erst dann, wenn die Funktion aufgerufen wird. Der enthaltene Code ist wiederverwendbar und kann so oft ausgeführt werden, wie die Funktion aufgerufen wird.

Dieses Merkmal teilen Funktionen mit der klassischen Unterprogrammtechnik. Funktionen haben aber noch weitere Vorteile: Sie können Werte übergeben bekommen (Parameter), diese verarbeiten und Werte an die aufrufende Anweisung zurückgeben (Return-Werte oder Rückgabewerte).

Die Arbeitsweise von Funktionen soll an einem Komplettbeispiel verdeutlicht werden. Das Beispiel stellt eine eigene Suche-Ersetze-Funktion für Zeichenketten vor, die es in dieser Form in JavaScript nicht gibt, die aber praktische Dienste leisten kann. Die Funktion erwartet einen Input, den sie aus Formulareingaben des Anwenders erhält, und liefert einen Output, der dynamisch ins Anzeigefenster geschrieben wird. Das HTML-Dokument mit der Beispielfunktion hat folgenden Quelltext:

Listing 7.6: HTML-Dokument mit eigener Suche-Ersetze-Funktion

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Suchen und Ersetzen</title>
<script type="text/javascript">
function string_replace(string, search, replace) {
   var new_string = "";
   var i = 0;
   while(i < string.length) {
      if(string.substring(i, i + search.length) == search) {
         new_string = new_string + replace;
         i = i + search.length - 1;
      }
      else
         new_string = new_string +
                      string.substring(i, i + 1);
   i++;
  }
  return new_string;
}
</script>
</head>
<body>
<form action="javascript:void(0)"
      style="background-color:#FFDDAA; padding:10px;">
<p>Suchtext:<br>
<input type="text" id="text" style="width:360px;
border:solid 1px gray"></p>
<p>Suche im Suchtext:<br>
<input type="text" id="search_for" style="width:120px;
border:solid 1px gray"></p>
<p>Ersetze es durch:<br>
<input type="text" id="replace_by" style="width:120px;
border:solid 1px gray"></p>
<p><button type="button" style="width:120px;
border:solid 1px gray"
onClick="document.getElementById('result').innerHTML =
   string_replace(document.getElementById('text').value,
   document.getElementById('search_for').value,
   document.getElementById('replace_by').value)">
   Ergebnis
</button></p>
</form>
<p id="result"> </p>
</body>
</html>
```

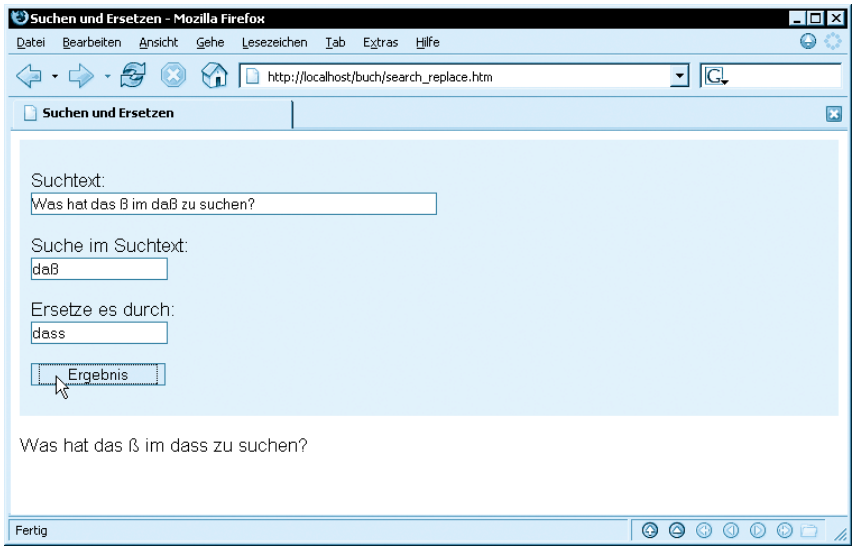


Abbildung 7.7: Formular zum Testen der Suche-Ersetze-Funktion

Im Formular der Seite kann der Anwender einen Text eingeben, eine Zeichenkette, nach der im Text gesucht werden soll, und eine Zeichenkette, durch die sie ersetzt werden soll. Beim Klick auf den Button ***Ergebnis*** wird der Text unterhalb des Formulars ausgegeben, und zwar so, dass darin eventuell gefundene Stellen ersetzt sind. Dazu wird mit `onClick=` die Funktion `string_replace()` aufgerufen, die in einem script-Bereich im Dateikopf notiert ist.

Funktionen beginnen mit dem Schlüsselwort `function`. Dahinter folgt der frei wählbare Name der Funktion. Unmittelbar hinter dem Namen werden ein Paar runde Klammern `()` notiert. Innerhalb der Klammern können Parameternamen notiert werden. Die Parameternamen können Sie wie Variablennamen frei bestimmen.

Funktions- und Parameternamen müssen den üblichen Konventionen genügen. Es sind also nur Buchstaben A bis Z, a bis z, Ziffern 0 bis 9 und der Unterstrich erlaubt. Das erste Zeichen darf keine Ziffer sein.

Der gesamte Funktionsinhalt ist ein Block und wird in geschweifte Klammern `{` und `}` eingeschlossen. Innerhalb des Funktionsblocks kann auf die übergebenen Parameter wie auf Variablen zugegriffen werden. Die Parameter enthalten Werte, die der Funktion beim Aufruf übergeben werden müssen. In unserem Beispiel erwartet die Funktion `string_replace()` drei Parameter: `string`, `search` und `replace`. Beim Aufruf der Funktion werden die drei erwarteten Parameter übergeben. Es handelt sich um die mittels DOM-Zugriffssyntax ermittelten Werte aus den Formularfeldern mit den id-Namen `text`, `search_for` und `replace_by`. In den drei Parametern stehen also, wenn die Funktion sie übergeben bekommt, die vom Anwender in den entsprechenden Feldern eingegebenen Werte.

Die Funktion selbst deklariert eine Variable namens `new_string`. Die Arbeit der Funktion besteht darin, den Wert des Parameters `string` Zeichen für Zeichen abzuarbeiten, eventuelle Ersetzungen vorzunehmen und die Variable `new_string` nach und nach mit der Kopie von `string` inklusive eventueller Ersetzungen zu füllen.

Am Ende wird `new_string` mit `return` zurückgegeben. Die Fähigkeit, solche Return-Werte an die funktionsaufrufende Anweisung zurückzugeben, ist eine besonders wertvolle Einrichtung. Gerade in Verbindung mit Parametern wird eine Funktion somit zu einer eigenen kleinen Datenverarbeitungsmaschine, die einen Input erhält und einen Output liefert. Die aufrufende Anweisung muss nur den Input übergeben und den Output empfangen.

Betrachten wir noch mal genau den Aufruf der Funktion `string_replace()` im HTML-Code des Buttons *Ergebnis*:

```
onClick="document.getElementById('result').innerHTML =
   string_replace(document.getElementById('text').value,
   document.getElementById('search_for').value,
   document.getElementById('replace_by').value)
```

Die Zuweisung an `onClick=` besteht aus einer einzigen JavaScript-Anweisung. Dass diese etwas länger ist, liegt an den leider etwas langen DOM-Zugriffen, die darin notiert sind. Die Anweisung selbst besteht aus einer Zuweisung. Links des Zuweisungsoperators `=` muss der »Speicher« stehen, dem der Wert zugewiesen werden soll. Innerhalb von JavaScript ist dies häufig eine Variable. In unserem Fall ist es jedoch die Eigenschaft `innerHTML` des Elements mit dem id-Namen `result`. Das ist das p-Element, welches unten im HTML-Dokument notiert ist. Die Funktion `string_replace()` schreibt ihren Return-Wert durch diese Zuweisung also direkt in das p-Element.

Funktionen müssen keine Parameter erwarten und keine Return-Werte liefern, aber sie können es. In Ihrem JavaScript-Code können Sie Funktionen einfach als Unterprogramme zur besseren Strukturierung benutzen oder als eigene kleine Programme, die aus einem Input einen Output erzeugen. In jedem Fall müssen Sie Code in Funktionen stecken, der nicht sofort beim Einlesen des Dokuments ausgeführt werden soll.

7.3.7 Die Objektstruktur von JavaScript 1.5

Wir haben nun die wichtigsten syntaktischen Regeln von JavaScript kennen gelernt. Jetzt wird es darum gehen, auf was man mit JavaScript eigentlich zugreifen kann. Auf Inhalte der aktuell angezeigten HTML-Seite im Browser kann über DOM-Syntax zugegriffen werden. Wir haben aber auch schon Beispiele vorgestellt, in denen das aktuelle Datum oder eine Zufallszahl ermittelt wurde.

Für all diese »Leistungen« stellt JavaScript ein Set an vordefinierten Objekten zur Verfügung. Um mit den Objekten arbeiten zu können, müssen Sie wissen, welche Objekte für welche Aufgabenbereiche zuständig sind, welche Eigenschaften und Methoden sie besitzen und wie mit ihnen gearbeitet werden kann.

Alle von JavaScript zur Verfügung gestellten Objekte mit Ausnahme von allem, was mit DOM zu tun hat, gehören zum so genannten *Core* (Kern) von JavaScript 1.5. Dazu zählen

Objekte für Zeit und Datum, mathematische Operationen, Zeichenmanipulation, Serienvariablen (Arrays), Fensterzugriff (Popups, Frames) usw.

Wir werden auf die wichtigsten Objekte eingehen und beschreiben, wie Sie damit arbeiten können.

Objekte, Instanzen, Methoden und Eigenschaften

Zu den meisten Objekten gehören objektspezifische Eigenschaften und Methoden. Auf Eigenschaften und Objekte wird über den Punktoperator zugegriffen. Schema:

```
Objekt.Eigenschaft
Objekt.Methode()
```

Eine Objekteigenschaft ist nichts anderes als eine Variable, die zum Objekt gehört, und eine Objektmethode ist nichts anderes als eine Funktion, die zum Objekt gehört. Bei einigen Objektmethoden kommt es daher auch vor, dass diese Parameter erwarten und Return-Werte zurückgeben.

Manche Eigenschaften können geändert, manche dagegen nur gelesen werden. Eigenschaften, die geändert werden dürfen, können also auch links von einer Zuweisung notiert werden, z.B. die bereits bekannte innerHTML-Eigenschaft des DOM-Objekts für HTML-Elemente. Nur lesbare Eigenschaftswerte dürfen dagegen nur rechts einer Zuweisung stehen, also zugewiesen werden, z.B. `document.URL`.

Mit manchen Objekten kann direkt gearbeitet werden (z.B. mit dem Math-Objekt). Deshalb ist es erlaubt, im JavaScript-Code ohne weiteres eine Objektmethode wie `Math.random()` aufzurufen.

Andere Objekte erfordern es, zuerst eine so genannte Instanz zu erzeugen (z.B. das Date-Objekt oder das Array-Objekt). Instanzen sind »Arbeitskopien« eines Objekts. Die Objektinstanz wird in einer frei benennbaren Variable gespeichert, die dann auch als Objektvariable bezeichnet wird. Ein Beispiel für das Date-Objekt:

```
var now = new Date();
document.write(now.getFullYear());
```

In diesem Beispiel ist `now` die Objektvariable. In ihr wird durch Initialzuweisung des so genannten Konstruktors `new Date()` eine neue Instanz des Date-Objekts gespeichert. Über die Objektvariable ist anschließend der Zugriff auf Eigenschaften und Methoden des Objekts möglich. Im Beispiel ist `getFullYear()` eine Methode des Date-Objekts, die aber nur über eine Objektinstanz, wie es bei `now` der Fall ist, aufgerufen werden kann.

Objekte können Unterobjekte enthalten. Unterobjekte unterscheiden sich syntaktisch nicht von Objekteigenschaften. Erkennbar sind sie letztlich erst, wenn auf ihre Eigenschaften und Methoden zugegriffen wird. Das Schema sieht dann so aus:

```
Objekt.Unterobjekt.Eigenschaft
Objekt.Unterobjekt.Methode()
```

7.3.8 Datum und Zeit mit JavaScript

Wenn Sie in JavaScript beispielsweise auf das aktuelle Datum und die aktuelle Uhrzeit zugreifen oder ermitteln möchten, auf welchen Wochentag der 31.12.2004 fiel, dann benötigen Sie das Date-Objekt.

Das Date-Objekt erfordert eine Instanz, um auf seine Eigenschaften und Methoden zugreifen zu können. Beim Initialisieren der Objektvariable haben Sie die Auswahl, ob Sie ein Datumsobjekt für den aktuellen Zeitpunkt wünschen oder ob das Datumsobjekt für einen beliebigen bestimmten Zeitpunkt in der Vergangenheit oder Zukunft erstellt werden soll. In der Praxis wird meistens nur eine Objektinstanz für den aktuellen Zeitpunkt benötigt, weil beispielsweise das aktuelle Datum ausgegeben werden soll oder ausgehend vom aktuellen Datum etwas berechnet werden soll, etwa die Anzahl der restlichen Tage im Jahr. Wenn Sie dagegen den Wochentag eines vergangenen oder künftigen Datums ermitteln möchten, müssen Sie beim Initialisieren der Objektvariable das gewünschte Datum übergeben.

Wenn Sie eine Objektinstanz für den aktuellen Zeitpunkt möchten, notieren Sie die Initialisierung einfach wie folgt:

```
var now = new Date();
```

Dabei können Sie anstelle von `now` auch einen anderen Namen vergeben.

Wenn Sie eine Objektinstanz für ein bestimmtes Datum benötigen, wobei aber die Uhrzeit keine Rolle spiel, ist die folgende Variante sinnvoll:

```
var last_day_2004 = new Date(2004,11,31);
```

In diesem Beispiel ist `last_day_2004` der frei gewählte Name der Objektvariable. Bei der Konstruktor-Funktion `Date()` werden drei Parameter übergeben. Der erste ist das gewünschte Jahr, der zweite der Monat und der dritte der Tag im Monat. Beim zweiten Parameter ist zu beachten, dass für Monate Werte zwischen 0 und 11 übergeben werden müssen (und nicht wie im Kalender üblich zwischen 1 und 12!). Der im Beispiel übergebene Wert 11 bedeutet also »Dezember«.

Wenn die Uhrzeit eine Rolle spielen soll, können Sie auch diese mit initialisieren:

```
var start_2005 = new Date(2005,0,1,0,0,0);
```

Dabei werden an `Date()` die Parameter Jahr, Monat, Tag, Stunden, Minuten, Sekunden übergeben. Das Beispiel definiert eine Objektinstanz für den 1. Januar 2005, 0:00:00 Uhr.

Der im Objekt anfänglich gespeicherte Zeitpunkt ist durch die `set...`-Methoden des Date-Objekts im Programmverlauf veränderbar.

Methoden des Date-Objekts

Das Date-Objekt stellt diverse Methoden bereit, um Einzelwerte des gespeicherten Datums oder der Uhrzeit zu ermitteln oder zu verändern. Außerdem werden noch ein paar Konvertierungsmethoden bereitgestellt.

Über die Variable der Objektinstanz ist der Zugriff möglich, also etwa `now.getDate()` oder `start_2005.parse()`.

Methode	Bedeutung
`getDate()`	Ermittelt den Monatstag.
`getDay()`	Ermittelt den Wochentag, und zwar in Form von Zahlen von 0 (für Sonntag), 1 (für Montag) usw. bis 6 (für Samstag).
`getFullYear()`	Ermittelt das Jahr.
`getHours()`	Ermittelt den Stundenwert der Uhrzeit.
`getMilliseconds()`	Ermittelt die Anzahl Millisekunden seit der letzten vollen Sekunde.
`getMinutes()`	Ermittelt den Minutenwert der Uhrzeit.
`getMonth()`	Ermittelt den Monat (Rückgabewerte sind 0 bis 11!).
`getSeconds()`	Ermittelt den Sekundenwert der Uhrzeit.
`getTime()`	Ermittelt, wie viele Millisekunden seit dem 1. Januar 1970, 0:00:00, bis zu dem im Objekt gespeicherten Zeitpunkt vergangen sind.
`getTimezoneOffset()`	Ermittelt die Zeitzonenabweichung der Lokalzeit gegenüber Greenwich Mean Time (GMT oder UTC) in Minuten.

`getUTCDate()`
`getUTCDay()`
`getUTCFullYear()`
`getUTCHours()`
`getUTCMilliseconds()`
`getUTCMinutes()`
`getUTCMonth()`
`getUTCSeconds()`
	Wie die entsprechenden Methoden ohne UTC, jedoch auf Greenwich Mean Time (GMT oder UTC) bezogen und nicht auf die lokale Rechnerzeit.

`setDate()`	Setzt den gespeicherten Monatstag auf einen anderen Wert.
`setFullYear()`	Setzt das gespeicherte Jahr auf einen anderen Wert.
`setHours()`	Setzt den gespeicherten Stundenwert der Uhrzeit auf einen anderen Wert.
`setMilliseconds()`	Setzt die Millisekunden auf einen anderen Wert.
`setMinutes()`	Setzt den gespeicherten Minutenwert der Uhrzeit auf einen anderen Wert.
`setMonth()`	Setzt den gespeicherten Monat auf einen anderen Wert (0 bis 11 übergeben!).
`setSeconds()`	Setzt den gespeicherten Sekundenwert der Uhrzeit auf einen anderen Wert.

Tabelle 7.7: Methoden des Date-Objekts

Methode	Bedeutung
`setTime()`	Setzt den kompletten gespeicherten Zeitpunkt neu. Erwartet wird eine Zahl, die als Anzahl Millisekunden seit dem 1.1.1970, 0:00:00 Uhr interpretiert wird.

`setUTCDate()`
`setUTCFullYear()`
`setUTCHours()`
`setUTCMilliseconds()`
`setUTCMinutes()`
`setUTCMonth()`
`setUTCSeconds()`
	Wie die entsprechenden Methoden ohne UTC, jedoch auf Greenwich Mean Time (GMT oder UTC) bezogen und nicht auf die lokale Rechnerzeit.

`toGMTString()`	Wandelt den gespeicherten Objektzeitpunkt ins IETF-Format um, z. B. Sat, 5 Feb 2005 12:09:36 GMT.
`toLocalString()`	Wandelt den gespeicherten Objektzeitpunkt in ein lokalisiertes Format um, das umgebungsabhängig vom Anwender ist.

Tabelle 7.7: Methoden des Date-Objekts (Fortsetzung)

Anwendungsbeispiel: Datums-/Uhrzeit-Anzeige

Wir erweitern die in ***Abschnitt 7.2.3*** vorgestellte Funktion dahin gehend, dass diese nun nicht nur einmalig das aktuelle Datum in ein gewünschtes Element schreibt, sondern jede Sekunde Datum und Uhrzeit. Der JavaScript-Code der Funktion lautet:

```
function show_clock(id_name) {
   var weekdays =  new Array("Sonntag","Montag","Dienstag",
                   "Mittwoch","Donnerstag","Freitag","Samstag");
   var now = new Date();
   var day = now.getDate();
   var month = now.getMonth() + 1;
   var year = now.getFullYear();
   var weekday = now.getDay();
   var hours = now.getHours();
   var minutes = now.getMinutes();
   var seconds = now.getSeconds();
   var day0  = ((day < 10) ? "0" : "");
   var month0  = ((month < 10) ? ".0" : ".");
   var hours0  = ((hours < 10) ? "0" : "");
   var minutes0  = ((minutes < 10) ? ".0" : ":");
   var seconds0  = ((seconds < 10) ? ".0" : ":");
   var output = weekdays[weekday] + ", " + day0 + day + month0 +
                month + "." + year + ", " + hours0 + hours +
                minutes0 + minutes + seconds0 + seconds +
                " Uhr";
   document.getElementById(id_name).innerHTML = output;
   window.setTimeout("show_clock('" + id_name + "')", 1000);
}
```

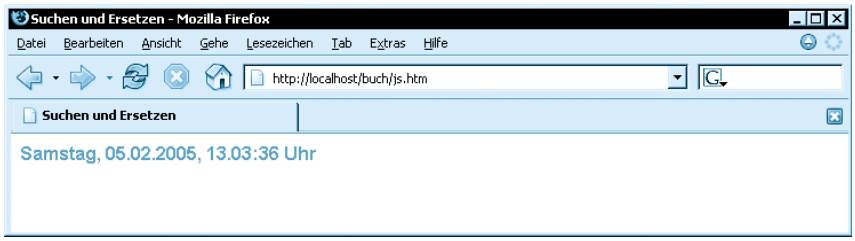


Abbildung 7.8: Sekündlich aktualisierte Datum-Uhrzeit-Anzeige

Die Funktion heißt nun `show_clock()` und erwartet als Parameter den id-Namen eines HTML-Elements, als dessen Elementinhalt die Datums-/Uhrzeit-Anzeige laufend neu geschrieben werden soll. Im abgebildeten Beispiel wurde in HTML Folgendes notiert:

```
<div id="running_clock" style="font-size:16px; font-weight:bold; color:red"></
div>
```

Ein leeres div-Element also mit dem id-Namen `running_clock`. Aufgerufen wird die Funktion `show_clock()` am besten im einleitenden `<body>`-Tag mit `onLoad=`. Im abgebildeten Beispiel:

```
<body onLoad="show_clock('running_clock')">
```

Innerhalb der Funktion wird zunächst ein Array mit den deutschen Wochentagnamen definiert. Anschließend wird mit `now = new Date()` eine Instanz des Date-Objekts für den aktuellen Zeitpunkt erzeugt. Die einzelnen Bestandteile des Zeitpunkts, also Tag, Monat, Jahr, Stunden, Minuten und Sekunden, werden daraufhin über die entsprechenden `get...`-Methoden des Objekts ermittelt und in Variablen gespeichert.

Die fünf Anweisungen, die Variablen mit einer 0 am Ende des Namens initialisieren, dienen dazu, führende Nullen zu erzeugen, falls der Return-Wert einer der get...-Methoden kleiner als 10 war. Nur so kann später beispielsweise das Datum in der Form 05.02.2005 dargestellt werden.

Das Konstrukt, das beim Erzeugen der führenden Nullen erzeugt wird, ist übrigens eine Spezialform von Kontrollstruktur, die im Prinzip eine if-else-Verzweigung leistet, jedoch in verkürzter Form und ohne die bekannten Schlüsselwörter. Das Statement hat die folgende Form:

```
Bedingung ? if-Zweig : else-Zweig;
```

Jeder der beiden Zweige kann nur aus einer Anweisung bestehen. Das kann natürlich auch ein Funktionsaufruf sein. In unserer Funktion wird beispielsweise der Variablen `day0` im if-Zweig, also wenn `day` kleiner als 10 ist, die Zeichenkette 0 zugewiesen, und im else-Zweig eine leere Zeichenkette. Bei anderen dieser Statements werden auch gleich die Trennzeichen zwischen den Datums-/Uhrzeit-Bestandteilen, also Punkte und Doppelpunkte, in diese Variablen eingefügt.

In der Variablen `output` wird schließlich die gesamte, auszugebende Zeichenkette zusammengebaut. Dabei wird auch der Wochentag mit berücksichtigt. Anschließend wird durch Zuweisung an die innerHTML-Eigenschaft des gewünschten HTML-Elements der Inhalt von `output` dynamisch ins Browser-Fenster geschrieben.

In der letzten Anweisung ruft sich die Funktion `show_clock()` selber wieder auf, und zwar mit einer Zeitverzögerung von 1000 Millisekunden, also von einer Sekunde. Auf diese Weise wird die Datums-/Uhrzeitanzeige jede Sekunde aktualisiert, solange die Seite angezeigt bleibt.

7.3.9 Mathematische Operationen mit JavaScript

Das Math-Objekt stellt Eigenschaften und Methoden zur Verfügung, die das Rechnen mit JavaScript erweitern und über die Möglichkeiten von Grundrechenarten hinausgehen. Eine Objektinstanz ist beim Math-Objekt nicht erforderlich – seine Eigenschaften und Methoden können direkt mit `Math.Eigenschaftsname` bzw. `Math.Methodenname()` angesprochen werden.

Eigenschaften des Math-Objekts

Die Eigenschaften des Math-Objekts sind eigentlich nur bekannte mathematische Konstanten. Keine der Eigenschaft ist daher änderbar.

Eigenschaft	Bedeutung
`E`	Eulersche Konstante (ca. 2.718)
`LN2`	Natürlicher Logarithmus zur Basis 2 (ca. 0,693)
`LN10`	Natürlicher Logarithmus zur Basis 10 (ca. 2,302)
`LOG2E`	Logarithmus zur Basis 2 (ca. 1,442)
`LOG10E`	Eulersche Konstante zur Basis 2 (ca. 0,434)
`PI`	Zahl PI (3,14159)
`SQRT1_2`	Konstante für Quadratwurzel aus 0.5 (ca. 0,707)
`SQRT2`	Konstante für Quadratwurzel aus 2 (ca. 1,414)

Tabelle 7.8: Eigenschaften des Math-Objekts

Methoden des Math-Objekts

Die Methoden des Math-Objekts erlauben beispielsweise trigonometrische Berechnungen, Ganzzahlrundung und die bereits bekannte Ermittlung einer Zufallszahl.

Eigenschaft	Bedeutung
`abs(x)`	Gibt x als positiven Wert zurück.
`acos(x)`	Gibt den Arcus Cosinus von x zurück.

Tabelle 7.9: Methoden des Math-Objekts

Eigenschaft	Bedeutung
`asin(x)`	Gibt den Arcus Sinus von x zurück.
`atan(x)`	Gibt den Arcus Tangens von x zurück.
`ceil(x)`	Gibt die nächsthöhere Ganzzahl von x (Kommazahl) zurück.
`cos(x)`	Gibt den Cosinus von x zurück.
`exp(x)`	Gibt den Exponentialwert von x auf Basis der Eulerschen Konstanten zurück.
`floor(x)`	Gibt die nächstniedrigere Ganzzahl von x (Kommazahl) zurück.
`log(x)`	Gibt den natürlichen Logarithmus, angewendet auf x, zurück.
`max(x,y)`	Gibt den größeren der beiden übergebenen Werte zurück.
`min(x,y)`	Gibt den kleineren der beiden übergebenen Werte zurück.
`pow(x,y)`	Gibt x hoch y zurück.
`random()`	Gibt eine Zufallszahl (Kommazahl) zwischen 0 und 1 zurück.
`round(x)`	Gibt die kaufmännisch gerundete Ganzzahl von x zurück.
`sin(x)`	Gibt den Sinus von x zurück.
`sqrt(x)`	Gibt die Quadratwurzel von x zurück.
`tan(x)`	Gibt den Tangens von x zurück.

Tabelle 7.9: Methoden des Math-Objekts (Fortsetzung)

Anwendungsbeispiel: Kaufmännisches Runden mit Nachkommastellen

Jede numerische Variable in JavaScript stellt ein eigenes Objekt dar, auf das sich unter anderem eine Methode namens `toFixed()` anwenden lässt. Diese erzwingt für den Wert der Variablen eine fixe Anzahl von Nachkommastellen. Dies ist ein häufiger Praxiswunsch, wenn etwa aus einer Berechnung von Werten, die ein Anwender eingegeben hat, ein Preis ermittelt werden soll und dieser aber mit genau zwei Nachkommastellen angezeigt werden soll. Nun schneidet `toFixed()` einfach zu viele Nachkommastellen ab. Gerade bei Preisen jedoch wird meist eine kaufmännische Rundung erwünscht.

Die nachfolgende, allgemein geschriebene Funktion erlaubt das Runden auf eine gewünschte Anzahl Nachkommastellen (inklusive 0 für keine Nachkommastellen) und rundet dabei kaufmännisch auf:

```
function round_float(x, pdp) {
   x *= Math.pow(10, pdp);
   y = Math.round(x);
   y /= Math.pow(10, pdp);
   return(y.toFixed(pdp));
}
```

Die Funktion `round_float()` erwartet zwei Parameter. Der erste, x, ist eine Zahl, die kaufmännisch gerundet werden soll. Der zweite Parameter, `pdp`, ist die Anzahl Nachkommastellen, die ausgegeben werden sollen.

Die Funktion bedient sich eines Tricks. Wenn man beispielsweise den Wert 3.267 kaufmännisch runden möchte, und zwar mit zwei Nachkommastellen, dann ist dies erreichbar, indem man die Zahl mit 100 multipliziert (Ergebnis: 326.7), diesen Wert ganzzahlig rundet (Ergebnis: 327) und das Ergebnis dann wieder durch 100 dividiert (neues Ergebnis: 3.27).

Genau das tut die obige Funktion mit den ersten drei Anweisungen. Da jedoch die Anzahl gewünschter Nachkommastellen nicht fest steht, sondern mit dem Parameter `pdp` übergeben wird, arbeitet die Funktion mit `Math.pow()`. Damit wird x mit einer Potenz von 10 multipliziert, wobei die Potenz durch den Parameter `pdp` festgelegt wird. In der zweiten Anweisung wird das so entstandene Zwischenergebnis mit `Math.round()` ganzzahlig gerundet. In der dritten Anweisung wird das gerundete Ergebnis wieder durch die gleiche Zehnerpotenz geteilt, wie zuvor multipliziert wurde.

Das Ergebnis stimmt nun zwar, enthält aber bei weniger Nachkommastellen als gewünscht keine Nullen am Ende. Ein Aufruf wie `round_float(4.097, 2)`ergibt bis dahin nur 4.1, nicht 4.10. Um Letzteres zu erreichen, gibt die Funktion nicht einfach y zurück, sondern wendet nun noch einmal die toFixed()-Methode auf `y` an, und zwar mit so viel gewünschten Nachkommastellen wie in `pdp` angegeben. Dieser Wert wird mit `return` an eine aufrufende Anweisung zurückgegeben.

7.3.10 Zeichenketten (Strings) in JavaScript

Zeichenketten können aus einem einzigen, aber auch aus sehr vielen Zeichen bestehen. So liefert etwa `document.getElementsByTagName('html')[0].innerHTML` den gesamten HTML-Quelltext zwischen `<html>` und `</html>` in einer einzigen Zeichenkette. Eine Eigenschaft und diverse Methoden unterstützen bei der Programmierung dabei, Zeichenketten zu verarbeiten.

Das Zeichenkettenobjekt erfordert keine explizite Instanz. Stattdessen ist jede Variable, in der eine Zeichenkette gespeichert ist, automatisch auch eine Instanzvariable des Zeichenkettenobjekts. Wenn Sie also beispielsweise eine Variable namens `name` haben, können Sie z.B. mit `name.length` die Anzahl der darin gespeicherten Zeichen ermitteln, oder mit `name.toLowerCase()` alle darin enthaltenen Großbuchstaben in Kleinbuchstaben umwandeln.

Eigenschaften des Zeichenketten-Objekts

Die einzige Eigenschaft ist `length`. Diese Eigenschaft kann nicht geändert werden und enthält die Anzahl Zeichen der Zeichenkette.

Methoden des Zeichenketten-Objekts

Die Methoden des Zeichenketten-Objekts erlauben Zeichenkettenmanipulation und Durchsuchung. Einige Methoden arbeiten mit regulären Ausdrücken. Dieses Thema behandeln wir jedoch nicht in Zusammenhang mit JavaScript, sondern erst bei PHP.

Methode	Bedeutung
`charAt(x)`	Liefert das Zeichen zurück, das an Position x steht.
`charCodeAt(x)`	Liefert den Zeichenwert (ISO-8859-1) des Zeichens zurück, das an Position x steht.
`fromCharCode(x,...)`	Interpretiert eine Reihe von numerischen Parametern als Zeichenwert (ISO-8859-1) und erzeugt eine entsprechende Zeichenkette.
`indexOf(str)`	Gibt zurück, an welcher Stelle die Teilzeichenkette `str` erstmals beginnt (0-index-basiert).
`lastIndexOf()`	Gibt zurück, an welcher Stelle die Teilzeichenkette `str` letztmalig beginnt (0-index-basiert).
`match(/regExp/)`	Durchsucht die Zeichenkette nach dem regulären Ausdruck `regExp`. Gibt alle gefundenen Treffer als Array zurück.
`replace(/regExp/,str)`	Durchsucht die Zeichenkette mithilfe des regulären Ausdrucks `regExp`. Ersetzt gefundene Passagen durch die Zeichenkette `str`.
`search(/regExp/)`	Durchsucht die Zeichenkette nach dem regulären Ausdruck `regExp`. Gibt die 0-index-basierte Position des ersten Treffers zurück, oder -1, wenn nichts gefunden wurde.
`slice(x,y)`	Gibt die Teilzeichenkette zurück, deren erstes Zeichen an Position x beginnt und deren letztes Zeichen an Position y-1 steht. Positionsangaben sind 0-index-basiert.
`split(str)`	Zerlegt die Zeichenkette in mehrere Teilzeichenketten, wobei `str` das Zeichen oder die Zeichenkette ist, die als Trenner fungiert. Gibt die Teilzeichenketten als Array zurück.
`substr(x,y)`	Gibt die Teilzeichenkette zurück, die an Position x beginnt und y Zeichen lang ist. Position x ist 0-index-basiert.
`toLowerCase()`	Wandelt alle Großbuchstaben der Zeichenkette in entsprechende Kleinbuchstaben um und gibt das Ergebnis zurück.
`toUpperCase()`	Wandelt alle Kleinbuchstaben der Zeichenkette in entsprechende Großbuchstaben um und gibt das Ergebnis zurück.

Tabelle 7.10: Methoden des Zeichenketten-Objekts

Einige der Methoden arbeiten mit Zeichenpositionen, wobei jeweils von »0-index-basiert« die Rede ist. Damit ist Folgendes gemeint: Das erste Zeichen einer Zeichenkette hat die Position 0. Das zweite hat die Position 1, das dritte die Position 2 usw. Wenn man also das 22. Zeichen meint, ist dessen Index innerhalb der Zeichenkette 21.

Anwendungsbeispiel: Horizontaler Courier-Ticker

Die Metapher »Börsenticker« erfreut sich auf vielen Websites immer noch großer Beliebtheit, um »brandaktuelle Neuigkeiten auf der Site« optisch auffällig zu präsentieren. Im nachfolgenden Beispiel zeigen wir, wie Sie einen einfachen Ticker dieser Art mit JavaScript realisieren. Der vollständige Quelltext des HTML-Dokuments mit dem Script lautet:

Listing 7.7: HTML-Dokument mit Ticker

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<title>Ticker</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
<style type="text/css">
#ticker { position:absolute; top:10px; left:20px; color:#FF0000;
          font-family:Courier New,courier;
          font-size:15px; font-weight:bold; white-space:nowrap; }
#page { position:absolute; top:50px; left:20px; right:20px; }
</style>
<script type="text/javascript">
var text_now = "";
var chars = 0;
var effect = "rtl";

function fill(n) {
 var empty_chars = "";
 for(var i = 0; i < n; ++i)
  empty_chars += " ";
 return(empty_chars);
}

function show_ticker(text, delay) {
 if(effect == "rtl") {
  chars += 1;
  text_now = fill(text.length - chars);
  text_now += text.substring(0, chars);
  if(chars >= text.length) effect = "ltn";
 }
 else if(effect == "ltn") {
  chars -= 1;
  text_now =
     text.substring((text.length - chars),(text.length - 1));
  text_now += fill(text.length - chars);
  if(chars <= 0) effect = "rtl";
 }
 document.getElementById("ticker").innerHTML = text_now;
 window.setTimeout
    ("show_ticker('" + text + "', " + delay + ")", delay);
}
</script>
</head>
<body onload="show_ticker('+++ Was Sie schon immer über
 Blindtexte wissen wollten +++', 200)">
<div id="ticker"></div>
<div id="page">
   Ein Blindtext ist ein Text wie dieser hier.
</div>
```

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

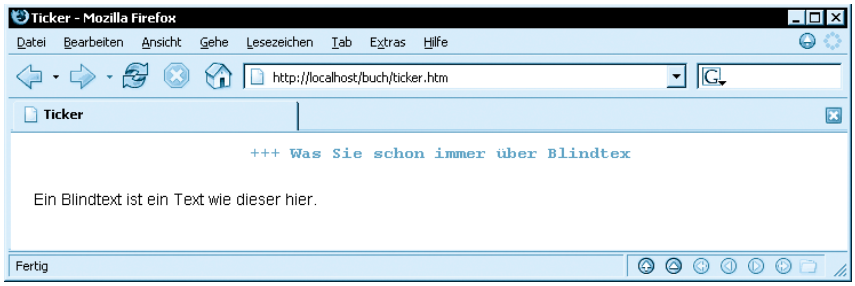


Abbildung 7.9: JavaScript-basierter Ticker in Aktion

Im body-Bereich des HTML-Dokuments wird ein div-Bereich mit id-Namen `ticker` definiert. Seine optische Erscheinung und Position wird im zentralen style-Bereich im Dateikopf festgelegt. Im einleitenden <body>-Tag wird über den Event-Handler `onload=`, also wenn das Dokument geladen ist, die JavaScript-Funktion `show_ticker()` aufgerufen. Ihr werden zwei Parameter übergeben. Der erste Parameter ist der gewünschte Lauftext, der zweite Parameter nennt die Anzahl Millisekunden von einem Bewegungszustand des Lauftextes zum nächsten. Je höher dieser Wert, desto langsamer der Lauftext, und je niedriger, desto schneller.

Die Funktion `show_ticker()` steht im script-Bereich, der in den HTML-Kopfdaten notiert ist.

Die Logik des Lauftextes ist Folgende: Es gibt zwei »Basiszustände«, nämlich einen Zustand, in dem gar nichts von dem Text zu sehen ist, und einen, in dem der Text vollständig zu sehen ist. Ist der Text nicht zu sehen, soll er von rechts her Zeichen für Zeichen aufgebaut werden, so lange, bis er vollständig zu sehen ist. Dann soll er nach links hinaus Zeichen für Zeichen wieder verschwinden. Diese beiden »Bewegungsarten« sind aus programmiertechnischer Sicht sehr verschieden. Im ersten Fall muss nämlich die anzuzeigende Zeichenkette führende Leerzeichen enthalten und hinten den Teil des schon sichtbaren Textes. Im zweiten Fall dagegen muss die anzuzeigende Zeichenkette erst den noch sichtbaren Text enthalten und hinten Leerzeichen. Die »Bewegungsart« wird in unserem Script in der Variablen `effect` gespeichert. Diese Variable kann zwei Werte annehmen: `rtl` und `ltn`. Bei `rtl` soll der Text von rechts hereinwandern und bei `ltn` nach links hinaus.

Da der Text von rechts hereintickern soll und zunächst gar nicht zu sehen ist, besteht die Aufgabe also zunächst darin, ihn zeichenweise von rechts her aufzubauen. Dazu wird `effect` beim Einlesen des Scripts auf `rtl` gesetzt. Die Funktion `show_ticker()` besteht hauptsächlich aus einer if-else-if-Verzweigung, die den Zustand von `effect` abfragt. In den if-Zweig gelangt der Interpreter, wenn der Text von rechts herein aufgebaut werden soll, und in den else-if-Zweig, wenn er nach links hinaus verschwinden soll.

In der Variablen `chars` wird die Anzahl der aktuell sichtbaren Zeichen des Tickertextes gespeichert. Beim Einlesen wird diese Variable mit 0 initialisiert. Wenn der Text sich von

rechts her zeichenweise aufbaut (`rtl`), nimmt die Zeichenzahl bei jedem Schritt zu. Deshalb wird `chars` im if-Zweig um 1 erhöht. Anschließend müssen die führenden Leerzeichen erzeugt werden, und zwar so viele, wie der Tickertext an Zeichen hat, abzüglich der Zeichen, die davon bereits angezeigt werden – ausgedrückt durch `text.length - chars`. Das Ergebnis wird einer Hilfsfunktion namens `fill()` als Parameter übergeben. Diese Funktion tut nichts anderes, als eine Zeichenkette zu erzeugen und mit so vielen erzwungenen Leerzeichen (` `) zu füllen wie im Parameter `n` übergeben wurde. Dazu wird die zu erzeugende Zeichenkette `empty_chars` in einer for-Schleife nach und nach um je ein Leerzeichen erweitert. Am Ende wird der Leerzeichen-String mit `return` zurückgegeben.

Die Funktion `show_ticker()` fängt den Rückgabewert von `fill()` auf, indem sie die den Aufruf der Funktion `fill()` der Variable `text_now` als Wert zuweist. In `text_now` stehen anschließend so viele Leerzeichen, wie von `fill()` erzeugt wurden. Fehlt noch, die anzuzeigenden Zeichen des Tickertextes an `text_now` anzuhängen. Dazu wird mit der substring()-Methode des Zeichenkettenobjekts die gewünschte Teilzeichenkette des Tickertextes extrahiert. `text.substring(0, chars)` bedeutet: »Ermittle aus dem Wert des Parameters `text` die Teilzeichenkette, die bei Position = (= erstes Zeichen) beginnt, und extrahiere so viele Zeichen, wie in `chars` gespeichert ist.«

Da `chars` immer wieder um 1 erhöht wird, muss am Ende geprüft werden, ob `chars` mittlerweile die Länge des gesamten Tickertextes erreicht hat. Ist das der Fall, wird `effect` auf `ltn` gesetzt. Der Interpreter gelangt ab dann in den if-else-Zweig, wo `text_now` umgekehrt gefüllt wird.

Am Ende der Funktion wird der aktuelle Inhalt von `text_now` mit der innerHTML-Eigenschaft dynamisch in das HTML-Element mit dem id-Namen `ticker` geschrieben. Zuletzt ruft sich die Funktion `show_ticker()` selbst wieder auf, und zwar mit den gleichen Parametern, die sie beim Erstaufruf übergeben bekommen hat. Da die Variablen `chars` und `effect` global, also außerhalb von Funktionen, deklariert wurden, bleibt ihr Wert erhalten und `show_ticker()` kann beim nächsten Durchlauf mit deren aktuellen Werten arbeiten.

7.3.11 Arrays in JavaScript

Arrays sind Speicherbereiche für Serienwerte, die einen gemeinsamen Variablennamen haben. Um einen Array zu erzeugen, müssen Sie eine Instanz des Array-Objekts in einem frei wählbaren Variablennamen erzeugen. Die einfachste Möglichkeit zeigt das folgende Beispiel:

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

Damit wird ein leerer Array erzeugt.

Ebenso ist es aber möglich, einen Array gleich mit gewünschten Werten zu initialisieren. Das sieht dann wie im folgenden Beispiel aus:

```
var month_names = new Array("Januar", "Februar", "März", "April", "Mai", "Juni",
"Juli", "August", "September", "Oktober", "November", "Dezember");
```

Die einzelnen Werte werden innerhalb der Klammern hinter `new Array` notiert. Zeichenkettenwerte werden wie üblich in hohe Anführungszeichen eingeschlossen, numerische Werte nicht. Die Einzelwerte werden durch Kommata voneinander getrennt.

So wie im zweiten Beispiel initialisiert, können Sie über Indexnummern auf die Einzelwerte zugreifen. Dabei wird bei 0 mit dem Zählen begonnen, d.h., das erste Array-Element hat den Index 0, das zweite den Index 1, das dritte den Index 2 usw. Die Indexnummern werden in eckigen Klammern notiert. Ein Beispiel:

```
var month = month_names[5];
```

Anschließend ist in `month` das sechste Element aus dem Array `month_names` gespeichert, also der Wert `Juni`.

Auch schreibend kann auf Array-Elemente zugegriffen werden, und zwar auch dann, wenn die Elemente bislang gar nicht existieren. Dadurch wird das entsprechende Element einfach automatisch angelegt. Im folgenden Beispiel nutzen wir dies, um eine Methode zum Füllen eines leeren Arrays zu zeigen:

```
month_names = new Array();
month_names[0] = "Januar";
month_names[1] = "Februar";
month_names[2] = "März";
month_names[3] = "April";
month_names[4] = "Mai";
month_names[5] = "Juni";
month_names[6] = "Juli";
month_names[7] = "August";
month_names[8] = "September";
month_names[9] = "Oktober";
month_names[10] = "November";
month_names[11] = "Dezember";
```

Durch Zuweisung eines Werts an ein über die Indexnummer bestimmtes Array-Element wird dieses Element geändert.

Nun ist es mitunter lästig, mit Indexnummern zu arbeiten. Als Alternative besteht die Möglichkeit so genannter ***assoziativer Arrays***. Dabei werden anstelle von Indexnummern Indexnamen vergeben:

```
month_names = new Array();
month_names["jan"] = "Januar";
month_names["feb"] = "Februar";
month_names["mar"] = "März";
month_names["apr"] = "April";
month_names["may"] = "Mai";
month_names["jun"] = "Juni";
month_names["jul"] = "Juli";
month_names["aug"] = "August";
month_names["sep"] = "September";
month_names["oct"] = "Oktober";
```

```
month_names["nov"] = "November";
month_names["dec"] = "Dezember";
```

Die gewünschten Indexnamen werden innerhalb der eckigen Klammern wie Variablennamen für die Einzelelemente vergeben. Im Beispiel lässt sich anschließend etwa mit `month_names["aug"]` auf den Wert `August` zugreifen.

Arrays können auch verschachtelt sein. Dazu wird einem Array-Element anstelle eines einzelnen Werts einfach wieder ein Array-Objekt zugewiesen. Beispiel:

```
table_cells = new Array();
for(i = 0; i < 10; i++)
   table_cells[i] = new Array();
```

Mit diesem Konstrukt wird der potenzielle Datenspeicher einer Tabelle angelegt. Die Tabelle besteht aus Zellen, jedoch nicht linear, sondern zweidimensional. Dazu werden im Beispiel in einer for-Schleife insgesamt zehn Array-Elemente von `table_cells` mit dem Wert »neuer Array« initialisiert. Anschließend kann über Notationen wie `table_cells[2][6]` etwa auf das siebte Serienelement des dritten übergeordneten Serienelements zugegriffen werden.

Eigenschaften des Array-Objekts

Die einzige Eigenschaft ist `length`. Diese Eigenschaft kann nicht geändert werden und enthält die Anzahl der Elemente im Array. Sehr praktisch ist diese Eigenschaft etwa, um gezielt das letzte Array-Element anzusprechen: `month_names[month_names.length-1]` greift beispielsweise auf das letzte Element im Array `month_names` zu.

Methoden des Array-Objekts

Die Methoden des Array-Objekts ermöglichen das bequeme Manipulieren von Arrays sowie attraktive Sonderfunktionen wie Sortierung oder Umwandlung in Zeichenketten. Methoden werden auf den Array-Variablennamen angewendet. Wenn ein Array also beispielsweise mit `month_names = new Array(...)` initialisiert wurde, ist eine Notation wie `month_names.pop()` möglich.

Methode	Bedeutung
`concat(array)`	Hängt `array` an den bestehenden Array an.
`join(str)`	Verwandelt den Array in eine Zeichenkette, wobei `str` ein Zeichen oder eine Zeichenfolge ist, die als Delimiter der Einzelwerte eingefügt wird. Üblich ist beispielsweise ein Komma.
`pop()`	Entfernt das letzte Element im Array. Gibt dessen Inhalt zurück.
`push(val,...)`	Fügt ein oder mehrere, durch Kommata getrennte Werte am Ende des Arrays hinzu.
`reverse()`	Kehrt die Elementreihenfolge im Array um.
`shift()`	Entfernt das erste Element aus dem Array. Die nachfolgenden Elemente rücken nach vorne. Gibt den Inhalt des gelöschten Elements zurück.

Tabelle 7.11: Methoden des Array-Objekts

Methode	Bedeutung
`slice(x,y)`	Extrahiert einen Teil des Arrays, und zwar die Elemente mit den Indexnummern x bis y-1 (0-index-basiert). Gibt die extrahierten Elemente als Array zurück.
`splice(x,y,val,...)`	Fügt Werte an gewünschter Stelle im Array ein. Überschreibt dabei gegebenenfalls existierende Werte. x ist die gewünschte Indexnummer (0-index-basiert), an der eingefügt werden soll. y ist die Anzahl einzufügender Elemente. Alle weiteren Parameter sind die einzufügenden Werte.
`sort()`	Sortiert die Array-Werte alphanumerisch.
`unshift(val,...)`	Fügt ein oder mehrere, durch Kommata getrennte Werte am Anfang des Arrays hinzu.

Tabelle 7.11: Methoden des Array-Objekts (Fortsetzung)

Anwendungsbeispiel: Sortierte Lottozahlenreihe mit Zusatzzahl

Das Beispiel erzeugt sieben unterschiedliche Zufallszahlen zwischen 1 und 49. Die ersten sechs davon werden numerisch sortiert. Die letzte Zahl gilt als Zusatzzahl. Die erzeugte Reihe plus Zusatzzahl wird am Ende ausgegeben.

So einfach die Aufgabenstellung zunächst klingt, so wenig trivial ist eine ordentliche Lösung. Das nachfolgende Beispiel zeigt eine Lösung. Dabei wird intensiv mit Arrays gearbeitet.

Listing 7.8: Numerisch sortierte Lottozahlenreihe inklusive Zusatzzahl mit JavaScript

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Lottozahlen</title>
<script type="text/javascript">
</script>
</head>
<body>
<script type="text/javascript">
function in_array(arr, val) {
   for(var i = 0; i < arr.length; i++)
      if(arr[i] === val)
          return(i);
   return(-1);
}

function numsort(a,b) {
   return a-b;
}

var lotto_series = new Array();
for(var i = 0; i < 7; i++) {
   var new_val = Math.round(Math.random() * 49);
```

```
    if(lotto_series.length > 1) {
       while(in_array(lotto_series, new_val) > -1 ||
             new_val == 0)
          new_val = Math.round(Math.random() * 49);
    }
    lotto_series.push(new_val);
}
var lotto_series_6 = lotto_series.slice(0, 6);
lotto_series_6.sort(numsort);

html = "<h1>Lottozahlen</h1>\n<p>";
for(i = 0; i < lotto_series_6.length; i++)
   html += lotto_series_6[i] + " ";
html += "<br>Zusatzzahl: " + lotto_series[6] + "</p>";
document.write(html);
</script>
</body>
</html>
```

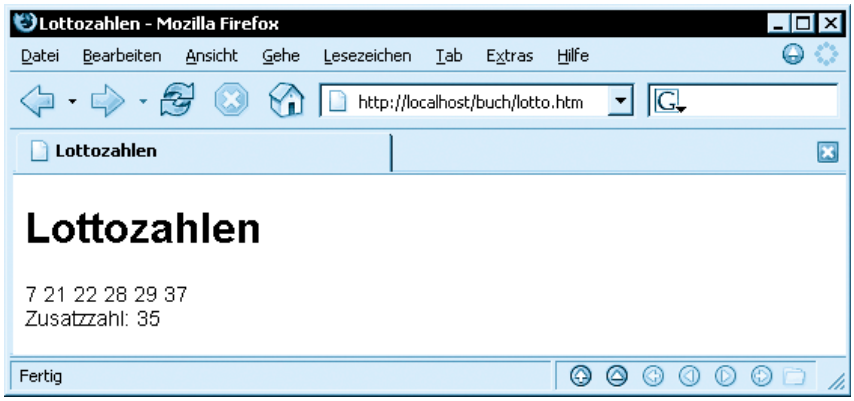


Abbildung 7.10: Numerisch sortierte Lottozahlenreihe inkl. Zusatzzahl mit JavaScript

Im body-Bereich des HTML-Dokuments ist ein script-Bereich notiert. Dieser enthält zunächst zwei Hilfsfunktionen. Die erste, `in_array()`, durchsucht einen Array in einer for-Schleife nach einem Wert. Wird der Wert gefunden, so wird die Indexnummer des gefundenen Elements zurückgegeben. Wird kein Element mit dem Wert gefunden, so wird -1 zurückgegeben. Als ersten Parameter erwartet `in_array()` den zu durchsuchenden Array und als zweiten Parameter den Wert, nach dem gesucht werden soll.

Die zweite Hilfsfunktion, `numsort()`, ist eine Hilfsfunktion, die wir zum Sortieren von numerischen Werten benötigen. Die Funktion gibt einfach das Ergebnis der Subtraktion des zweiten vom ersten übergebenen Parameter zurück. Das Ergebnis ist entweder positiv, 0 oder negativ. Diesen Input kann die sort-Methode des Array-Objekts verarbeiten.

Der Rest des Scripts steht außerhalb jeder Funktion und wird daher gleich beim Einlesen ausgeführt. Zunächst wird in `lotto_series` eine Instanz des Array-Objekts gespeichert, also ein neuer Array erzeugt.

Anschließend werden in einer for-Schleife insgesamt sieben Zufallszahlen erzeugt. Jede Zahl sollte eine Ganzzahl zwischen 1 und 49 sein. Sichergestellt wird dies durch das Berechnungskonstrukt `Math.round(Math.random() * 49)`. Allerdings kann es immer noch vorkommen, dass zwei gleiche Zahlen vorkommen. Um dies zu verhindern, muss, sobald bereits eine oder mehrere Zahlen im Array sind (`lotto_series.length > 1`), überprüft werden, ob die neu ermittelte Zufallszahl schon vorkommt. Dazu wird `in_array()` aufgerufen. Außerdem soll verhindert werden, dass die Zahl 0 als Zufallszahl ermittelt wird. Aus beiden Wünschen wird eine Bedingung für eine while-Schleife gebastelt. Innerhalb der while-Schleife wird nochmals ein neuer Zufallswert als Ganzzahl kleiner oder gleich 49 ermittelt. Das wird gegebenenfalls so oft wiederholt, wie eine der Bedingungen erfüllt ist (Wert kommt schon vor oder Wert ist 0).

Ist alles in Ordnung, dann wird dem Array `lotto_series` mit der push()-Methode der neue Zufallswert angehängt.

Die ersten sechs der sieben ermittelten Zahlen sollen die reguläre Lottoreihe bilden. Die siebte Zahl soll die Zusatzzahl sein. Die ersten sechs Zahlen sollen sortiert ausgegeben werden. Dazu werden zunächst mithilfe der slice()-Methode die ersten sechs der sieben Zahlen aus `lotto_series` extrahiert und in `lotto_series_6` gespeichert. Dieser Array wird anschließend mit der sort()-Methode sortiert – und zwar numerisch. Da die sort()-Methode normalerweise alphanumerisch sortiert, würden beispielsweise Folgen wie 15, 2, 25, 3, 31, 42 entstehen. Erst eine numerische Sortierung liefert wie gewünscht 2, 3, 15, 25, 31, 42. Damit die sort()-Methode numerisch sortiert, wird ihr der Funktionsname der Hilfsfunktion `numsort()` als Parameter übergeben. Nehmen Sie diese etwas ungewöhnliche Notation einfach als gegeben hin.

Zuletzt wird im Beispiel HTML-Code erzeugt, der die Zahlenreihe und die Zusatzzahl ausgibt.

7.3.12 Weitere Objekte, Eigenschaften und Methoden

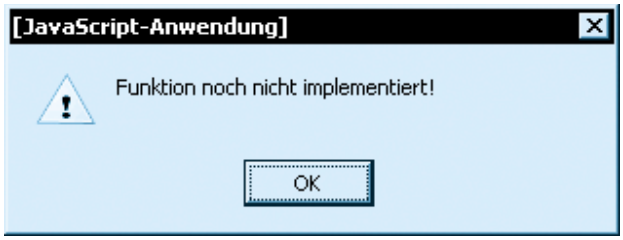
Wir haben nun wichtige Standardobjekte von JavaScript 1.5 behandelt. JavaScript bietet eine ganze Reihe weiterer Objekte einschließlich Eigenschaften und Methoden an. Viele davon kommen in der Praxis jedoch kaum zum Einsatz und werden deshalb auch im Rahmen dieses Buchs nicht behandelt. Zahlreiche andere Objekte, Methoden und Eigenschaften gehören heute entweder zum DOM oder wurden durch andere DOM-Entsprechungen ersetzt. Einige sind auch ganz unter den Tisch gefallen, weil sie weder zu JavaScript 1.5 noch zum DOM gehören, obwohl sie eigentlich nützlich sind. In diesem Abschnitt behandeln wir daher quer Beet verschiedene Möglichkeiten, die Sie kennen sollten. Einst beliebte, aber aussterbende Möglichkeiten wie das Öffnen von Popup-Fenstern, das Verändern von Fensterposition oder Fenstergröße oder das Manipulieren der Statusleiste stellen wir bewusst nicht vor, da die entsprechenden Features auf professionellen Sites nichts mehr zu suchen haben.

Benutzerinteraktion mit einfachen Dialogfenstern

Als JavaScript-Programmierer haben Sie die Möglichkeit, einfache, vordefinierte Dialogfenster beim Anwender anzuzeigen, die dieser bestätigen bzw. bearbeiten kann. Drei solche Fenstertypen gibt es:

- **Meldungsfenster:** gibt einen Text aus. Der Anwender kann nur mit einer Schaltfläche für OK bestätigen. Gedacht, um wichtige Hinweise einzublenden. Beispiel:

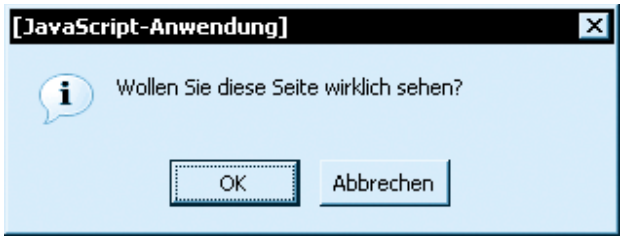
```
alert('Funktion noch nicht implementiert!');
```



Die Methode `alert()` bekommt den gewünschten Meldungstext übergeben. Eine Anweisung wie im Beispiel könnte etwa an einer Stelle im Script stehen, die zwar schon erreichbar ist (z.B. durch Eingaben in einem Formular), aber noch nicht die gewünschte Funktionalität bietet.

- **Bestätigungsfenster:** gibt einen Text aus, der als Rückfrage zum Auslösen einer Aktion gedacht ist. Der Anwender kann mit einer Schaltfläche für Ja oder Nein bestätigen. Das Script bekommt übergeben, wie sich der Anwender entschieden hat. Beispiel:

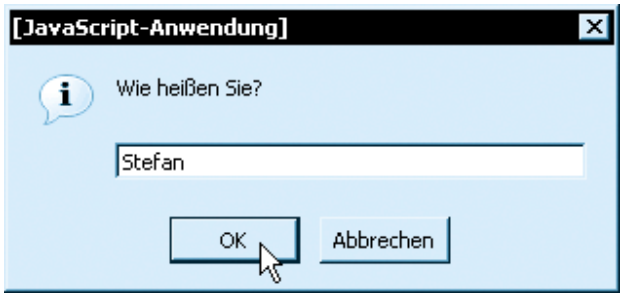
```
check = confirm("Wollen Sie diese Seite wirklich sehen?");
if(check == false) history.back();
else location.href = "gnadenlos.html"
```



Die Methode `confirm()` bekommt den gewünschten Text übergeben. Der Aufruf wird einer Variablen zugewiesen. Die Variable enthält anschließend den Wert true, wenn der Anwender positiv bestätigt, oder den Wert false, wenn er den Dialog negativ quittiert.

- **Eingabefenster:** bietet ein Eingabefeld mit einem voranstehenden Text. Der Anwender kann einen Wert eingeben. Gedacht, um einzelne Daten vom Anwender einzuholen, ohne dafür ein HTML-Formular verwenden zu müssen. Beispiel:

```
user_name = prompt("Wie heißen Sie?","");
document.write("<h1>Willkommen, " + user_name + "</h1>");
```



Die Methode `prompt()` erwartet zwei Parameter. Der erste ist der gewünschte Fragetext, der zweite die Vorbelegung des Eingabefelds. Wenn Sie keine Vorbelegung wünschen, übergeben Sie wie im Beispiel eine leere Zeichenkette. Der Aufruf wird einer Variablen zugewiesen. Die Variable enthält anschließend als Zeichenkette den Wert, den der Anwender eingegeben hat.

Das Aussehen der Dialogfenster ist system- und browserspezifisch und kann nicht geändert werden.

Browser-Grundfunktionen wie Back, Forward, Stop, Reload und Drucken

JavaScript bietet die Möglichkeit, diese Browser-Grundfunktionen auszuführen. So ist es beispielsweise möglich, die entsprechenden Funktionen über einen Hyperlink anzustoßen. Das nachfolgende Beispiel definiert solche Hyperlinks:

```
<p>
   <a href="javascript:history.back()">Seite zurück</a><br>
   <a href="javascript:history.forward()">Seite vor</a><br>
   <a href="javascript:stop()">Laden abbrechen</a><br>
   <a href="javascript:location.reload()">Neu laden mit neuem
      Hist.Eintrag</a><br>
   <a href="javascript:location.replace()">Neu laden</a><br>
   <a href="javascript:print()">Seite drucken</a>
</p>
```

Die Methoden `stop()` und `print()` gehören zum Fenster-Objekt von JavaScript die Methoden `reload()` und `replace()` zum location-Objekt und die Methoden `back()` und `forward()` zum history-Objekt von JavaScript vor Version 1.5. Browser interpretieren diese Objekte und Methoden noch. Wenn Sie Links wie im obigen Beispiel notieren, sollten Sie sich jedoch dessen bewusst sein, dass Anwender, die JavaScript deaktiviert haben, beim Anklicken solcher Links eine Fehlermeldung erhalten oder dass gar nichts passiert.

Wechsel zu anderer Seite

Über das location-Objekt von JavaScript ist es auch möglich, JavaScript einen Sprung zu einer anderen Seite ausführen zu lassen. In Zusammenhang mit den einfachen Dialogfenstern hatten wir bereits ein Beispiel dafür:

```
check = confirm("Wollen Sie diese Seite wirklich sehen?");
if(check == false) history.back();
else location.href = "gnadenlos.html"
```

Durch Zuweisung eines URI an `location.href` wird ein Sprung zum entsprechenden URI ausgeführt. Es kann sich um eine lokal referenzierte Quelle oder um eine absolute Internetadresse handeln. Durch den Sprung zur anderen Seite wird die aktuell angezeigte Seite verlassen. Das Script wird dadurch natürlich auch nicht weiter ausgeführt. Ein entsprechender Aufruf sollte also nur an geeigneten Stellen stehen.

Datenübergabe zwischen Seiten

Jedes JavaScript bezieht sich immer nur auf die aktuell angezeigte Seite. Variablen usw. gehen verloren, wenn eine neue Seite angezeigt wird. Eine Möglichkeit, Daten an eine nachfolgende Seite zu übergeben, besteht darin, in JavaScript einen GET-String zu basteln und diesen an Aufrufe der nächsten Seite in den URI einzubauen. Es existiert jedoch eine elegantere Methode. Im Fenster-Objekt von JavaScript gibt es eine Eigenschaft `name`, welche eigentlich den Namen eines Fensters speichert. Diese Eigenschaft kann gelesen und geändert werden. Sie hat den Vorteil, dass ihr Wert so lange stabil bleibt, wie das betreffende Fenster geöffnet ist. Da nun der Fenstername etwa des Browser-Hauptfensters völlig uninteressant ist, kann die Eigenschaft `window.name` hervorragend dazu genutzt werden, um Zeichenketten zu speichern. Das folgende Beispiel demonstriert eine Datenübergabe dieser Art:

Zunächst Seite 1:

Listing 7.9: Dokument mit Script für Datenübergabe

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Eingabe sichern</title>
<script type="text/javascript">
function save_input() {
  window.name = document.getElementById("text").value;
  return true;
}
</script>
</head>
<body>
<form action="js2.htm" onsubmit="save_input()"
```

```
      style="background-color:#FFDDAA; padding:10px;">
<p>Geben Sie Text ein:<br>
<input type="text" id="text" style="width:360px;
border:solid 1px gray"></p>
<p><button type="submit" style="width:120px;
border:solid 1px gray">Speichern</button></p>
</form>
</body>
</html>
```

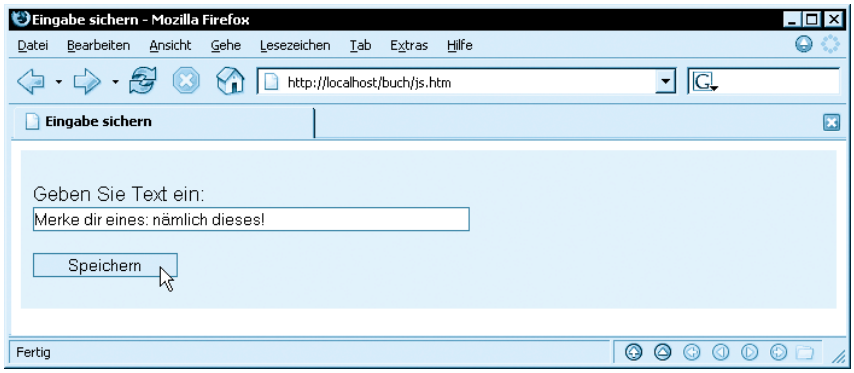


Abbildung 7.11: Formular für Eingabe eines zu speichernden Inhalts

Das Dokument enthält ein Formular mit einem Eingabefeld (id-Name `text`). Außerdem enthält das Formular einen Submit-Button mit der Aufschrift *Speichern*. Beim action-Attribut ist der URI einer Seite eingetragen, die aufgerufen werden soll, wenn der Anwender auf den Submit-Button klickt. Zuvor soll jedoch noch eine JavaScript-Funktion namens `save_input()` ausgeführt werden, die in einem script-Bereich im Dateikopf steht. Damit die Funktion noch ausgeführt wird, ist im einleitenden <form>-Tag der Event-Handler `onsubmit=` notiert. Dieses Ereignis tritt genau dann ein, wenn der Anwender das Formular absendet. Die Funktion `save_input()` liest den eingegebenen Wert aus dem Eingabefeld aus und speichert ihn in unserem zweckentfremdeten Datenspeicher `window.name`.

Die Anweisung `return true` ist notwendig, damit das Formular tatsächlich abgesendet und die bei `action=` angegebene Adresse aufgerufen wird.

Der Quelltext der zweiten Seite (*js2.htm*) lautet in unserem Beispiel:

Listing 7.10: Dokument mit Script für Datenauswertung

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Eingabe anzeigen</title>
</head>
<body>
```

```
<p>Der eingegebene Text lautet:<br>
<script type="text/javascript">
  document.write(window.name);
</script>
</body>
</html>
```

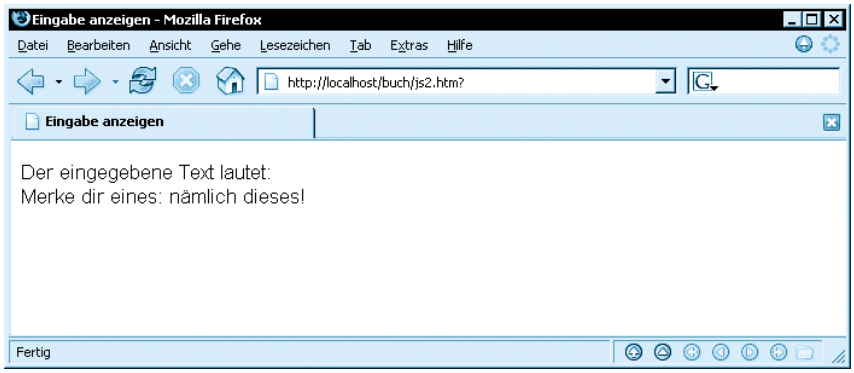


Abbildung 7.12: Ausgabe des in window.name gespeicherten Inhalts

Das Dokument liest in einem script-Bereich einfach den gespeicherten Wert von `window.name` aus und schreibt ihn ins Dokument.

Falls Sie beispielsweise Inhalte aus mehreren Formularfeldern sichern möchten, müssen Sie diese so zu einer einzigen Zeichenkette zusammenfügen, dass es im aufgerufenen Dokument wieder möglich ist, die Zeichenkette korrekt auseinander zu nehmen. Es bietet sich an, Einzelwerte in der zusammengesetzten Zeichenkette durch eine selbst definierte Trennzeichenfolge wie `"|+|"` zu trennen. Das aufgerufene Dokument kann dann mit einer Anweisung wie `var fields = window.name.split("|+|")` die einzelnen Werte in einem Array sammeln.

Seiten-Umgebungsinformationen

Über das document-Objekt von JavaScript lassen sich unter anderem wichtige Informationen zur aktuell angezeigten Seite ermitteln:

Eigenschaft	Bedeutung
`document.title`	Seitentitel. In neueren Browser auch änderbar.
`document.URL`	URI der Seite. Nur lesbar.
`document.charset`	Im Dokument verwendeter Zeichensatz oder Zeichenkodierung. Auch änderbar.
`document.lastModified`	Zeichenkette, die den letzten Änderungszeitpunkt an der HTML-Datei der Seite speichert. Bei dynamisch erzeugtem HTML-Code der aktuelle Zeitpunkt. Nur lesbar.

Tabelle 7.12: Seiteneigenschaften mittels document-Objekt

Eigenschaft	Bedeutung
`document.referrer`	URI der zuletzt besuchten Seite (wenn verfügbar). Nur lesbar.
`document.cookie`	Liest oder speichert Cookies zur Seite. Mit `x = document.cookie` wird der Cookie-String in x gespeichert. Mit `document.cookie = x` wird das Cookie gesetzt, wobei x nach den Konventionen von Cookies aufgebaut sein muss. Mit `if(document.cookie)` kann überprüft werden, ob Cookies gesetzt sind.

Tabelle 7.12: Seiteneigenschaften mittels document-Objekt (Fortsetzung)

Anwender-Umgebungsinformationen

Frühere JavaScript-Versionen erlaubten es, Umgebungsversionen wie Daten zum verwendeten Browser, Bildschirminformationen und installierte Plug-Ins abzurufen. Leider wurden diese Daten nur allzu oft dazu missbraucht, um dem Anwender etwas aufzuzwingen, wie etwa sein Browser-Fenster auf Bildschirmgröße zu trimmen, oder um wildes, proprietäres Zeug für bestimmte Browser zu programmieren. Die wichtigsten dieser Abfrageinformationen sollen hier vorgestellt werden, jedoch mit der Warnung, diese nicht zu missbrauchen.

Eigenschaft	Bedeutung
`screen.width`	Breite des Anwenderbildschirms in Pixel.
`screen.height`	Höhe des Anwenderbildschirms in Pixel.
`screen.availWidth`	Breite in Pixel, auf die ein Fenster maximal ausdehnbar ist.
`screen.availHeight`	Höhe in Pixel, auf die ein Fenster maximal ausdehnbar ist.
`screen.colorDepth`	Farbtiefe des Anwenderbildschirms in Bit.
`navigator.appName`	Name des vom Anwender verwendeten Browsers.
`navigator.appVersion`	Produktversion des vom Anwender verwendeten Browsers.
`navigator.language`	Oberflächensprache des vom Anwender verwendeten Browsers.
`navigator.platform`	Betriebssystem oder Betriebssystemgattung, auf der der Browser beim Anwender läuft.
`navigator.userAgent`	Zeichenkette, mit der sich der vom Anwender verwendete Browser bei der Kommunikation mit einem Webserver ausweist.

Tabelle 7.13: Eigenschaften für Anwender-Umgebungsinformationen

Vor allem Informationen des navigator-Objekts sind jedoch mit Vorsicht zu genießen. Einige Browser erlauben es, diese Daten zu konfigurieren, weshalb die Abfrage möglicherweise unbrauchbare und nicht klassifizierbare Ergebnisse liefert.

Ein sinnvolles Anwendungsbeispiel:

```
<img id="pic" alt="Bild">
<script type="text/javascript">
   if(Math.pow(2,screen.colorDepth) >= Math.pow(2,16))
      document.getElementById("pic").setAttribute("src",
      "wueste.jpg");
```

```
    else
       document.getElementById("pic").setAttribute("src",
       "wueste.gif");
</script>
```

Zunächst wird eine HTML-Grafikreferenz notiert, jedoch ohne src-Attribut. Das src-Attribut und damit die Entscheidung, welche Grafikdatei referenziert werden soll, bestimmt stattdessen JavaScript. Unterhalb der Grafikreferenz steht ein script-Bereich. Darin wird abgefragt, ob die Anzahl Farben am Anwenderbildschirm größer als oder gleich 65536 ist. Wenn ja, wird eine andere Grafikdatei angezeigt als im else-Fall.

Diverse objektunabhängige Funktionen

Neben den Default-Objekten von JavaScript gibt es einige wichtige Standalone-Funktionen. Nachfolgende Tabelle stellt die wichtigsten davon vor:

Funktion	Bedeutung
`decoceURI(uri)`	Wandelt URI-gerechte Sonderzeichen wie `%C3` in ihre normalen Zeichenwerte um.
`encodeURI(uri)`	Bereitet `uri` so auf, dass alle Sonderzeichen darin URI-gerecht kodiert werden.
`eval(expr)`	Bewertet einen beliebigen JavaScript-Ausdruck, der als `expr` zu übergeben ist. Wird beispielsweise eine Rechenoperation übergeben, liefert diese Funktion das Ergebnis. Wegen möglicher folgenreicher Fehlinterpretation von Ausdrücken und Manipulation durch Anwendereingaben ist die Funktion allerdings in Fachkreisen verpönt.
`isFinite(x)`	Ermittelt, ob x noch im Zahlenbereich ist, den JavaScript verarbeiten kann. Gibt `true` bzw. `false` zurück.
`isNaN(x)`	Ermittelt, ob x eine ungültige Zahl ist. Gibt `true` zurück, wenn x keine Zahl ist, und `false`, wenn es eine ist.
`parseInt(str)`	Wandelt die Zeichenkette `str` in eine Ganzzahl um. Praktisch, um Anwendereingaben aus Formularfeldern in numerische Werte umzuwandeln.
`parseFloat(str)`	Wandelt die Zeichenkette `str` in eine Fließkommazahl um. Praktisch, um Anwendereingaben aus Formularfeldern in numerische Werte umzuwandeln.
`String(obj)`	Wandelt ein in `obj` gespeichertes Objekt in eine Zeichenkette um.

Tabelle 7.14: Wichtige objektunabhängige Default-Funktionen in JavaScript

7.3.13 Existenz von Objekten, Eigenschaften und Methoden

Da die Umsetzung von JavaScript in den verschiedenen Browsern uneinheitlich ist, ist es mitunter sinnvoll, abzufragen, ob ein Objekt, eine Eigenschaft oder eine Methode verfügbar ist, bevor man diese im Code anspricht. So lassen sich Fehler bei der Ausführung des Scripts vermeiden. Da sich die Situation bei den modernen Browsern bereits gebessert hat, sind nicht mehr ganz so viele Abfrageorgien erforderlich, wie es früher einmal der Fall war. Dennoch gehört es nach wie vor zum sauberen Programmierstil, zumindest bei solchen Objekten, Eigenschaften und Methoden auf Existenz zu prüfen, die es noch nicht so lange gibt. Ein Beispiel:

```
function set_heading() {
   if(!document.getElementById)
      return;
   now = new Date;
   var text = now.getDate() + "." + (now.getMonth() + 1) + "." +
              now.getFullYear();
   document.getElementById("heading").innerHTML = text;
}
```

In dieser Funktion werden alle Browser, welche die typische und häufig benutzte DOM-Methode `document.getElementById` nicht kennen, mit einem einfachen `return` sofort wieder aus der Funktion verbannt. Nur Browser, welche die Methode kennen, führen den restlichen Code in der Funktion aus.

Um die Existenz eines Objekts, einer Eigenschaft oder einer Methode abzufragen, wird der entsprechende Name einfach in einer if-Abfrage als Bedingung eingesetzt (bei Methoden ohne die runden Klammern). Jedes Objekt, jede Eigenschaft und jede Methode erzeugt intern bei Bekanntheit ein `true`, und wenn unbekannt, ein `false`.

Auf die gleiche Weise sind etwa folgende Anweisungen denkbar:

```
if(Array) {
   var numbers = new Array(1,2,3,4,5);
   if(numbers)
      document.write(numbers.length);
   if(numbers.toString)
      document.write(numbers.toString());
}
```

Das Beispiel prüft zunächst, ob das Array-Objekt verfügbar ist, und legt abhängig davon eine Objektinstanz namens `numbers` an. Auch die Existenz von Objektinstanzen ist abfragbar. Das Beispiel fragt ab, ob `numbers` erfolgreich angelegt wurde, und gibt im Erfolgsfall die Anzahl der Elemente aus. Dann wird noch abgefragt, ob die allgemeine Objektmethode `toString()` für die Objektinstanz `numbers` verfügbar ist, welche den Objektinhalt in eine Zeichenkette verwandelt. Falls verfügbar, wird die Methode angewendet und ihre Zeichenkette ausgegeben.

7.3.14 Eigene Objekte, Eigenschaften und Methoden

JavaScript selbst präsentiert sich zwar weitgehend in Form von Objekten, Eigenschaften und Methoden, doch in der Praxis der JavaScript-Programmierung wird meist nicht streng objektorientiert programmiert, sondern eher strukturiert und funktionsorientiert im Sinne von klassischem C, oder auch einfach nur »Spaghetti-Code«. Das liegt sicherlich zum Teil daran, dass mit kaum einer Programmiersprache so viele unerfahrene Nicht-Programmierer in Berührung kommen wie mit JavaScript. Zum anderen begründet es sich aber auch dadurch, dass das objektorientierte Konzept von JavaScript 1.5 noch nicht wirklich konsequent und weitreichend genug ist. Erst JavaScript 2.0 soll »echtes« objektorientiertes Programmieren mit Features wie Vererbung, Kapselung usw. realisieren.

Dennoch ist es auch mit JavaScript 1.5 möglich, ansatzweise objektorientiert zu programmieren. Der Einsatz von Objekten empfiehlt sich vor allem dann, wenn komplexere Datenstrukturen angelegt werden sollen. An einem Beispiel soll die grundsätzliche Vorgehensweise erläutert werden. Das Beispiel enthält einen script-Bereich, der zwei Objekte mit Eigenschaften und einer Methode anlegt und den Zugriff darauf testet.

```
<script type="text/javascript">
function team() {
   this.name;
   this.city;
   this.founded;
   this.members = new Array();
   this.add_member = add_member;
}
function member() {
   this.gname;
   this.fname;
   this.birthday;
}
function add_member(member) {
   this.members.push(member);
}

var fire_lions = new team();
fire_lions.name = "FireLions";
fire_lions.city = "Feuerstadt";
fire_lions.founded = 1960;

var hs = new member;
hs.gname = "Hans";
hs.fname = "Sander";
hs.birthday = "21.03.1980";

document.write("Team-Name: " + fire_lions.name + "<br>");
document.write("Statd: " + fire_lions.city + "<br>");
document.write("Team gegründet: " + fire_lions.founded + "<br>");
document.write("Spieler: " + hs.gname + " " + hs.fname + "<br>");
document.write("Geburtstag: " + hs.birthday + "<br>");

fire_lions.add_member(hs);

document.write("Team-Mitglieder: "
    + fire_lions.members.length + "<br>");
document.write("Vorname: "
    + fire_lions.members[0].gname + "<br>");
document.write("Zuname: "
    + fire_lions.members[0].fname + "<br>");
</script>
```

Ein eigenes Objekt wird mithilfe einer gewöhnlichen Funktion erzeugt. Im Beispiel werden zwei Objekte erzeugt: `team` und `member`. Die Objektfunktionen `team()` und `member()`,

auch als Konstruktorfunktionen bezeichnet, unterscheiden sich nur innerhalb des Funktionskörpers von gewöhnlichen Funktionen.

Innerhalb der Objektfunktion werden Eigenschaften und Methoden des Objekts deklariert. Eine Initialisierung ist möglich, aber nicht nötig. Die Deklaration von Eigenschaften und Methoden wird durch das Schlüsselwort `this` eingeleitet, gefolgt durch den Punktoperator. Dahinter folgt der Name, unter dem die Eigenschaft oder Methode angesprochen werden kann.

Im Beispiel des Objekts `team` werden zunächst drei gewöhnliche Eigenschaften deklariert, dann ein Array und zuletzt eine Methode. Bei Methoden wird zur Deklaration die Syntax `this.Methodenname = Methode` benutzt (ohne runde Klammern!). Dabei ist Methodenname der Name, unter dem die Methode später aufrufbar ist, und Methode der Name einer Funktion, welche den JavaScript-Code dieser Methode enthält. `Methodenname` und `Methode` können gleich lauten, müssen es aber nicht. In unserem Beispiel lauten sie gleich. Mit `this.add_member = add_member` wird eine Methode `add_member` deklariert. Die zugehörige Funktion `add_member()` ist etwas weiter unten im Script notiert.

Beim zweiten Beispielobjekt `member` werden lediglich drei Objekteigenschaften deklariert.

Um mit eigenen Objekten zu arbeiten, müssen Instanzen davon erzeugt werden. Dazu wird die Objektfunktion mit dem new-Operator aufgerufen. Im Beispiel wird mit `var fire_lions = new team()` eine Instanz des Objekts `team` in der Objektvariablen `fire_lions` gespeichert.

Über die Objektvariable ist anschließend der lesende und schreibende Zugriff auf Eigenschaften des Objekts nach dem Schema `Objektvariable.Eigenschaft` möglich, also über den Punktoperator.

Die Anweisung `fire_lions.add_member(hs)` in unserem Beispiel weiter unten demonstriert einen Zugriff auf eine Objektmethode. Auch diese verwendet den Punktoperator. Im Beispiel fügt die Methode `add_member()` das als Parameter übergebene Objekt vom Typ `member` in den Array `members` des team-Objekts hinzu. Der Array enthält also keine Zahlen oder Zeichenketten als Elemente, sondern ganze Objekte. Der Zugriff auf Einzeldaten daraus ist aber sehr wohl möglich, wie etwa `fire_lions.members[0].gname` zeigt.

Wie aus dem Beispiel deutlich wird, ist das Anlegen eigener Objekte vor allem hinsichtlich einer ordentlichen Datenstrukturierung zweckmäßig, ansonsten jedoch von keinem besonderen Nutzen.

7.4 Zugriff auf Seiteninhalte (DOM)

Das eigentlich Spannende an JavaScript ist, Seiteninhalte nachträglich dynamisch zu ändern. Denn dadurch eröffnen sich zahlreiche Möglichkeiten:

- In Verbindung mit Mausereignissen können bestimmte Inhalte ein- und ausgeblendet werden, z. B. Zusatzinformationen oder als direkte Reaktion auf eine vom Anwender getroffene Auswahl.

- Inhalte können sich automatisch bewegen, z. B. als automatisch scrollende News.
- Inhalte können anders angeordnet werden, z. B. lassen sich Tabellen sortieren.
- Navigationseffekte wie explorerartige Menüs sind möglich.

Dank der durch das DOM vereinheitlichten Zugriffssyntax auf einzelne Bestandteile im HTML-Quelltext ist es keine Hexerei mehr, etwas Derartiges zu programmieren. Lange Zeit war das anders, als browser-spezifische, völlig inkompatible DOM-Konzepte nebeneinander existierten. Diese Zeiten sind mittlerweile jedoch so weit in die Vergangenheit gerückt, dass man die alten Modelle getrost vergessen darf. Da Webseiten grundsätzlich auch bei deaktiviertem JavaScript einwandfrei funktionieren sollten, ist es problemlos möglich, einfach alle Browser, deren JavaScript-Interpreter noch keine W3-gerechte DOM-Syntax beherrschen, von der weiteren Ausführung eines Scripts auszuschließen. Die Anzahl solcher Browser dürfte mittlerweile längst geringer sein als die Anzahl derer, bei denen ihre Anwender JavaScript generell deaktiviert haben.

Der vorliegende Abschnitt enthält keine systematische Beschreibung des DOM, sondern geht gezielt auf diejenigen Bestandteile ein, die für die JavaScript-Praxis mit HTML von Bedeutung sind.

7.4.1 Zugriff auf Elemente und Elementinhalte

Das DOM begreift das gesamte Markup eines XML- oder HTML-Dokuments als Hierarchie aus *nodes* (zu Deutsch: *Knoten*). Das Ausgangsobjekt für den Zugriff auf einzelne Bestandteile des Markup und des Inhalts ist das document-Objekt.

Das document-Objekt stellt unter anderem eine ganze Reihe wichtiger Methoden zur Verfügung, um schnell und gezielt auf einzelne Elemente im Dokument zugreifen zu können. Dazu müssen Sie die drei folgenden Methoden kennen:

- `document.getElementById()` erlaubt den direkten Zugriff auf ein Element, das mit dem Attribut `id=` im einleitenden Tag einen dokumentweit eindeutigen Namen erhalten hat.
- `document.getElementsByName()` erlaubt den direkten Zugriff auf Elemente, die mit dem Attribut `name=` im einleitenden Tag einen Namen erhalten haben. Da solche Namen auch mehrfach vorkommen können, beispielsweise bei zusammengehörigen Elementen des Typs `<input type="radio" name="gemeinsamer_Name">`, werden gefundene Elemente in jedem Fall in einem Array gespeichert, auch wenn nur ein Element gefunden wird.
- `document.getElementsByTagName()` erlaubt den direkten Zugriff auf Elemente auf Grund ihres Elementnamens, also `body`, `h1`, `blockquote`, `td` usw. Da solche Namen ebenfalls mehrfach vorkommen können, werden gefundene Elemente auch in diesem Fall in einem Array gespeichert, selbst wenn nur ein Element gefunden wird.

Bei allen drei Methoden werden die ermittelten Elemente als Objekt gespeichert, und zwar als Objekt des Typs »Knoten«. Die drei Methoden erlauben also den schnellen Zugriff auf bestimmte Elemente. Ohne Aufruf von Objekteigenschaften, Objektmethoden

oder Unterobjekten des ermittelten Objekts ist jedoch kein Zugriff auf konkrete Daten oder Inhalte des Elements möglich.

Ein Beispiel soll den Umgang mit den genannten Methoden des document-Objekts demonstrieren. Dabei werden wir auch einige Eigenschaften des Knotenobjekts kennen lernen.

Listing 7.11: HTML-Dokument mit Script und DOM-Zugriffen

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>JavaScript</title>
</head>
<body>
<div id="text" style="border:3px gray solid; padding:10px;">
<h1>Blindtext</h1>
<p>Lorem ipsum dolor sit amet, ...</p>
<p>Duis autem vel eum iriure dolor ...</p>
<p>Nam liber tempor cum soluta nobis ...</p>
<p>Duis autem vel eum iriure dolor in hendrerit ...</p>
<p>At vero eos et accusam et justo ...</p>
<p>Consetetur sadipscing elitr ...</p>
</div>
<h1>Statistik</h1>
<p>Anzahl Absätze im Blindtext: <strong></strong></p>
<script type="text/javascript">
   var blind_text = document.getElementById('text');
   var elem_count = 0;
   for(i = 0; i < blind_text.childNodes.length; i++)
     if(blind_text.childNodes[i].nodeType == 1 &&
        blind_text.childNodes[i].nodeName.toLowerCase() == 'p')
           elem_count += 1;
   document.getElementsByTagName('strong')[0].innerHTML
        = elem_count;
</script>
</body>
</html>
```

Das HTML-Dokument besteht innerhalb des body-Elements zunächst aus einem div-Bereich mit dem id-Namen `text`. Es ist der Bereich, der in der Abbildung grau umrandet erscheint. Der div-Bereich enthält eine h1-Überschrift sowie eine Reihe von Textabsätzen (p-Elemente). Unterhalb des umrandeten div-Bereichs ist eine h1-Überschrift »Statistik« notiert und darunter ein Textabsatz, der am Ende eine leere `<strong></strong>`-Markierung enthält.

Unterhalb all dieser Elemente ist ein script-Bereich notiert. Der script-Bereich befindet sich an dieser Stelle, weil sein Code nicht innerhalb von Funktionen steht, also gleich beim

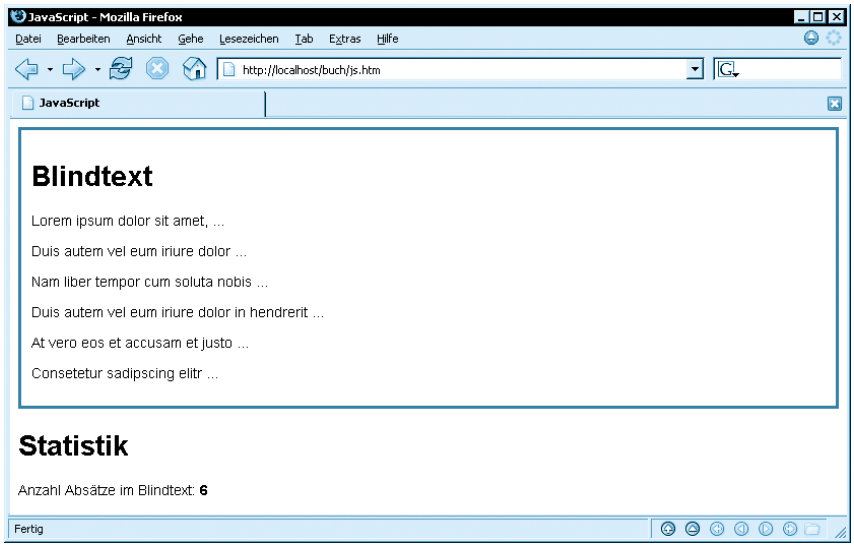


Abbildung 7.13: Angezeigtes Ergebnis der DOM-Zugriffe im Browser

Laden ausgeführt wird. Da sich seine Anweisungen aber auf HTML-Elemente des Dokuments beziehen, müssen diese dem Browser zum Zeitpunkt der Scriptausführung schon bekannt sein.

Die erste Anweisung im Script lautet:

```
var blind_text = document.getElementById('text');
```

Durch `document.getElementById('text')` wird auf das Element mit dem id-Namen `text` zugegriffen. Im Beispieldokument ist dies das div-Element mit dem grauen Rahmen. Ermittelt wird, wie bereits erwähnt, das Knotenobjekt des Elements. Durch die Zuweisung an die Variable `blind_text` wird diese zu einem Speicher des entsprechenden Element-Knotenobjekts. Die Anweisung hat lediglich den Zweck, mit `blind_text` eine verkürzte Schreibweise für `document.getElementById('text')` zu definieren.

Als Nächstes wird eine Variable namens `elem_count` mit 0 initialisiert. In dieser Variable soll die Anzahl der p-Elemente im div-Bereich `text` gespeichert werden, also die Blindtextabsätze im grau umrahmten Bereich ohne die Überschrift.

Um die Anzahl der p-Elemente unterhalb des div-Elements `text` zu zählen, werden in einer for-Schleife alle Kindknoten dieses Elements ermittelt. Das Objekt des Elements `text` ist ja in der Variablen `blind_text` gespeichert. Um auf Kindknoten eines Knotenobjekts zugreifen zu können, gibt es das Unterobjekt `childNodes`. In `blind_text.childNodes` sind also alle Kindknoten des div-Elements `text` gespeichert, und zwar als Array aus Knotenobjekten. Da es sich um einen Array handelt, ist die Eigenschaft `length` anwendbar. In der for-Schleife können wir diese Eigenschaft für die Formulierung der Durchlaufbedingung gut gebrauchen. So können wir mithilfe der for-Schleife alle Kindknoten des Elements der Reihe nach durchgehen und prüfen.

Geprüft soll werden, ob es sich beim jeweils aktuellen Kindknoten um einen Knoten des Typs »Element« handelt. Dazu gibt es die Objekteigenschaft `nodeType`. Diese Eigenschaft speichert in Form einer Nummer, um welche Art von Knoten es sich handelt. Nachfolgende Tabelle listet die möglichen Nummern auf:

nodeType=	Knotentyp
1	Element
2	Attribut
3	Textinhalt
4	CDATA-Bereich (XML)
5	Entity-Referenz (XML)
6	Entity (XML)
7	Verarbeitungsanweisung (XML)
8	Kommentar
9	Dokument
10	Dokumenttyp
11	Dokumentfragment (XML)
12	Notation (XML)

Tabelle 7.15: Werte der Objekteigenschaft nodeType

In unserem Beispiel werden alle Kindelemente gesucht. Deshalb wird in der Bedingung gefragt, ob `nodeType == 1` ist. Außerdem sollen nur p-Elemente gefunden werden. Deshalb wird mit logischem `Und` (`&&`) eine zweite Bedingung aufgestellt, die erfüllt ist, wenn `nodeName.toLowerCase() == 'p'`. Dabei ist `nodeName` wieder eine Objekteigenschaft des Knotenobjekts. Wenn es sich um einen Knoten des Typs Element handelt, wird darin der Elementname als Zeichenkette gespeichert. Da JavaScript zwischen Groß- und Kleinschreibung unterscheidet, wenden wir die Methode `toLowerCase()` auf diese Zeichenkette an und können sichergehen, dass der Vergleich klein geschriebene Elementnamen erzwingt.

Nur wenn beide Bedingungen der if-Abfrage (Knotentyp `Element` und Element namens `p`) erfüllt sind, wird die abhängige Anweisung ausgeführt. Diese besteht darin, den Wert der Variablen `elem_count` um 1 zu erhöhen.

Nach Beendigung der for-Schleife steht in `elem_count` die Anzahl der p-Elemente, die sich innerhalb des div-Elements `text` befinden. Das Ergebnis wird nun noch dynamisch ins Dokument eingefügt, und zwar als Elementinhalt des strong-Elements. Da dieses keinen id-Namen hat, bleibt nur, es über `document.getElementsByTagName('strong')` zu ermitteln. Die Methode liefert einen Array aller strong-Elemente. Da es nur ein strong-Element im Dokument gibt, kann mit `document.getElementsByTagName('strong')[0]` darauf zugegriffen werden.

Als Objekteigenschaft wird diesmal `innerHTML` verwendet. Diese bereits bekannte »Schlamper-Eigenschaft« ermöglicht es sehr bequem, HTML-Code in ein Element einzu-

fügen. Durch Zuweisung von `elem_count` an `innerHTML` wird der Wert von `elem_count` als Inhalt in das strong-Element eingefügt.

7.4.2 Elemente erzeugen und integrieren

Über das DOM ist es nicht nur möglich, auf vorhandene Bestandteile des Markup eines Dokuments zuzugreifen, sondern auch, neue Bestandteile zu erzeugen und in die Dokumentstruktur zu integrieren. Das funktioniert zwar auch mit der innerHTML-Eigenschaft, doch in diesem Abschnitt zeigen wir, welche Methoden das DOM für entsprechende Zwecke favorisiert.

Das nachfolgende Beispiel zeigt, wie eine Seite Anwendereingaben in den Seiteninhalt integrieren kann.

Listing 7.12: Einfügen von Seiteninhalten auf Grund von Anwendereingaben

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>JavaScript</title>
<script type="text/javascript">
function add_female_name() {
   var name = document.getElementById("female_name").value;
   document.getElementById("female_name").value = "";
   var li_elem = document.createElement("li");
   var li_text = document.createTextNode(name);
   li_elem.appendChild(li_text);
   document.getElementById("female_names").appendChild(li_elem);
}
function add_male_name() {
   var name = document.getElementById("male_name").value;
   document.getElementById("male_name").value = "";
   var li_elem = document.createElement("li");
   var li_text = document.createTextNode(name);
   li_elem.appendChild(li_text);
   document.getElementById("male_names").appendChild(li_elem);   }
</script>
</head>
<body>
<form action="javascript:void(0)"
   style="background-color:#FFDDAA; padding:10px;
   position:absolute; top:10px; left:10px;">
<p>Notiere deine weiblichen Lieblingsnamen:<br>
<input type="text" id="female_name" style="width:150px;
border:solid 1px gray">
<button type="button" style="border:solid 1px gray"
onclick="add_female_name()">OK</button></p>
<p>Notiere deine männlichen Lieblingsnamen:<br>
```

```
<input type="text" id="male_name" style="width:150px;
border:solid 1px gray">
<button type="button" style="border:solid 1px gray"
onclick="add_male_name()">OK</button></p>
</form>
<div style="position:absolute; top:10px; left:350px;">
<h1 style="font-size:120%">Weibliche Lieblingsnamen</h1>
<ul id="female_names">
</ul>
<h1 style="font-size:120%">Männliche Lieblingsnamen</h1>
<ul id="male_names">
</ul>
</div>
</body>
</html>
```

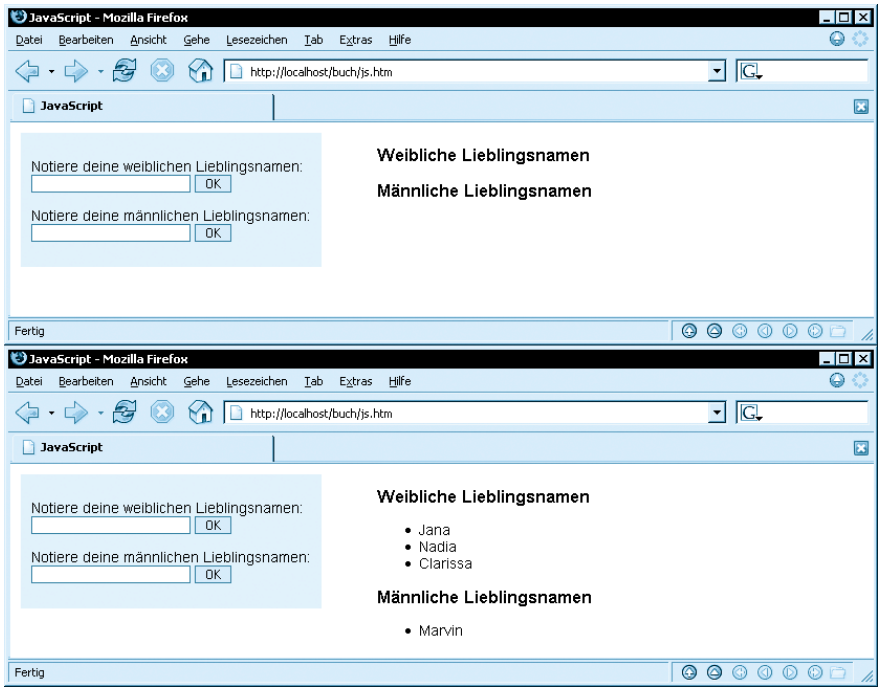


Abbildung 7.14: Seiteninhalt bei Anfangszustand und Zustand nach Anwendereingaben

Das Beispieldokument enthält ein Formular mit zwei Eingabefeldern und zwei zugehörigen Buttons mit der Aufschrift *OK*. Bei Anklicken eines OK-Buttons wird der im Feld links davon eingegebene Name rechts vom Formular in eine Aufzählungsliste übernommen und das Eingabefeld wird für die nächste Eingabe geleert. Das Beispiel zeigt, wie eine Seite Anwenderdaten ohne komplizierte Formulare sammeln kann und wie sie dem Anwender außerdem noch ein attraktives Feedback seiner Eingaben anzeigt.

In den einleitenden `<button>`-Tags der OK-Buttons ist je ein Event-Handler `onclick=` notiert. Als Wert wird jeweils ein Funktionsaufruf zugewiesen. Dadurch wird erreicht,

dass beim Anklicken des einen Buttons die Funktion `add_female_name()` und beim Anklicken des anderen die Funktion `add_male_name()` aufgerufen wird. Erstere besorgt das Verarbeiten von eingegebenen weiblichen Vornamen und Letztere das von männlichen Vornamen.

Die beiden Funktionen stehen im Kopfbereich des HTML-Dokuments innerhalb eines script-Bereichs. Beide haben den gleichen Aufbau und unterscheiden sich im Code nur durch Details.

Zunächst wird der vor dem Button-Klick eingegebene Name ermittelt. Dazu wird im Fall der weiblichen Namen auf `document.getElementById("female_name").value` zugegriffen. Dieses Konstrukt bedarf einer genaueren Erläuterung.

In `document.getElementById("female_name")` ist, wie wir mittlerweile wissen, ein Objekt gespeichert, nämlich das Knotenobjekt des input-Elements, in dem der Anwender weibliche Vornamen eingeben kann. Durch einfaches Anhängen des Attributnamens `value` kann auf den Wert des Attributs `value=` zugegriffen werden. Diese Zugriffssyntax ist speziell auf HTML ausgerichtet und gilt für alle Attribute. Mehr dazu in ***Abschnitt 7.4.3.***

In unserem Beispiel wird der eingegebene Formularfeldwert in der Variablen `name` gespeichert. Anschließend wird das Formularfeld geleert, denn der Anwender soll ja weitere Namen eingeben können. Dazu wird dem value-Attribut des entsprechenden Eingabefelds ein Leerstring `""` zugewiesen. Der Browser entfernt in diesem Moment der Script-Ausführung alle eingegebenen Zeichen im entsprechenden Feld.

Dem Script ist nun der eingegebene Name bekannt. Als Nächstes kann es den Namen in die Aufzählungsliste rechts einfügen. Dazu werden die entsprechenden DOM-Methoden verwendet. Zunächst wird mit `document.createElement("li")` ein neues Element mit Elementnamen `li` erzeugt. Das erzeugte Objekt wird im Beispiel in einer Objektinstanzvariablen namens `li_elem` gespeichert.

Die Methode `createElement()` ist wie beispielsweise `getElementById()` eine Methode des Dokumentobjekts im DOM. Und ebenso, wie in `getElementById()` ein Objekt gespeichert wird, so ist dies auch bei `createElement()` der Fall. Die Methode tut nichts weiter, als ein leeres Knotenobjekt des Typs »Element« mit einem bestimmten Elementnamen zu erzeugen. Das neue Objekt befindet sich zu diesem Zeitpunkt noch nicht innerhalb der Dokumentstruktur, sondern »freischwebend« im Datenspeicher.

Nach dem gleichen Schema, wie das li-Element erzeugt wurde, entsteht als Nächstes ein Textknotenobjekt. Ein solches Objekt kann eine Zeichenkette (genauer: CDATA) speichern, jedoch kein HTML-Markup enthalten. Zum Erzeugen von Textknoten wird die Methode `createTextNode()` verwendet. Die Methode bekommt als Parameter den Text übergeben, der in dem Textknotenobjekt gespeichert werden soll. In unserem Beispiel übergeben wir die Variable `name`, in der ja der im Namensfeld eingegebene Wert gespeichert ist. In der Objektinstanzvariablen `li_text` ist also anschließend der eingegebene Vorname gespeichert.

Nachdem die Knotenobjekte für das li-Element und seinen Textinhalt erzeugt wurden, kann es daran gehen, die Objekte zusammenzubauen und in die Dokumentstruktur an

der gewünschten Stelle einzuhängen. Alles das ist mithilfe der Objektmethode `appendChild()` möglich. Mit der Anweisung `li_elem.appendChild(li_text)` wird der erzeugte Textknoten als Kindelement an das erzeugte li-Element angehängt. Damit ist in `li_elem` eine Markup-Struktur wie `<li>Vorname</li>` realisiert, die aber immer noch nicht ins Dokument eingebunden ist. Letzteres geschieht erst mit der folgenden Anweisung:

```
document.getElementById("female_names").appendChild(li_elem);
```

Mit Ausführen dieser Anweisung wird der eingegebene Name als neuer Aufzählungspunkt in der Liste rechts sichtbar.

Die appendChild()-Methode lässt sich also auf alle Knotenobjekte anwenden, die Kindknoten haben können, unabhängig davon, ob sich das Knotenobjekt in der Dokumentstruktur befindet oder mit `document.create...()` als neues, leeres Objekt erzeugt wurde. Die Notationsform lautet immer `Objekt.appendChild(Kindobjekt)`.

7.4.3 Zugriff auf Attribute und Werte

Um auf Werte von HTML-Attributen zuzugreifen, bietet das DOM eine vereinfachte Syntax an. Es genügt, das zugehörige Element über eine der drei Basismethoden genau zu bestimmen und dann den Attributnamen wie eine Objekteigenschaft anzuhängen.

Angenommen, im HTML-Quelltext kommt folgende Passage vor:

```
<p id="summary" lang="de">In diesem Artikel wird beschrieben, wie ...</p>
```

Dann kann der Wert des lang-Attributs dieses Elements wie folgt ermittelt werden:

```
document.getElementElementById("summary").lang
```

Wird einem solchen Konstrukt ein Wert zugewiesen (steht es also links in einer Zuweisung), dann wird der Attributwert geändert. Steht das Konstrukt rechts in einer Zuweisung, wird der Attributwert in der links stehenden Variablen gespeichert.

Nachfolgende Anweisung speichert die Sprache des p-Elements `summary` in einer Variablen namens `summary_language`:

```
summary_language =
   document.getElementElementById("summary").lang;
```

Angenommen, im HTML-Quelltext steht:

```
<table id="bundesliga" border="1"> ... </table>
```

In diesem Fall können Sie ganz einfach den Tabellenrahmen verschwinden lassen:

```
document.getElementElementById("bundesliga").border = "0";
```

Oder angenommen, im Quelltext steht:

```
<a id="mylink" href="http://localhost/">Dummy-Link</a>
```

In diesem Fall können Sie das Linkziel so ändern:

```
document.getElementElementById("mylink").href = "http://www.example.org/";
```

Ein paar Ausnahmen sind jedoch zu beachten, da einige Attribute in JavaScript nicht mit ihrem normalen HTML-Attributnamen notiert werden dürfen. Die nachfolgende Tabelle listet Attribute mit unterschiedlichen Schreibweisen auf.

HTML-Attribut	Schreibweise in JavaScript/DOM
accept-charset	acceptCharset
accesskey	accessKey
charoff	chOff
class	className
codebase	codeBase
codetype	codeType
colspan	colSpan
datetime	dateTime
for	htmlFor
frameborder	frameBorder
http-equiv	httpEquiv
longdesc	longDesc
maxlength	maxLength
noresize	noResize
noshade	noShade
readonly	readOnly
rowspan	rowSpan
standby	standBy
tabindex	tabIndex
usemap	useMap

Tabelle 7.16: Unterschiedliche Attributschreibweisen in HTML und JavaScript/DOM

Ein typisches Beispiel zum Ändern von Attributwerten, das mächtige Möglichkeiten eröffnet, sind die so genannten Style-Switcher. Dabei kann der Anwender mittels Formular oder Link ein Stylesheet aussuchen. Dieses wird dem Dokument dann zugewiesen. Als Seitenanbieter können Sie den Anwender somit selbst auswählen lassen, welche »Skin« er wünscht. Das könnte beispielsweise so aussehen:

Listing 7.13: Style-Switcher mit JavaScript

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>JavaScript</title>
<link id="style" type="text/css" rel="stylesheet"
      href="style-1.css">
<script type="text/javascript">
function change_style(n) {
  if(n == 1) {
     document.getElementById('style').href = "style-1.css";
     document.getElementById('heading').innerHTML = "Style 1";
 return;
  }
  if(n == 2) {
     document.getElementById('style').href = "style-2.css";
     document.getElementById('heading').innerHTML = "Style 2";
 return;
  }
}
</script>
</head>
<body>
<h1 id="heading">Style 1</h1>
<p>Wählen Sie:
<a href="javascript:change_style(1)">Style 1</a><br>
<a href="javascript:change_style(2)">Style 2</a>
</p>
</body>
</html>
```

Das Beispieldokument enthält im body-Bereich zwei Links, die bei Anklicken beide die Funktion `change_style()` aufrufen. Diese ist im Dateikopf in einem script-Bereich notiert. Übergeben wird ihr beim Aufruf eine Zahl. Die Funktion wertet die Zahl, die sie als Parameter `n` erwartet, aus. Beträgt der Wert 1, gelangt die Ausführung in den ersten if-Zweig. Beträgt er zwei, wird der zweite if-Zweig ausgeführt. Die jeweils erste Anweisung greift auf das link-Element zu, das ebenfalls im Dateikopf notiert ist. Es hat ein id-Attribut mit dem Wert `style` und kann daher über `document.getElementById('style')` angesprochen werden. Geändert wird das href-Attribut, indem eine andere externe CSS-Datei zugewiesen wird. Moderne Browser reagieren umgehend, lesen die andere CSS-Datei aus und ändern dynamisch im angezeigten Dokument alles, was auf Grund der anderen CSS-Angaben zu ändern ist.

Ferner tauscht die Funktion im Beispiel noch den Inhalt der h1-Überschrift aus, damit der Anwender sieht, welches Stylesheet aktuell ist.

Neben dieser vereinfachten Zugriffssyntax auf HTML-Attribute gibt es aber auch »echte« DOM-Methoden für den Zugriff auf Attribute. Ein paar Beispiele:

```
language = document.getElementElementById("summary").getAttribute('lang');
```

In dieser Anweisung wird in `language` der Wert des lang-Attributs eines p-Elements mit dem id-Namen `summary` gespeichert. Die DOM-Methode `getAttribute()` kann an eine der drei Basismethoden zum Zugriff auf ein bestimmtes Element angehängt werden. Sie ermittelt den Wert eines Attributs. Dazu bekommt sie den gewünschten Attributnamen als Parameter übergeben.

```
document.getElementElementById("mylink").setAttribute("href", "http://
www.example.org/");
```

In dieser Anweisung wird das Verweisziel des a-Elements mit dem id-Namen `mylink` geändert. Die DOM-Methode `setAttribute()` erwartet als ersten Parameter den Namen des gewünschten Attributs und als zweiten Parameter den gewünschten Wert. Ist das Attribut im HTML-Quelltext bereits vorhanden, wird sein alter Wert durch den neuen ersetzt. Ist es noch nicht vorhanden, wird es neu angelegt und mit dem neuen Wert belegt.

Analog zu den beiden DOM-Methoden `document.createElement()` und `document.createTextNode()`, die wir bereits kennen gelernt haben, gibt es auch eine Methode `document.createAttribute()`. Dazu ein Beispiel:

```
<blockquote id="kuehe">In der Nacht sind alle Kühe schwarz, auch die blonden</
blockquote>
<script type="text/javascript">
   var author = document.createAttribute("cite");
   author.nodeValue = "Karl Kraus";
   document.getElementById("Kuehe").setAttributeNode(author);
</script>
```

Das Beispiel fügt mittels Script einem blockquote-Element ein Attribut `cite=` hinzu. Dazu wird zunächst mit `document.createAttribute()` ein neues Knotenobjekt des Knotenyps »Attribut« erzeugt. Als Parameter wird der Name des Attributs übergeben – im Beispiel `cite`. Das neu erzeugte Objekt wird in einer Objektinstanzvariablen gespeichert, wie im Beispiel `author`. Zu diesem Zeitpunkt hat das Attribut noch keinen Wert. Dieser wird ihm mithilfe der Objekteigenschaft `nodeValue` zugewiesen. Im Beispiel existiert nach Ausführung der Anweisung `author.nodeValue = "Karl Kraus"` ein Markup-Konstrukt `cite="Karl Kraus"`. Dieses ist jedoch noch keinem bestimmten Element zugewiesen, d.h., es muss noch in die Dokumentstruktur eingefügt werden. Dazu dient die letzte Anweisung im Beispielscript. Mit `document.getElementById("Kuehe")` wird das gewünschte Element angesprochen. Die Methode `setAttributeNode()` fügt ein Knotenobjekt des Knotentyps »Attribut« in ein Knotenobjekt des Knotentyps »Element« ein. Als Parameter wird der Methode die Variable übergeben, in der das Attributobjekt gespeichert ist.

7.4.4 Zugriff auf CSS-Eigenschaften

Der Zugriff auf CSS-Eigenschaften von HTML-Elementen ist ähnlich einfach wie derjenige auf Attributwerte. Voraussetzung ist auch hierbei wieder, mithilfe einer der drei Basismethoden `document.getElementById()`, `document.getElementsByName()` oder `document.getElementsByTagName()` ein Element zu bestimmen, dessen style-Eigenschaften geändert werden sollen. Ein Beispiel:

Listing 7.14: Dokument mit scriptgesteuertem Fading (langsamem Einblenden des Inhalts)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Fading</title>
<style type="text/css">
body {
   background-color:rgb(128,128,128);
   color:rgb(128,128,128);
}
</style>
<script type="text/javascript">
var bg_value = 128;
var fg_value = 128;
function fade() {
  if(bg_value < 0 || fg_value > 255)
     return;
  document.getElementById('body').style.backgroundColor =
  "rgb(" + bg_value + ","  + bg_value + ","  + bg_value + ")";
  document.getElementById('body').style.color =
  "rgb(" + fg_value + ","  + fg_value + ","  + fg_value + ")";
  bg_value += 1;
  fg_value -= 1;
  window.setTimeout("fade()",30);
}
</script>
</head>
<body id="body" onload="window.setTimeout('fade()',1000)">
<h1>H1-Überschrift</h1>
<p>Textabsatz</p>
</body>
</html>
```

Der body-Bereich des Beispieldokuments besteht aus einer h1-Überschrift und einem Textabsatz. Im Dateikopf ist ein style-Bereich notiert, in dem das body-Element mit CSS eine Hintergrundfarbe und eine Textfarbe zugewiesen bekommt. Beide Farben sind gleich (dunkles Grau). Wegen der Vererbung von CSS erhalten alle Textelemente innerhalb des body-Bereichs diese Farben. Zunächst ist daher gar kein Text zu sehen, da dieser Grau auf Grau erscheint.

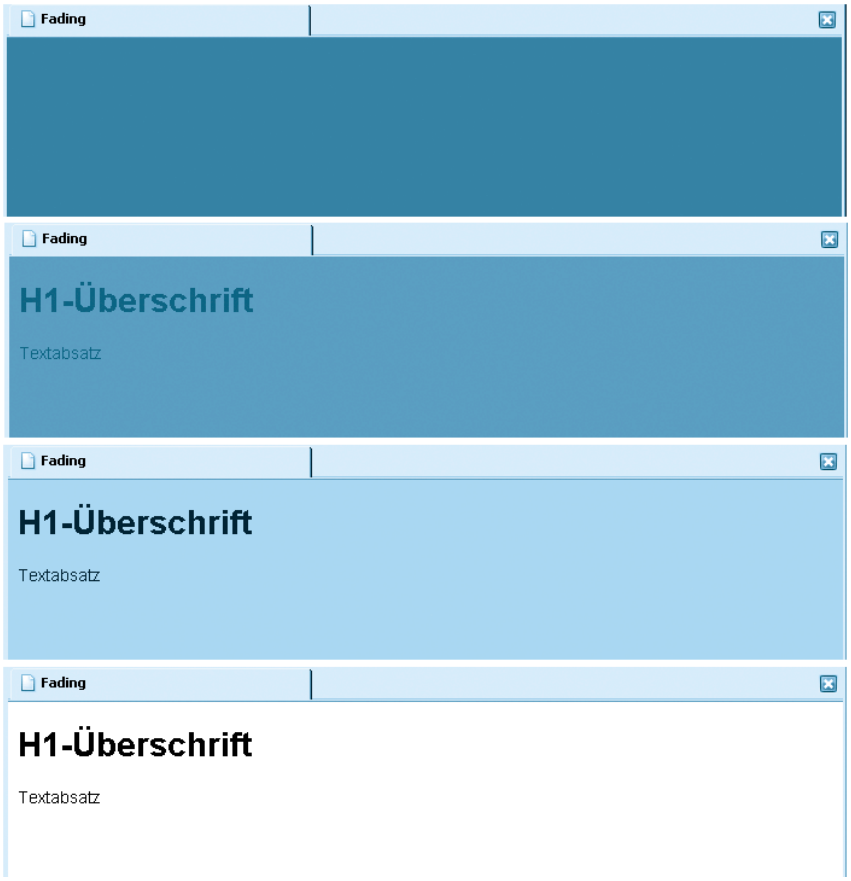


Abbildung 7.15: Dokument mit script-gesteuertem Fading (langsamem Einblenden des Inhalts)

Im einleitenden <body>-Tag wird jedoch über den Event-Handler `onload=` zeitverzögert um 1000 Millisekunden (eine Sekunde) die Funktion `fade()` aufgerufen, die in einem script-Bereich im Dateikopf steht. In dem script-Bereich werden gleich beim Einlesen zwei Variablen `bg_value` (RGB-Einheitswert für Hintergrundfarbe) und `fg_value` (RGB-Einheitswert für Textfarbe) deklariert und mit dem gleichen Wert initialisiert, der auch in den CSS-Formatdefinitionen benutzt wurde.

Die Funktion `fade()` ändert die CSS-Eigenschaftswerte. Dazu greift sie über die Basismethode `document.getElementById()` auf das body-Element zu, das den id-Namen `body` erhalten hat. Für den Zugriff auf CSS-Eigenschaften eines so ermittelten Elements wird einfach `style` wie eine Objekteigenschaft angehängt. Dahinter folgt, abermals durch den Punktoperator getrennt, der Name der CSS-Eigenschaft.

Bei den Namen von CSS-Eigenschaften in JavaScript ist jedoch zu beachten, dass dabei nicht der übliche Bindestrich benutzt werden darf. Stattdessen wird der Bindestrich weggelassen und das nachfolgende Teilwort beginnt mit einem Großbuchstaben. Aus der CSS-Eigenschaft `background-color` wird daher in JavaScript `backgroundColor`. Nach dem glei-

chen Schema würde beispielsweise aus `margin-left` in JavaScript `marginLeft` und `border-top-width` würde zu `borderTopWidth`.

Wertzuweisungen an CSS-Eigenschaften werden in JavaScript als Zeichenkette notiert, also z.B. `marginLeft = "20px"`. In unserem Fading-Beispiel werden jeweils Farbwerte in der Notationsform `rgb(R,G,B)` zugewiesen. Die als Wert zugewiesene Zeichenkette wird dabei aus den aktuellen Werten der Variablen `bg_value` bzw. `fg_value` zusammengesetzt. Anschließend wird `bg_value` um 1 erhöht und `fg_value` um 1 verringert. Schließlich ruft sich die Funktion `fade()` mit einer Zeitverzögerung von 30 Millisekunden selbst wieder auf.

Nach insgesamt 128 Durchläufen, die 128 × 30, also 3840 Millisekunden Nettozeit brauchen (3,8 Sekunden), ist die Textfarbe Schwarz, weil `fg_value` bei 0 angelangt ist, und die Hintergrundfarbe ist Weiß, weil `bg_value` 255 erreicht hat. Damit sich die Funktion `fade()` nicht endlos weiter aufruft und die beiden Variablen auf unzulässige Werte ändert, sorgt eine if-Abfrage am Anfang der Funktion dafür, dass die Funktion sofort verlassen wird, wenn die Grenzwerte erreicht sind.

Für die Praxis ist das vorgestellte Fading-Beispiel jedoch nur bedingt zu empfehlen, denn bei deaktiviertem JavaScript bleibt die Seite Grau auf Grau und der Anwender bekommt nichts vom Text zu sehen. Insofern kann das Beispiel auch als schlechtes Beispiel dienen und das Bewusstsein dafür schärfen, dass manches, was mit JavaScript schick aussieht, ohne JavaScript zu Anzeigeproblemen führt. Natürlich sind auch kreative Lösungen gefragt. In unserem Beispiel wäre es etwa möglich, im style-Bereich die Farben Schwarz auf Weiß zu definieren und stattdessen im script-Bereich außerhalb der Funktion, also gleich beim Einlesen, die Farben Grau auf Grau zuzuweisen. Auf diese Weise würden Browser, die das Script nicht interpretieren, die Seite so anzeigen wie nach dem Fading der Fall.

8 Praxisfälle für JavaScript/DOM

In Kapitel 7 wurden zwar schon diverse für die Praxis nützliche Scripts vorgestellt, doch die Beispiele hatten dort unterstützenden Charakter, um bestimmte Sprachkonzepte zu erläutern. In diesem Kapitel sollen die Beispiele im Vordergrund stehen. Das Kapitel zeigt, wie praxistypische Scripts entstehen und was dabei berücksichtigt werden sollte.

Wir konzentrieren uns auf drei wirklich nützliche Anwendungen, die JavaScript auch für professionelle Seiten salonfähig macht. Das Kapitel hat Workshop-Charakter.

Link zu JavaScript-Sammlungen:

Michael Mailer's JavaScripts:
http://www.mywebaid.de/
Scriptindex – JavaScript:
http://www.scriptindex.de/scripts/8/javascript/kat.php
JavaScript Source:
http://javascript.internet.com/toc.html

8.1 Clientseitige Formularüberprüfung

Bei Webanwendungen mit serverseitiger Datenanbindung erfolgt die Interaktion mit dem Anwender in der Regel über Formulare in HTML. Durch seine Angaben im Formular löst der Anwender nach Absenden des Formulars auf dem Webserver eine Datenverarbeitung aus. So wird beispielsweise ein Suchvorgang gestartet oder eingegebene Daten werden in eine Datenbank geschrieben oder in Dateiform gespeichert.

Auch wenn ein Formular so gestaltet ist, dass ein Anwender weiß, wie er es korrekt und sinnvoll ausfüllt, so kommt es doch immer wieder vor, dass Anwender bewusst Unsinn in ein Formular schreiben und es absenden oder dass sie ein wichtiges Feld vergessen auszufüllen oder – was am schlimmsten ist – dass sie versuchen, durch gefährliche Eingaben dem Server Schaden zuzufügen.

Bevor Anwendereingaben verarbeitet werden, sollten diese daher stets überprüft werden: Sind die Angaben vollständig bzw. sind alle wichtigen Felder ausgefüllt? Enthalten sie offensichtlichen Unsinn wie z. B. numerisch nicht interpretierbare Werte in einem Feld, in dem eine Zahl eingegeben werden soll? Enthalten sie Werte, die aus Server-Sicht problematisch sein können? (Wenn z. B. 1 Mio. Datensätze durchsucht werden und alle gefunde-

nen Sätze in einem Rutsch ausgegeben werden, ist es problematisch, wenn ein Anwender nur nach dem Buchstaben »a« sucht.)

Nun gibt es zwei Orte, an denen eine Überprüfung von eingegebenen Formulardaten stattfinden kann: Der eine Ort ist das Script, welches auf dem Server die Formulardaten empfängt und die Daten verarbeitet. Der Vorteil bei dieser Lösung ist, dass sie browserunabhängig ist und in jedem Fall funktioniert. Der Nachteil ist, dass zusätzliche Client-Server-Kommunikation zwischen Browser und Server nötig ist, falls Unzulänglichkeiten bei den Formulardaten festgestellt werden. Das Handling für den Anwender ist einfach umständlicher.

Der andere mögliche Ort einer Formularüberprüfung ist ein JavaScript, welches zusammen mit dem Formular an den Browser übertragen wird. Das Script wird dann aufgerufen, wenn der Anwender das Formular absenden will. Sind alle Eingaben in Ordnung, werden die Daten an den Server gesendet. Stellt das Script Fehler fest, kann der Anwender diese gleich korrigieren. Der Vorteil dieser Lösung ist also, dass bei Unzulänglichkeiten keine zusätzliche Client-Server-Konfiguration erforderlich ist und dass die Rechenkapazitäten des Anwenders mit der Eingabeprüfung belastet werden, statt die des Servers. Der Nachteil ist, dass bei deaktiviertem JavaScript keine Überprüfung stattfindet und die Daten ungeprüft zum Server gelangen.

In der Praxis ist daher eine doppelte Prüfung am realistischsten, wobei sich die serverseitige Prüfung auf problematische Fälle beschränken kann, während die clientseitige Prüfung auch den Charakter eines Hilfe-Assistenten annehmen kann.

8.1.1 Das Beispiel

In unserem Beispiel zur Formularüberprüfung mit JavaScript soll es um die Bestellung eines Horoskops gehen. Mit der Anmeldung geht der Anwender einen Kaufvertrag ein. Diverse Angaben sind zur Abwicklung Pflicht und einige davon müssen einem bestimmten Datenmuster entsprechen. Ein Script soll die Vollständigkeit und Korrektheit der Daten prüfen, bevor das Formular zum Server gesendet wird.

8.1.2 Quelltexte und Erläuterungen

Zunächst der Quelltext des HTML-Dokuments. Der script-Bereich zur Formularüberprüfung ist darin bereits vorgesehen, doch der Script-Code ist noch nicht eingefügt. Den JavaScript-Code werden wir noch separat behandeln. Im Quelltext des HTML-Dokuments sollen zunächst die Felder vorgestellt werden, die später mittels JavaScript auf Korrektheit überprüft werden sollen.

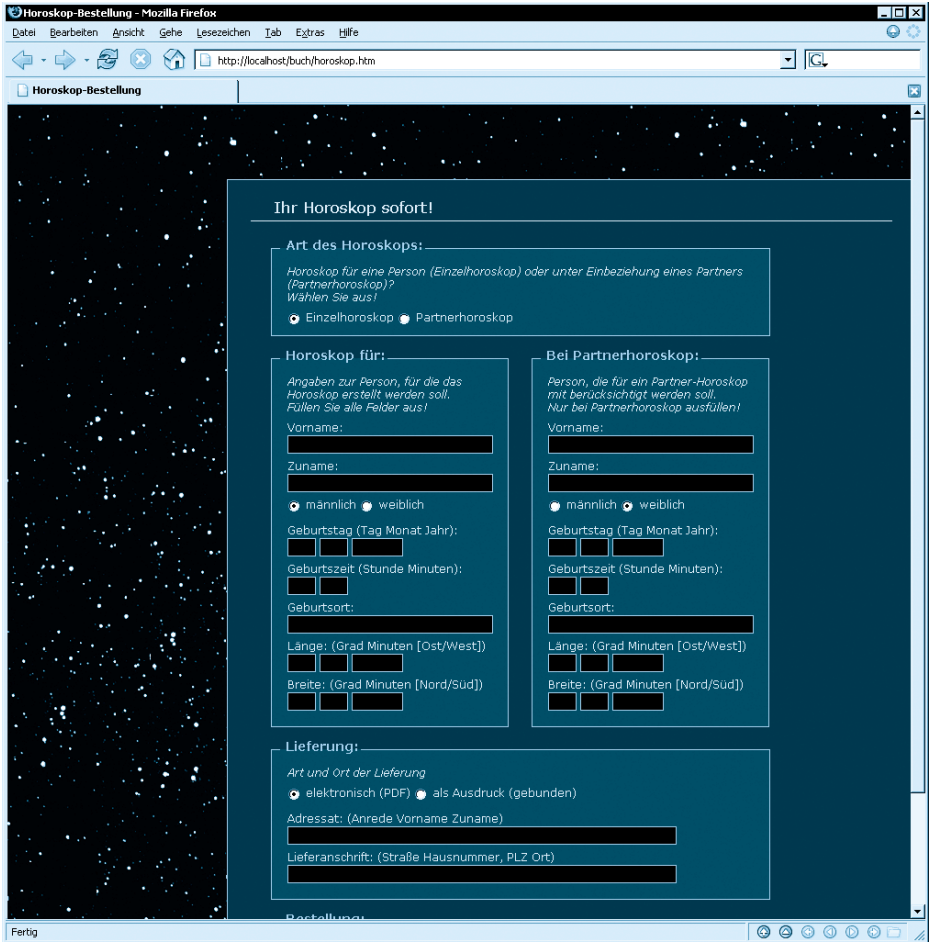


Abbildung 8.1: Formular zur Bestellung eines Horoskops

Listing 8.1: HTML-Quelltext mit Formular

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Horoskop-Bestellung</title>
<link rel="stylesheet" type="text/css" href="horoskop.css">
<script type="text/javascript">
function check_form() {
   // hier folgt die Formularüberprüfung
}
</script>
</head>
<body>
```

```
<div id="top">
</div>
<div id="left">
</div>
<div id="content">
<h1>Ihr Horoskop sofort!</h1>
<form id="order" action="/scripts/order.php" method="post"
      onsubmit="check_form()">
<table id="form_table"><tr><td colspan="2">
<fieldset id="type"><legend>Art des Horoskops:</legend>
<div class="explain">Horoskop für eine Person (Einzelhoroskop)
oder unter Einbeziehung eines Partners (Partnerhoroskop)?<br>
Wählen Sie aus!</div>
<div style="margin-bottom:12px">
<input type="radio" class="radio" name="horoscope_type"
       value="e" checked>
Einzelhoroskop
<input type="radio" class="radio" name="horoscope_type"
       value="p">
Partnerhoroskop
</div>
</fieldset>
</td>
</tr><tr>
<td><fieldset id="person"><legend>Horoskop für:</legend>
<div class="explain">Angaben zur Person, für die das Horoskop
erstellt werden soll.<br>Füllen Sie alle Felder aus!</div>
<div>Vorname:<br>
<input type="text" name="given_name" class="text"
       style="width:218px">
</div>
<div>Zuname:<br>
<input type="text" name="family_name" class="text"
       style="width:218px">
</div>
<div style="margin-bottom:12px">
<input type="radio" class="radio" name="gender"
       value="m" checked>
männlich
<input type="radio" class="radio" name="gender"
       value="w">
weiblich
</div>
<div>Geburtstag (Tag Monat Jahr):<br>
<input type="text" name="birthday_day" class="text"
       style="width:25px">
<input type="text" name="birthday_month" class="text"
       style="width:25px">
<input type="text" name="birthday_year" class="text"
       style="width:50px">
</div>
<div>Geburtszeit (Stunde Minuten):<br>
<input type="text" name="birthtime_hours" class="text"
```

```
       style="width:25px">
<input type="text" name="birthtime_minutes" class="text"
       style="width:25px">
</div>
<div>Geburtsort:<br>
<input type="text" name="birth_place" class="text"
       style="width:218px">
</div>
<div>Länge: (Grad Minuten [Ost/West])<br>
<input type="text" name="long_deg" class="text"
       style="width:25px">
<input type="text" name="long_min" class="text"
       style="width:25px">
<input type="text" name="long_dir" class="text"
       style="width:50px">
</div>
<div>Breite: (Grad Minuten [Nord/Süd])<br>
<input type="text" name="lat_deg" class="text"
       style="width:25px">
<input type="text" name="lat_min" class="text"
       style="width:25px">
<input type="text" name="lat_dir" class="text"
       style="width:50px">
</div>
</fieldset></td>
<td>
<fieldset id="partner"><legend>Bei Partnerhoroskop:</legend>
<div class="explain">Person, die für ein Partner-Horoskop mit
berücksichtigt werden soll.<br>
Nur bei Partnerhoroskop ausfüllen!</div>
<div>Vorname:<br>
<input type="text" name="p_given_name" class="text"
       style="width:218px">
</div>
<div>Zuname:<br>
<input type="text" name="p_family_name" class="text"
       style="width:218px">
</div>
<div style="margin-bottom:12px">
<input type="radio" class="radio" name="p_gender"
       value="m">
männlich
<input type="radio" class="radio" name="p__gender"
       value="w" checked>
weiblich<br>
</div>
<div>Geburtstag (Tag Monat Jahr):<br>
<input type="text" name="p_birthday_day" class="text"
       style="width:25px">
<input type="text" name="p_birthday_month" class="text"
       style="width:25px">
<input type="text" name="p_birthday_year" class="text"
       style="width:50px">
```

```
</div>
<div>Geburtszeit (Stunde Minuten):<br>
<input type="text" name="p_birthtime_hours" class="text"
       style="width:25px">
<input type="text" name="p_birthtime_minutes" class="text"
       style="width:25px">
</div>
<div>Geburtsort:<br>
<input type="text" name="p_birth_place" class="text"
       style="width:218px">
</div>
<div>Länge: (Grad Minuten [Ost/West])<br>
<input type="text" name="p_long_deg" class="text"
       style="width:25px">
<input type="text" name="p_long_min" class="text"
       style="width:25px">
<input type="text" name="p_long_dir" class="text"
       style="width:50px">
</div>
<div>Breite: (Grad Minuten [Nord/Süd])<br>
<input type="text" name="p_lat_deg" class="text"
       style="width:25px">
<input type="text" name="p_lat_min" class="text"
       style="width:25px">
<input type="text" name="p_lat_dir" class="text"
       style="width:50px">
</div>
</fieldset></td>
</tr><tr>
<td colspan="2">
<fieldset id="delivery"><legend>Lieferung:</legend>
<div class="explain">Art und Ort der Lieferung
</div>
<div style="margin-bottom:12px">
<input type="radio" class="radio" name="delivery_type"
       value="e" checked>
elektronisch (PDF)
<input type="radio" class="radio" name="delivery_type"
       value="p">
als Ausdruck (gebunden)<br>
</div>
<div>Adressat: (Anrede Vorname Zuname)<br>
<input type="text" name="delivery_person" class="text"
       style="width:418px">
</div>
<div>Lieferanschrift: (Straße Hausnummer, PLZ Ort)<br>
<input type="text" name="delivery_address" class="text"
       style="width:418px">
</div>
</fieldset></td>
</tr><tr>
<td colspan="2">
<fieldset id="submit"><legend>Bestellung:</legend>
```

```
<p><strong>Preise:</strong><br>
Einzelhoroskop elektronisch als PDF: &euro; 28,00<br>
Einzelhoroskop gedruckt und gebunden: &euro; 38,00<br>
Partnerhoroskop elektronisch als PDF: &euro; 36,00<br>
Partnerhoroskop gedruckt und gebunden: &euro; 46,00</p>
<div>
<input type="submit" class="button" value="Absenden">
</div>
</fieldset></td>
</tr></table>
</form>
</div>
</body>
</html>
```

Den Quelltext der CSS-Datei sparen wir an dieser Stelle aus, da er nicht von Belang ist.

Das HTML-Formular hat einen praxistypischen Umfang. Bei den Eingabefeldern soll Folgendes überprüft werden:

- Alle Feldgruppen mit Radiobuttons müssen jeweils eine gültige Auswahl enthalten.
- Wenn als Art des Horoskops *Einzelhoroskop* ausgewählt wurde, müssen alle Felder im Bereich *Horoskop für* einen Inhalt aufweisen. Wenn als Horoskopart *Partnerhoroskop* ausgewählt wurde, müssen sowohl im Bereich *Horoskop für* als auch im Bereich *Bei Partnerhoroskop* alle Felder ausgefüllt sein.
- Eingaben in Feldern für Vor- und Zuname sollen nicht mehr als 60 Zeichen haben dürfen, weil sie z.B. in eine Datenbank mit so begrenzten Feldlängen übernommen werden sollen.
- In den Einzelfeldern für Geburtstag und Geburtszeit müssen gültige Werte eingetragen werden: Monatstage zwischen 1 und 31, Monate zwischen 1 und 12, Jahr vierstellig, Stunden zwischen 0 und 23 und Minuten zwischen 0 und 59. Ferner soll geprüft werden, ob die Angaben zum Geburtsdatum ein gültiges Kalenderdatum ergeben.
- In den Einzelfeldern für Längengrad, Längenminuten, Breitengrad und Breitenminuten müssen ebenfalls gültige Werte eingetragen werden, also Längengrade zwischen 0 und 180, Breitengrade zwischen 0 und 90, Längen- und Breitenminuten zwischen 0 und 59 und in den Feldern für die Richtung ein Wort wie Ost oder West bzw. Nord oder Süd.
- Die Felder für den Geburtsort dürfen nicht mehr als 100 Zeichen haben. Die Felder für Adressat und Lieferanschrift dürfen nicht mehr als 160 Zeichen haben.

Bei Fehlern soll ein Meldungsfenster das aufgetretene Problem ausgeben. Im Fall von Eingabefeldern soll anschließend der Cursor direkt in das Feld mit der fehlenden oder fehlerhaften Eingabe positioniert werden, damit der Anwender den Wert gleich korrigieren kann.

Da diverse Checks öfter vorkommen, z.B. die Überprüfung eines Werts daraufhin, ob er sich in einem bestimmten numerischen Bereich von/bis bewegt, strukturieren wir den

JavaScript-Code so, dass einzelne, wiederverwendbare Funktionen bestimmte Prüfungen vornehmen und lediglich zurückgeben, ob die Eingabe den Prüfungskriterien entspricht (`true`) oder nicht (`false`).

Zum Management der einzelnen Fehler spendieren wir ebenfalls eine eigene Funktion. Eine Zentralfunktion schließlich geht die einzelnen Felder durch.

Zunächst zu den wiederverwendbaren Einzelfunktionen, die bestimmte Arten von Prüfungen vornehmen oder eine Hilfsfunktionalität leisten:

Listing 8.2: Wiederverwendbare Einzelfunktionen für Formularüberprüfung

```
function check_radios(name) {
   var check = false;
   for(i = 0; i < document.getElementsByName(name).length; i++) {
      if(document.getElementsByName(name)[i].checked == true)
         check = true;
   }
   return(check);
}

function check_field_length(name, length) {
   var check = false;
   if(document.getElementsByName(name)[0].value.length <= length)
      check = true;
   return(check);
}

function check_field_min_length(name, min_length) {
   var check = false;
   if(document.getElementsByName(name)[0].value.length >= min_length)
      check = true;
   return(check);
}

function check_field_str_range(name, comma_str) {
   var check = false;
   var strings = comma_str.split(",");
   for(i = 0; i < strings.length; i++) {
      if(document.getElementsByName(name)[0].value.toLowerCase() ==
strings[i].toLowerCase())
         check = true;
   }
   return(check);
}

function check_field_int_range(name, min, max) {
   var check = false;
   var x = parseOint(document.getElementsByName(name)[0].value);
   if(x >= min && x <= max)
      check = true;
   return(check);
}
```

```
function check_valid_date(day, month, year) {
   if(isNaN(day) || isNaN(month) || isNaN(year))
      return false;
   if(year < 1801 || year > 2799)
      return false;
   if(month < 1 || month > 12)
      return false;
   var feb = 28;
   if(month == 2) {
      if(year % 4 == 0) feb += 1;
      if(year % 100 == 0) feb -= 1;
      if(year % 400 == 0) feb += 1;
   }
   var days = new Array(31,feb,31,30,31,30,31,31,30,31,30,31);
   if(day > days[month-1] || day < 1)
      return false;
   return true;
}

function point_to(name) {
   document.getElementsByName(name)[0].focus();
}

function parseOint(x) {
   y = parseOint(x * 10);
   y = y / 10;
   return(y);
}
```

Folgende Funktionen sind in diesem Listing enthalten:

- `check_radios()`: Prüft, ob von einer Gruppe von Radiobuttons einer aktiviert ist. Erwartet als Parameter den gemeinsamen Wert des name-Attributs einer Gruppe von Radiobuttons. Ist einer der Buttons aktiviert, wird `true` zurückgegeben, wenn nicht, dann `false`.
- `check_field_length()`: Prüft, ob die Anzahl Zeichen in einem Feld einen bestimmten Wert nicht überschreitet. Erwartet als ersten Parameter den Wert des name-Attributs des Eingabefelds und als zweiten Parameter die maximal erlaubte Anzahl Zeichen. Ist die Eingabe nicht länger als erlaubt, wird `true` zurückgegeben, andernfalls `false`.
- `check_field_min_length()`: Prüft, ob in einem Feld mindestens eine bestimmte Anzahl Zeichen eingegeben wurden. Erwartet als ersten Parameter den Wert des name-Attributs des Eingabefelds und als zweiten Parameter die Mindestanzahl von Zeichen, die das Feld enthalten sollte. Ist die Eingabe mindestens so lang wie angegeben, wird `true` zurückgegeben, andernfalls `false`.
- `check_field_str_range()`: Prüft, ob ein Feld einen Zeichenkettenwert aus einem bestimmten Bereich erlaubter Werte enthält. Erwartet als ersten Parameter den Wert des name-Attributs des Eingabefelds und als zweiten Parameter eine Zeichenkette,

welche die erlaubten Werte enthält, wobei die einzelnen Werte durch Kommata voneinander getrennt sein müssen. Kommt der eingegebene Feldwert in den erlaubten Werten vor, wird `true` zurückgegeben, wenn nicht, dann `false`.

- **`check_field_int_range()`:** Prüft, ob ein Feld einen Ganzzahlwert aus einem bestimmten Bereich enthält. Erwartet als ersten Parameter den Wert des name-Attributs des Eingabefelds, als zweiten Parameter den Mindestwert des erlaubten Bereichs und als dritten Parameter den Höchstwert des erlaubten Bereichs. Kommt der eingegebene Feldwert im erlaubten Bereich vor, wird `true` zurückgegeben, wenn nicht, dann `false`.
- **`check_valid_date()`:** Prüft, ob ein Datum, bestehend aus Tag, Monat und Jahr, kalendarisch gültig ist. Erwartet als ersten Parameter den Wert für »Tag«, als zweiten den für »Monat« und als dritten den für »Jahr«. Alle Parameter müssen bereits numerisch übergeben werden. Ist das Datum kalendarisch korrekt, wird `true` zurückgegeben, wenn nicht, dann `false`.
- **`point_to()`:** Setzt den Cursor in ein bestimmtes Eingabefeld. Erwartet als Parameter den Wert des name-Attributs des gewünschten Eingabefelds.
- **`parseOint()`:** Hilfsfunktion, die Eingaben wie 08 oder 09 in numerischen Feldern davor bewahrt, von JavaScript als Fehler interpretiert zu werden. Wird im Script anstelle der Standardfunktion `parseInt()` aufgerufen.

Erkennbar ist, dass die Funktionen die Aufgabenstellungen widerspiegeln. Die einzelnen Check-Funktionen tun noch nichts weiter, als ihre Prüfung vorzunehmen und `true` oder `false` zurückzugeben. Erst die aufrufende Funktion entscheidet dann, was auf Grund des Rückgabewerts zu tun ist. Dies ist typisch funktionsorientiertes Programmieren nach dem Vorbild von C. JavaScript ist für eine solche Art von Programmierung bestens geeignet.

Alle Check-Funktionen greifen über die Methode `document.getElementsByName()` auf die Feldinhalte zu. Das bietet sich an, da die Formularfelder ohnehin ein name-Attribut erhalten müssen, welches für die spätere Datenverarbeitung auf dem Server von Bedeutung ist. Die Check-Funktionen greifen stets auf `document.getElementsByName()[0]` zu, da sie davon ausgehen, dass alle name-Attribute von Eingabefeldern dokumentweit eindeutig sind. Nur bei Radiobuttons, die jeweils eine Gruppe bilden, liegt der Fall anders.

Zur Funktion `check_valid_date()` ist noch anzumerken, dass sie das übergebene Jahr prüft und nur einen Wertebereich von 1801 bis 2799 zulässt. Dieser Wertebereich ist nicht streng begründbar, aber auch nicht willkürlich. Dahinter stehen Argumentationen rund um die Verbreitung und die künftige Korrektheit des gregorianischen Kalenders. Und für unser Horoskop-Beispiel sollte dieser Wertebereich allemal ausreichen. Die Funktion prüft ansonsten noch, ob der angegebene Monat überhaupt so viele Tage hat wie als Monatstag angegeben. Dabei werden, was den Februar betrifft, auch Schaltjahre inklusive 100-Jahres- und 400-Jahres-Regelung berücksichtigt. Ferner wird überprüft, ob der Monat sich im gültigen Bereich zwischen 1 und 12 befindet.

Die Funktion `point_to()` wird vom Fehlerhandling bei Auftreten eines Fehlers aufgerufen, um den Cursor nach Aufforderung des Anwenders zur Korrektur in das gewünschte Feld zu positionieren. Dabei wird die focus()-Methode verwendet, die auf HTML-Formularelemente anwendbar ist.

Ein Problem, das schon einige JavaScript-Programmierer fast verzweifeln ließ, ist, wenn die Standardmethode `parseInt()` auf eine Formulareingabe angewendet wird und deren Inhalt mit 08 oder 09 beginnt. In unserem Fall kann genau das sehr häufig vorkommen, da zahlreiche Anwender beim Ausfüllen von Tag, Monat, Stunde und Minute führende Nullen eingeben. Das Problem bei `parseInt()` besteht darin, dass es Werte mit führender 0 als Oktalwert interpretiert. Die Folgen 08 und 09 sind dann aber ungültig, da Oktalzahlen nur aus den Ziffern 0 bis 7 bestehen können. Mithilfe unserer »Tricky«-Funktion `parseOint()` lösen wir dieses Problem. Die Funktion multipliziert einfach den ihr übergebenen Wert mit 10. So wird aus 08 der Wert 80. Dieser Wert wird dann wieder durch 10 dividiert, was 8 ergibt. Dadurch ist ein ordentlicher numerischer Wert erreicht.

Damit kommen wir zur Funktion, welche das Fehlerhandling übernimmt:

```
function handle_error(number) {
   var error_text = new Array(
     "Hinweis: Keine Eingabenüberprüfung im Browser möglich!",
     "Art des Horoskops nicht ausgewählt!",
     "Keinen Vornamen eingegeben!",
     "Vorname zu lang!",
     "Keinen Zunamen eingegeben!",
     "Zuname zu lang!",
     "Kein Geschlecht ausgewählt!",
     "Geburtstag: Tag fehlt oder ist ungültig!",
     "Geburtstag: Monat fehlt oder ist ungültig!",
     "Geburtstag: Jahr fehlt oder ist ungültig!",
     "Geburtszeit: Stunden fehlen oder sind ungültig!",
     "Geburtszeit: Minuten fehlen oder sind ungültig!",
     "Geburtstag: ungültiges Kalenderdatum!",
     "Keinen Geburtsort eingegeben!",
     "Geburtsort zu lang!",
     "Länge: Gradangabe fehlt oder ist ungültig!",
     "Länge: Minutenangabe fehlt oder ist ungültig!",
     "Länge: 'West' oder 'Ost' eingeben!",
     "Breite: Gradangabe fehlt oder ist ungültig!",
     "Breite: Minutenangabe fehlt oder ist ungültig!",
     "Breite: 'Nord' oder 'Süd' eingeben!",
     "Partner: keinen Vornamen eingegeben!",
     "Partner: Vorname zu lang!",
     "Partner: keinen Zunamen eingegeben!",
     "Partner: Zuname zu lang!",
     "Partner: kein Geschlecht ausgewählt!",
     "Partner-Geburtstag: Tag fehlt oder ist ungültig!",
     "Partner-Geburtstag: Monat fehlt oder ist ungültig!",
     "Partner-Geburtstag: Jahr fehlt oder ist ungültig!",
     "Partner-Geburtstag: ungültiges Kalenderdatum!",
     "Partner-Geburtszeit: Stunden fehlen oder sind ungültig!",
     "Partner-Geburtszeit: Minuten fehlen oder sind ungültig!",
     "Partner: Keinen Geburtsort eingegeben!",
     "Partner: Geburtsort zu lang!",
     "Partner: Länge: Gradangabe fehlt oder ist ungültig!",
     "Partner: Länge: Minutenangabe fehlt oder ist ungültig!",
```

```
      "Partner: Länge: 'West' oder 'Ost' eingeben!",
      "Partner: Breite: Gradangabe fehlt oder ist ungültig!",
      "Partner: Breite: Minutenangabe fehlt oder ist ungültig!",
      "Partner: Breite: 'Nord' oder 'Süd' eingeben!",
      "Art der Lieferung nicht ausgewählt!",
      "Keinen Adressaten eingegeben!",
      "Adressat zu lang!",
      "Keine Lieferanschrift eingegeben!",
      "Lieferanschrift zu lang!"
    );
    alert(error_text[number]);
    switch(number) {
       case 0: break;
       case 1: break;
       case 2: case 3: point_to('given_name'); break;
       case 4: case 5: point_to('family_name'); break;
       case 6: break;
       case 7: point_to('birthday_day'); break;
       case 8: point_to('birthday_month'); break;
       case 9: point_to('birthday_year'); break;
       case 10: point_to('birthtime_hours'); break;
       case 11: point_to('birthtime_minutes'); break;
       case 12: point_to('birthday_day'); break;
       case 13: case 14: point_to('birth_place'); break;
       case 15: point_to('long_deg'); break;
       case 16: point_to('long_min'); break;
       case 17: point_to('long_dir'); break;
       case 18: point_to('lat_deg'); break;
       case 19: point_to('lat_min'); break;
       case 20: point_to('lat_dir'); break;
       case 21: case 22: point_to('p_given_name'); break;
       case 23: case 24: point_to('p_family_name'); break;
       case 25: break;
       case 26: point_to('p_birthday_day'); break;
       case 27: point_to('p_birthday_month'); break;
       case 28: point_to('p_birthday_year'); break;
       case 29: point_to('p_birthtime_hours'); break;
       case 30: point_to('p_birthtime_minutes'); break;
       case 31: point_to('p_birthday_day'); break;
       case 32: case 33: point_to('p_birth_place'); break;
       case 34: point_to('p_long_deg'); break;
       case 35: point_to('p_long_min'); break;
       case 36: point_to('p_long_dir'); break;
       case 37: point_to('p_lat_deg'); break;
       case 38: point_to('p_lat_min'); break;
       case 39: point_to('p_lat_dir'); break;
       case 40: break;
       case 41: case 42: point_to('delivery_person'); break;
       case 43: case 44: point_to('delivery_address'); break;
       default: break;
    }
    return false;
}
```

Die Funktion, `handle_error()`, bekommt beim Aufruf einfach eine Fehlernummer übergeben. Der längere Array aus Fehlermeldungstexten ist so definiert, dass aufgrund der übergebenen Fehlernummer die gleichlautende Array-Indexnummer die zugehörige Fehlermeldung liefert. Mit der alert()-Methode wird die so ermittelte Meldung im Anschluss an den Array auch gleich ausgegeben. An diesem Moment der Script-Ausführung bekommt der Anwender also einen bei der Eingabeprüfung aufgetretenen Fehler in einem Meldungsfenster angezeigt.

Anschließend soll in das betroffene Feld positioniert werden, damit der Anwender den genannten Fehler gleich korrigieren kann. Dazu muss die Funktion auch wieder alle denkbaren Fehlernummern einzeln unterscheiden. Das tut sie in einem switch-case-Konstrukt. Abhängig von der übergebenen Fehlernummer wird in das zugehörige Feld positioniert.

Am Ende gibt die Funktion `handle_error()` den Wert `false` zurück. Das ist nicht unwichtig, wie wir gleich noch sehen werden. Die Funktion `handle_error()` ist nämlich so programmiert, dass sie von der eigentlichen Zentralfunktion der Formularüberprüfung, nämlich der Funktion `check_form()`, optimal bedient werden kann. Zum Schluss noch der Code der zentralen Funktion:

```
function check_form() {
   if(!document.getElementsByName) {
      handle_error(0);
      return(true);
   }
   var chk = check_radios('horoscope_type');
   if(chk == false) return(handle_error(1));
   chk = check_field_min_length('given_name', 1);
   if(chk == false) return(handle_error(2));
   chk = check_field_length('given_name', 60);
   if(chk == false) return(handle_error(3));
   chk = check_field_min_length('family_name', 1);
   if(chk == false) return(handle_error(4));
   chk = check_field_length('family_name', 60);
   if(chk == false) return(handle_error(5));
   chk = check_radios('gender');
   if(chk == false) return(handle_error(6));
   chk = check_field_int_range('birthday_day', 1, 31);
   if(chk == false) return(handle_error(7));
   chk = check_field_int_range('birthday_month', 1, 12);
   if(chk == false) return(handle_error(8));
   chk = check_field_int_range('birthday_year', 1801, 2799);
   if(chk == false) return(handle_error(9));
   chk = check_field_int_range('birthtime_hours', 0, 23);
   if(chk == false) return(handle_error(10));
   chk = check_field_int_range('birthtime_minutes', 0, 59);
   if(chk == false) return(handle_error(11));
   var d = document.getElementsByName('birthday_day')[0].value;
   var m = document.getElementsByName('birthday_month')[0].value;
   var y = document.getElementsByName('birthday_year')[0].value;
   chk = check_valid_date(parseOint(d), parseOint(m), parseOint(y));
   if(chk == false) return(handle_error(12));
```

```
chk = check_field_min_length('birth_place', 1);
if(chk == false) return(handle_error(13));
chk = check_field_length('birth_place', 100);
if(chk == false) return(handle_error(14));
chk = check_field_int_range('long_deg', 0, 180);
if(chk == false) return(handle_error(15));
chk = check_field_int_range('long_min', 0, 59);
if(chk == false) return(handle_error(16));
chk = check_field_str_range('long_dir', 'Ost,West');
if(chk == false) return(handle_error(17));
chk = check_field_int_range('lat_deg', 0, 90);
if(chk == false) return(handle_error(18));
chk = check_field_int_range('lat_min', 0, 59);
if(chk == false) return(handle_error(19));
chk = check_field_str_range('lat_dir', 'Nord,Süd');
if(chk == false) return(handle_error(20));
type = document.getElementsByName('horoscope_type');
if(type[1].checked == true) {
   chk = check_field_min_length('p_given_name', 1);
   if(chk == false) return(handle_error(21));
   chk = check_field_length('p_given_name', 60);
   if(chk == false) return(handle_error(22));
   chk = check_field_min_length('p_family_name', 1);
   if(chk == false) return(handle_error(23));
   chk = check_field_length('p_family_name', 60);
   if(chk == false) return(handle_error(24));
   chk = check_radios('p_gender');
   if(chk == false) return(handle_error(25));
   chk = check_field_int_range('p_birthday_day', 1, 31);
   if(chk == false) return(handle_error(26));
   chk = check_field_int_range('p_birthday_month', 1, 12);
   if(chk == false) return(handle_error(27));
   chk = check_field_int_range('p_birthday_year', 1801, 2799);
   if(chk == false) return(handle_error(28));
   chk = check_field_int_range('p_birthtime_hours', 0, 23);
   if(chk == false) return(handle_error(29));
   chk = check_field_int_range('p_birthtime_minutes', 0, 59);
   if(chk == false) return(handle_error(30));
   d = document.getElementsByName('p_birthday_day')[0].value;
   m = document.getElementsByName('p_birthday_month')[0].value;
   y = document.getElementsByName('p_birthday_year')[0].value;
   chk = check_valid_date(parseOint(d), parseOint(m), parseOint(y));
   if(chk == false) return(handle_error(31));
   chk = check_field_min_length('p_birth_place', 1);
   if(chk == false) return(handle_error(32));
   chk = check_field_length('p_birth_place', 100);
   if(chk == false) return(handle_error(33));
   chk = check_field_int_range('p_long_deg', 0, 180);
   if(chk == false) return(handle_error(34));
   chk = check_field_int_range('p_long_min', 0, 59);
   if(chk == false) return(handle_error(35));
   chk = check_field_str_range('p_long_dir', 'Ost,West');
   if(chk == false) return(handle_error(36));
```

```
        chk = check_field_int_range('p_lat_deg', 0, 90);
        if(chk == false) return(handle_error(37));
        chk = check_field_int_range('p_lat_min', 0, 59);
        if(chk == false) return(handle_error(38));
        chk = check_field_str_range('p_lat_dir', 'Nord,Süd');
        if(chk == false) return(handle_error(39));
      }
      var chk = check_radios('delivery_type');
      if(chk == false) return(handle_error(40));
      chk = check_field_min_length('delivery_person', 1);
      if(chk == false) return(handle_error(41));
      chk = check_field_length('delivery_person', 160);
      if(chk == false) return(handle_error(42));
      chk = check_field_min_length('delivery_address', 1);
      if(chk == false) return(handle_error(43));
      chk = check_field_length('delivery_address', 160);
      if(chk == false) return(handle_error(44));
      if(chk == true) return true;
    }
```

Die Zentralfunktion `check_form()` geht der Reihe nach alle Felder des Formulars durch. Entsprechend der Vorgaben ruft sie eine der wiederverwendbaren Einzelfunktionen zur Überprüfung des jeweiligen Feldinhalts auf. Den Rückgabewert dieser Funktionen speichert sie in der Variablen `chk`. Nach jedem Aufruf dieser Art wird mit `if(chk == false)` geprüft, ob die Einzelprüfung des jeweiligen Feldinhalts einen Fehler ergeben hat. Ist das der Fall, so wird eine return-Anweisung ausgeführt. Als Wert wird der return-Anweisung ein Aufruf der Funktion `handle_error()` übergeben, und zwar mit der gewünschten Fehlernummer. Die Funktion `handle_error()` sorgt, wie bereits bekannt, für die Ausgabe der zugehörigen Fehlermeldung und für die Korrekturpositionierung ins zugehörige Feld. Da `handle_error()` am Ende den Boolean-Wert `false` zurückgibt, bedeutet ein Konstrukt wie `return(handle_error(44))` letztlich auch nur `return(false)`.

Der Vorteil dieser Konstrukte ist, dass die Funktion `check_form()` sofort nach Erledigung der Arbeiten von `handle_error()` beendet wird. Der Rückgabewert `false` landet beim aufrufenden Event-Handler `onsubmit` im einleitenden `<form>`-Tag. Dieser gibt ihn ebenfalls wieder mit `return` weiter:

```
<form id="order" action="horoskop2.htm" method="post"
      onsubmit="return(check_form())">
```

Damit wird der false-Wert an den Browser selbst geleitet. Dieser reagiert so, dass das Formular nicht abgesendet wird, was ja im Fehlerfall wichtig ist.

Fazit

Das Beispiel zeigt, dass bei ordentlichen Prüfungen von Formularfeldinhalten schnell eine ganze Menge Script-Code entstehen kann. Nutznießer davon sind jedoch einerseits der Anwender, der im Fehlerfall präzise Meldungen erhält und gleich das betreffende Feld korrigieren kann, und der Seitenanbieter, der ordentlich verarbeitbare Daten zum Server übertragen bekommt.

8.2 Navigation im Explorer-Stil

An kaum eine Form der Navigation haben sich Anwender in den letzten Jahren so gewöhnt wie an diejenige im Stil des Windows Explorers. Längst findet sich diese Art des Auswählens in allen erdenklichen Anwendungen wieder, egal ob Vektorgrafikprogramm, Programmieroberfläche, elektronische Steuererklärung oder Bookmarks im Browser.

In HTML gibt es von Haus aus kein Element, das eine solche Navigation realisiert (leider, denn eigentlich darf von einer Sprache, die »Hypertext« im Namen trägt, durchaus erwartet werden, dass sie mehr Möglichkeiten zur Realisierung einer Navigation bietet als einfache Hyperlinks).

Im Rahmen des CSS-Praxiskapitels (***Abschnitt 5.2***) lernten Sie bereits Möglichkeiten kennen, eine attraktive Navigation für eigene Websites zu realisieren, und zwar allein mit HTML und CSS. Für explorerartige Menüs geht es nicht ohne ein Minimum an JavaScript. Wir werden in unserem Beispiel mit überraschend wenig JavaScript auskommen. Das Beispiel erfüllt auch noch einige andere wichtige Kriterien:

- Das HTML-Markup der Navigation soll strukturell sinnvoll sein, so dass es auch bei nicht grafischen Browsern ordentlich aussieht. Gerade weil Projekte, die ein baumartiges Menü anbieten, in der Regel aus sehr vielen Unterseiten bestehen, ist ein strukturell gut nachvollziehbarer Aufbau der Navigation wichtig. Wir werden in unserem Beispiel mit verschachtelten Aufzählungslisten arbeiten.
- Die Navigation soll auch bei deaktiviertem JavaScript einwandfrei und ohne Zusatzaufwand funktionieren. Auch das ist bei unserem Beispiel der Fall.
- Das Auf- und Zuklappen von Teilbäumen soll aus Anwendersicht schnell reagieren und auch bei größeren Menüstrukturen nicht zu merklichen Verzögerungen führen.
- Der aktuelle Menüzustand (auf- und zugeklappte Teilbäume) soll an die nächste aufgerufene Seite übergeben werden, so dass er beim Seitenwechsel erhalten bleibt. Andernfalls wäre es für den Anwender mühsam, sich nach jedem Seitenwechsel wieder erneut durch die Struktur zu klicken.
- Das Script soll wiederverwendbar sein, d.h. auf beliebige andere Menüstrukturen anwendbar.

8.2.1 Das Beispiel

Navigationsstrukturen dieser Art bieten sich vor allem für größere technische Dokumentationen, für wissenschaftliche Abhandlungen und für systematische Darstellungen von Wissensgebieten an. In unserem Beispiel zeigen wir dies anhand eines kleinen Ausschnitts der Systematik der Tierwelt (Fauna).

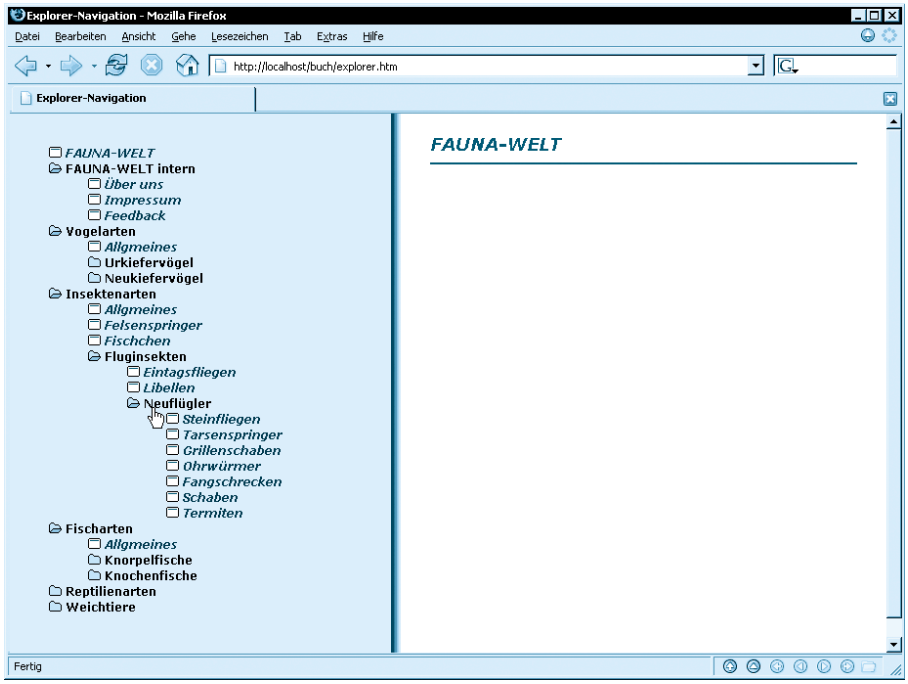


Abbildung 8.2: Angedeutete Fauna-Systematik mithilfe eines explorerartigen Menüs

8.2.2 Quelltexte und Erläuterungen

Zunächst der Quelltext des HTML-Dokuments. Der JavaScript-Code wird in eine externe Datei ausgelagert, da er ja auf allen Seiten wiederverwendet werden soll:

Listing 8.3: HTML-Quelltext mit verschachtelter Aufzählungsliste für Explorer-Menü

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Explorer-Navigation</title>
<link rel="stylesheet" type="text/css" href="explorer.css">
<script type="text/javascript" src="explorer.js"></script>
</head>
<body onload="init_nav()">
<div id="navigation">
<ul class="opened" id="root">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=001">FAUNA-WELT</a></li>
<li id="faunawelt_intern">
<a class="folder" onclick="toggle('faunawelt_intern',true)">
<img src="opened.gif" alt="" id="img_faunawelt_intern">
FAUNA-WELT intern</a>
```

```
<ul id="ul_faunawelt_intern" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=002">Über uns</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=010">Impressum</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=002">Feedback</a></li>
</ul><!-- Ende: ul_faunawelt_intern -->
</li>
<li id="vogelarten">
<a class="folder" onclick="toggle('vogelarten',true);">
<img src="opened.gif" alt="" id="img_vogelarten">
Vogelarten</a>
<ul id="ul_vogelarten" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=004">Allgemeines</a></li>
<li id="urkiefervoegel">
<a class="folder" onclick="toggle('urkiefervoegel',true);">
<img src="opened.gif" alt="" id="img_urkiefervoegel">
Urkiefervögel</a>
<ul id="ul_urkiefervoegel" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=024">Laufvögel</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=028">Steißhühner</a></li>
</ul><!-- Ende: ul_urkiefervoegel -->
</li>
<li id="neukiefervoegel">
<a class="folder" onclick="toggle('neukiefervoegel',true);">
<img src="opened.gif" alt="" id="img_neukiefervoegel">
Neukiefervögel</a>
<ul id="ul_neukiefervoegel" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=064">Hühnervögel</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=038">Gänsevögel</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=055">Kranichvögel</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=061">Schreitvögel</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=086">Ruderfüßer</a></li>
</ul><!-- Ende: ul_neukiefervoegel -->
</li>
</ul><!-- Ende: ul_vogelarten -->
</li>
<li id="insektenarten">
<a class="folder" onclick="toggle('insektenarten',true);">
<img src="opened.gif" alt="" id="img_insektenarten">
Insektenarten</a>
<ul id="ul_insektenarten" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=108">Allgemeines</a></li>
```

```
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=029">Felsenspringer</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=012">Fischchen</a></li>
<li id="fluginsekten">
<a class="folder" onclick="toggle('fluginsekten',true);">
<img src="opened.gif" alt="" id="img_fluginsekten">
Fluginsekten</a>
<ul id="ul_fluginsekten" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=033">Eintagsfliegen</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=012">Libellen</a></li>
<li id="neufluegler">
<a class="folder" onclick="toggle('neufluegler',true);">
<img src="opened.gif" alt="" id="img_neufluegler">
Neuflügler</a>
<ul id="ul_neufluegler" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=093">Steinfliegen</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=065">Tarsenspringer</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=090">Grillenschaben</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=035">Ohrwürmer</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=071">Fangschrecken</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=080">Schaben</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=082">Termiten</a></li>
</ul><!-- Ende: ul_neufluegler -->
</li>
</ul><!-- Ende: ul_fluginsekten -->
</li>
</ul><!-- Ende: ul_insektenarten -->
</li>
<li id="fischarten">
<a class="folder" onclick="toggle('fischarten',true);">
<img src="opened.gif" alt="" id="img_fischarten">
Fischarten</a>
<ul id="ul_fischarten" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=075">Allgemeines</a></li>
<li id="knorpelfische">
<a class="folder" onclick="toggle('knorpelfische',true);">
<img src="opened.gif" alt="" id="img_knorpelfische">
Knorpelfische</a>
<ul id="ul_knorpelfische" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=074">Plattenkiemer</a></li>
<li><img src="doc.gif" alt="">
```

```
<a class="nav" href="/index.php?id=098">Seekatzen</a></li>
</ul><!-- Ende: ul_knorpelfische -->
</li>
<li id="knochenfische">
<a class="folder" onclick="toggle('knochenfische',true);">
<img src="opened.gif" alt="" id="img_knochenfische">
Knochenfische</a>
<ul id="ul_knochenfische" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=024">Flössler</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=108">Knorpelganoide</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=102">Knochenganoide</a></li>
<li id="echte_knochenfische">
<a class="folder" onclick="toggle('echte_knochenfische',true);">
<img src="opened.gif" alt="" id="img_echte_knochenfische">
Echte Knochenfische</a>
<ul id="ul_echte_knochenfische" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=049">
Knochenzünglerartige</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=067">Aalartige</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=112">Lachsartige</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=099">Hechtartige</a></li>
</ul><!-- Ende: ul_echte_knochenfische -->
</li>
</ul><!-- Ende: ul_knochenfische -->
</li>
</ul><!-- Ende: ul_fischarten -->
</li>
<li id="reptilienarten">
<a class="folder" onclick="toggle('reptilienarten',true);">
<img src="opened.gif" alt="" id="img_reptilienarten">
Reptilienarten</a>
<ul id="ul_reptilienarten" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=144">Allgemeines</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=124">Anapsida</a></li>
<li id="romeriida">
<a class="folder" onclick="toggle('romeriida',true);">
<img src="opened.gif" alt="" id="img_romeriida">
Romeriida</a>
<ul id="ul_romeriida" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=029">Diapsida</a></li>
</ul><!-- Ende: ul_romeriida -->
</li>
</ul><!-- Ende: ul_reptilienarten -->
```

```
</li>
<li id="weichtiere">
<a class="folder" onclick="toggle('weichtiere',true);">
<img src="opened.gif" alt="" id="img_weichtiere">
Weichtiere</a>
<ul id="ul_weichtiere" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=123">Allgemeines</a></li>
<li id="stachelweichtiere">
<a class="folder" onclick="toggle('stachelweichtiere',true);">
<img src="opened.gif" alt="" id="img_stachelweichtiere">
Stachelweichtiere</a>
<ul id="ul_stachelweichtiere" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=106">Wurmmollusken</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=115">Käferschnecken</a></li>
</ul><!-- Ende: ul_stachelweichtiere -->
</li>
<li id="schalenweichtiere">
<a class="folder" onclick="toggle('schalenweichtiere',true);">
<img src="opened.gif" alt="" id="img_schalenweichtiere">
Stachelweichtiere</a>
<ul id="ul_schalenweichtiere" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=073">Gekrümmtschaler</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=079">Gestrecktschaler</a></li>
</ul><!-- Ende: ul_schalenweichtiere -->
</li>
</ul><!-- Ende: ul_weichtiere -->
</li>
</ul>
</div>
<div id="content">
<h1>FAUNA-WELT intern: über uns</h1>
</div>
</body>
</html>
```

Bevor wir auf die Details eingehen, ein Wort zum Thema »Vermeidung von Frames«. Da die Menüstruktur doch recht umfangreich ist und bei Verzicht auf Frames auf jeder Seite wiederholt werden muss, entsteht natürlich eine Menge vielfach wiederholter HTML-Quellcode. Solange mit statischen HTML-Dateien gearbeitet wird, ist es ohne Frames tatsächlich nicht vermeidbar, den Code auf jeder Unterseite zu wiederholen. Sobald jedoch serverseitige Techniken, egal ob Server Side Includes (SSI), PHP oder eine andere Scriptsprache zum Einsatz kommen, kann der entsprechende Code in ein Template ausgelagert werden, dass dann einfach nur dynamisch integriert wird. Auf diese Möglichkeiten gehen wir jedoch erst in Zusammenhang mit serverseitigen Techniken ein.

Der body-Bereich unseres Beispiels besteht aus zwei div-Bereichen namens *navigation* und *content*. Ein Ausschnitt aus der CSS-Datei zeigt, wie das Seitenlayout sich grundsätzlich zusammensetzt:

```
html, body {
   margin:0;
   padding:0;
   height:100%;
}

body {
   background-color:white;
   color:black;
   background-image:url(fauna_back.png);
   background-repeat:repeat-y;
   font-family:Verdana,sans-serif;
}

#navigation {
  position:absolute;
  top:20px;
  left:0px;
  width:400px;
  overflow:hidden;
}

#content {
  margin-top:20px;
  margin-right:30px;
  margin-left:430px;
}
```

Der Bereich für die Navigation ist absolut positioniert und erhält eine großzügige Breite von 400 Pixel. Das ist für stark verzweigte Baumstrukturen durchaus angemessen. Falls der Inhalt zu breit ist, wird er abgeschnitten (`overflow:hidden`). Der hellgrüne Bereich mit dem Schatteneffekt, der als Hintergrund der Navigation dient, ist durch ein Hintergrundbild realisiert. Der Bereich `content` bleibt im normalen Textfluss, damit seine Breite sich an der zur Verfügung stehenden Breite des Browserfensters orientieren kann. Sein linker Rand beträgt 430 Pixel, da 400 Pixel davon dem Bereich für die Navigation gehören.

Der Bereich *navigation* besteht aus einer umfangreichen verschachtelten Aufzählungsliste, also aus ul- und li-Elementen. Das äußerste ul-Element erhält den id-Namen `root`. Dies wird vom Script ausgewertet und ist daher Pflicht.

Zur Erinnerung noch einmal das Prinzip verschachtelter Aufzählungslisten in HTML:

```
<ul>
   <li>Listenpunkt</li>
   <li>Unterliste:
      <ul>
         <li>Listenpunkt</li>
```

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

Eine untergeordnete Liste beginnt also damit, dass ein li-Element der übergeordneten Liste mit einem »Überschriftentext« für die Unterliste beginnt. Daran anschließend wird die innere ul-Liste komplett notiert. Abschließend wird das schließende `</li>`-Tag notiert. Untergeordnete Listen müssen also als Elementinhalt eines li-Elements der übergeordneten Liste notiert werden.

In unserem Beispiel erhält jedes ul-Element sowohl einen id-Namen (`id=`) als auch eine Klasse (`class=`), z. B.:

```
<ul id="ul_faunawelt_intern" class="opened">
```

Allen ul-Elementen der Navigation wird der Klassenname `opened` zugewiesen. In CSS ist für diese Klasse Folgendes definiert:

```
ul.opened {
  display:block;
}
```

Das mag trivial erscheinen, da es der Default-Einstellung entspricht. Im Script werden wir jedoch über den Klassennamen das Auf- und Zuklappen von Teilbäumen realisieren und deshalb wird in der CSS-Datei bereits folgende »Parallelklasse« eingerichtet:

```
ul.closed {
  display:none;
}
```

Für beide Klassen werden im Beispiel außerdem folgende Gemeinsamkeiten festgelegt:

```
ul.closed, ul.opened {
  font-size:12px;
  list-style-type:none;
  font-weight:bold;
  line-height:16px;
}
```

Der id-Name jedes ul-Elements sollte den Teilbaum, den die jeweilige ul-Liste enthält, möglichst sprechend charakterisieren. Außerdem – und das ist für unser Script wichtig – ***muss*** der Name mit `ul_` beginnen. Als Alternative zu sprechenden id-Namen können Sie natürlich auch ein Namensschema wie ***ul_section_1_1***, ***ul_section_1_2*** usw. verwenden. Letzteres hat jedoch den Nachteil, dass es weniger änderungsfreundlich ist, wenn mal neue Seiten eingefügt werden sollen.

Betrachten wir nun ein li-Element, welches eine untergeordnete ul-Liste eröffnet, im Zusammenhang:

```
<li id="faunawelt_intern">
<a class="folder" onclick="toggle('faunawelt_intern'),true">
<img src="opened.gif" alt="" id="img_faunawelt_intern">
FAUNA-WELT intern</a>
<ul id="ul_faunawelt_intern" class="opened">
...
</ul>
</li>
```

Jedes li-Element, das eine untergeordnete ul-Liste enthält, erhält ebenfalls einen id-Namen. Und zwar den gleichen id-Namen wie die ul-Liste, nur ohne voranstehendes `ul_`. Im Code-Ausschnitt hat das li-Element den id-Namen `faunawelt_intern` und die innere ul-Liste folgerichtig den id-Namen `ul_faunaweld_intern`. Dieser Zusammenhang muss bei allen inneren Listen nach der gleichen Regel hergestellt werden, damit unser Script korrekt arbeitet.

Das Gleiche gilt für weitere Regeln zum Aufbau des einleitenden Teils des li-Elements vor der inneren ul-Liste. Dieser Teil besteht aus einem a-Element, welches den Klassennamen `folder` erhalten muss. Diese Klasse ist in CSS wie folgt definiert:

```
a.folder {
  cursor:pointer;
}
```

Da kein href-Attribut notiert ist, wird kein echter Link angezeigt. Der li-Eintrag soll ja auch kein echter Link sein, sondern beim Anklicken lediglich den inneren Teilbaum auf- bzw. wieder zuklappen. Damit er als anklickbar erkennbar ist, erscheint der Mauscursor beim Überfahren wie bei einem Hyperlink.

Das so definierte a-Element bei li-Texten, welche eine untergeordnete ul-Liste einleiten, muss außerdem einen Event-Handler `onclick=` erhalten. Dieser muss als Wertzuweisung einen Aufruf der JavaScript-Funktion `toggle()` erhalten. Als erster Parameter muss der Funktion der id-Name des li-Elements übergeben werden und als zweiter der Wert `true`.

Der Elementinhalt des a-Elements besteht aus einer Grafikreferenz, gefolgt von dem eigentlichen Eintragstext. Die Grafik erhält den Dateinamen *opened.gif*. Sie sollte ein geöffnetes Ordnersymbol darstellen und in der Höhe nicht mehr als 14 Pixel haben. Das img-Element der Grafikreferenz erhält ebenfalls zwingend einen id-Namen, und zwar wieder den gleichen Basisnamen wie das li-Element, jedoch angeführt von der Zeichenfolge `img_`. Auch das ist für das Script wichtig.

Bleiben noch diejenigen li-Elemente, die echte Links zu aufrufbaren Seiten enthalten. Ein Beispielauszug:

```
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=002">Über uns</a></li>
```

Der Elementinhalt besteht hierbei aus einer Grafik namens *doc.gif*, welche ein Seitensymbol enthalten und ebenfalls nicht höher als 14 Pixel sein sollte. Anschließend folgt der Link zur Seite. Er erhält den Klassennamen `nav`. Für diese Klasse ist in CSS im Beispiel Folgendes definiert:

```
a.nav {
  font-weight:bold;
  font-style:italic;
}

a.nav:link, a.nav:visited, a.nav:hover, a.nav:active {
  color:rgb(8,108,8);
  text-decoration:none;
}
```

Wie aus den vorangehenden Beschreibungen deutlich wird, haben wir eine Menge »Intelligenz« bereits in den HTML-Code verlagert. Das ist erstens die Voraussetzung für ein schlankes JavaScript und zweitens wertet es den HTML-Code semantisch auf, indem durch id- und Klassennamen die inneren Bezüge verdeutlicht werden.

Damit zum Inhalt des Scripts, welches im Beispiel in eine externe Datei namens *explorer.js* ausgelagert wurde:

Listing 8.4: JavaScript für Explorer-Menü

```
function init_nav() {
   for(i = 0; i < document.getElementsByTagName('ul').length;
       i++) {
      if(document.getElementsByTagName('ul')[i].className ==
      "opened") {
         id =
         document.getElementsByTagName('ul')[i].parentNode.id;
         toggle(id, false);
      }
   }
   if(window.name.length > 0)
      load_nav();
}

function toggle(id, save) {
   ul = "ul_" + id;
   img = "img_" + id;
   ul_element = document.getElementById(ul);
   img_element = document.getElementById(img);
   if(ul_element) {
       if(ul_element.className == 'closed') {
          ul_element.className = "opened";
          img_element.src = "opened.gif";
       }
       else {
          ul_element.className = "closed";
          img_element.src = "closed.gif";
```

```
        }
    }
    if(save == true) save_nav();
}

function save_nav() {
    var save = "";
    for(var i = 0; i < document.getElementsByTagName('ul').length;
    i++) {
        if((document.getElementsByTagName('ul')[i].className ==
        "opened" ||
        document.getElementsByTagName('ul')[i].className ==
        "closed") && document.getElementsByTagName('ul')[i].id !=
        'root')
            save = save + document.getElementsByTagName('ul')[i].id
            + "=" + document.getElementsByTagName('ul')[i].className
            + ",";
    }
    if(save.lastIndexOf(",") > 0)
        save = save.substring(0, save.lastIndexOf(","));
    window.name = save;
}

function load_nav() {
    var items = window.name.split(",");
    if(items.length > 0) {
        for(var i = 0; i < items.length; i++) {
            id_value = items[i].split("=");
            if(id_value.length == 2) {
                id = id_value[0];
                value = id_value[1];
                document.getElementById(id).className = value;
                img = "img_" + id.substring(3, id.length);
                img_element = document.getElementById(img);
                if(value == "closed")
                    img_element.src = "closed.gif";
                else
                    img_element.src = "opened.gif";
            }
        }
    }
}
```

Im HTML-Code wird die Verbindung zum JavaScript an zwei Stellen hergestellt. Im einleitenden <body>-Tag ist `onload="init_nav()"` notiert. Die Funktion `init_nav()` ist im Script als Erste von insgesamt vier Funktionen notiert. Sie geht zunächst in einer for-Schleife alle ul-Elemente durch und erfragt in einer if-Bedingung, ob das jeweils aktuell betrachtete ul-Element den Klassennamen `opened` hat. Ist das der Fall, so wird das Elementobjekt des Elternelements des aktuellen ul-Elements in einer Variablen namens `id` gespeichert. Bei allen ul-Elementen außer der äußersten ul-Liste ist das Elternelement das li-Element, welches die innere ul-Liste einleitet. Dessen Elementobjekt wird in `id` gespeichert. Anschlie-

ßend wird die Funktion `toggle()` aufgerufen. Ihr wird als erster Parameter die Variable `id` übergeben und als zweiter Parameter der Wert `false`. Auf Letzteren kommen wir später noch zurück.

Die Funktion `toggle()` tut Folgendes: Wenn ein Teilbaum der Navigation aufgeklappt ist, schließt sie ihn. Ist er dagegen geschlossen, klappt sie ihn auf. Dazu benutzt sie die Klassennamen, welche die ul-Listen mit dem jeweiligen Teilbaum enthalten. In HTML – wir erinnern uns – haben wir bewusst alle ul-Elemente von Teilbäumen mit dem Klassennamen `opened` ausgestattet. In CSS bewirkt dieser `display:block` also eine normale Anzeige. Der Grund ist, dass bei ausgeschaltetem JavaScript der gesamte Navigationsbaum aufgeklappt angezeigt werden soll. Im Script möchten wir jedoch anfänglich einen Zustand herstellen, der den Baum zunächst zugeklappt darstellt. Genau das leistet die for-Schleife in `init_nav()`, die alle betroffenen Elemente ermittelt und durch Aufruf von `toggle()` dafür sorgt, dass diese zugeklappt werden.

Damit aber nicht genug. Nachdem die for-Schleife abgearbeitet ist, wird abgefragt, ob die Eigenschaft `window.name` eine Länge von mehr als 0 Zeichen hat. In ***Abschnitt 7.3.12*** wurde diese Default-Eigenschaft vorgestellt und gezeigt, wie sich damit JavaScript-Informationen seitenübergreifend speichern lassen. Unser Script benutzt diese Eigenschaft, um den aktuellen Auf-/Zuklappzustand der Navigation zu speichern und wieder laden zu können. Allein diesem Zweck dienen die beiden letzten Funktionen im Script, `save_nav()` und `load_nav()`. Die Funktion `toggle()` ruft gegebenenfalls `save_nav()` auf, um den aktuellen Menüzustand zu speichern, und `init_nav()` ruft `load_nav()` auf, um einen gespeicherten Menüzustand wiederherzustellen. Ist in `window.name` also etwas gespeichert, dann kommt `load_nav()` zum Einsatz.

Die zweite Schnittstelle zwischen HTML und dem JavaScript-Code sind die Pseudo-Links in den li-Elementen, welche eine untergeordnete ul-Liste enthalten. Zur Erinnerung nochmals der Beispielauszug:

```
<li id="faunawelt_intern">
<a class="folder" onclick="toggle('faunawelt_intern',true)">
<img src="opened.gif" alt="" id="img_faunawelt_intern">
FAUNA-WELT intern</a>
<ul id="ul_faunawelt_intern" class="opened">
...
</ul>
</li>
```

Im Event-Handler `onclick=` wird `toggle()` aufgerufen. Dabei wird als erster Parameter der id-Name des li-Elements übergeben und als zweiter der Wert `true`. Der zweite Parameter bestimmt, ob `toggle()` den aktuellen Menüzustand in `window.name` speichern lassen soll oder nicht. In `init_nav()` wird `toggle()` mit `false` aufgerufen. In diesem Fall wird nichts gespeichert. Da beim Anklicken von Ordner-Links jedoch eine vom Anwender bewirkte Zustandsveränderung im Menü eintritt, wird `true` übergeben, damit der neue Zustand gespeichert wird.

Die Funktion `toggle()` greift auf zwei Elemente zu: auf das ul-Element des untergeordneten Teilbaums und auf die Grafikreferenz, die je nach Aufklappzustand des Teilbaums entweder ein geöffnetes oder ein geschlossenes Ordnersymbol darstellen soll. Da sie als ersten Parameter den id-Namen des den Teilbaum öffnenden li-Elements übergeben bekommt, kann sie auf Basis der vereinbarten Regel durch Voranstellen von `ul_` bzw. `img_` die id-Namen der betroffenen Elemente selbst ermitteln. Das Herzstück des Auf-/Zuklappeffekts ist das if-else-Konstrukt innerhalb von `toggle()`. Ist der aktuelle Klassenname des inneren ul-Elements `opened`, wird er in `closed` geändert, und andernfalls umgekehrt: `closed` wird zu `opened`. Da der Klasse `closed` in CSS mit `display:none` zugewiesen wurde, verschwindet die Anzeige des Teilbaums, sobald diese Klasse zugewiesen ist. Damit sich auch das Ordnersymbol entsprechend ändert, wird durch Zuweisen einer anderen Grafik an das src-Attribut der Grafikreferenz im öffnenden li-Element der Klappeffekt zusätzlich visualisiert.

Die Funktionen `save_nav()` und `load_nav()` haben wie schon erwähnt nichts unmittelbar mit der Funktionalität des Auf-/Zuklappeffekts zu tun. In `save_nav()` besorgt eine for-Schleife die Zuweisung von id-Namen zu aktuellen Klassennamen und fügt die Einzelwerte durch Kommata getrennt zu einer Zeichenkette zusammen. Diese wird am Ende in `window.name` gespeichert. In `load_nav()` wird `window.name` wieder ausgelesen und die darin gespeicherte Zeichenkette wird durch Anwendung der Zeichenkettenmethode `split()` wieder in ihre Einzelwerte zerlegt. Ähnlich wie in `toggle()` werden die ermittelten Klassennamen den zugehörigen, über id-Namen ermittelten Elementen zugewiesen, jedoch nicht nur einem einzigen Element, sondern allen Elementen des Menüs. Auf diese Weise wird nach Aufruf einer anderen Seite der Menüzustand wiederhergestellt, der nach Verlassen der letzten Seite aktuell war.

Fazit

Das Beispiel zeigt, dass man in Zeiten des DOM nicht mehr viel JavaScript benötigt und außerdem noch ein sauberes HTML-Markup einsetzen kann, um die allseits beliebten Baumnavigationen zu realisieren.

8.3 Interaktive Tabellensortierung

Es gibt zahlreiche Informationen, die sich am besten in einer Tabelle darstellen lassen. In HTML müssen Sie sich dabei entscheiden, nach welchem Kriterium Sie die Tabellendaten sortieren, oder Sie verzichten ganz auf eine Sortierung.

Gerade bei größeren Tabellen wünscht sich mancher Anwender jedoch, die Tabellenzeilen nach bestimmten Spalten sortieren zu können. Das steigert die Informationsauswertung. Gewöhnt sind die Anwender das Feature aus zahlreichen Desktop- und Internetanwendungen, etwa aus E-Mail-Programmen, wo sich Nachrichtenlisten wahlweise nach Absender, Eingangsdatum usw. sortieren lassen.

Auf Webseiten sucht man diese Möglichkeit dagegen meist vergebens. Dabei ist gerade die Möglichkeit, größere Tabellen dynamisch zu sortieren, eine der interessantesten und

nützlichsten Anwendungen für JavaScript. Mit diesem Beispiel möchten wir ein Script vorstellen, das wiederverwendbar eingesetzt werden kann und mit dessen Hilfe Webseiten mit tabellarischen Informationen eine echte Aufwertung erfahren.

8.3.1 Das Beispiel

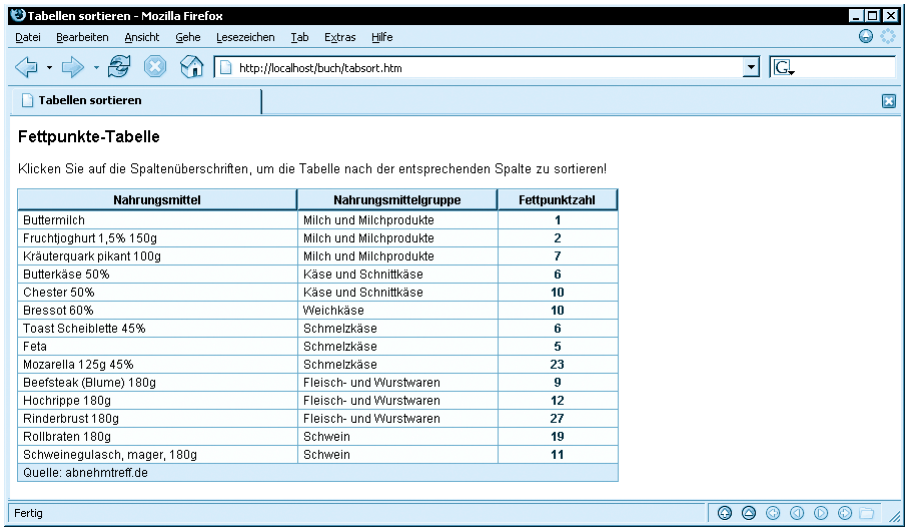


Abbildung 8.3: Tabelle im HTML-Originalzustand

Die abgebildete Tabelle ist durch Anklicken der Spaltenüberschriften nach der angeklickten Spalte sortierbar:

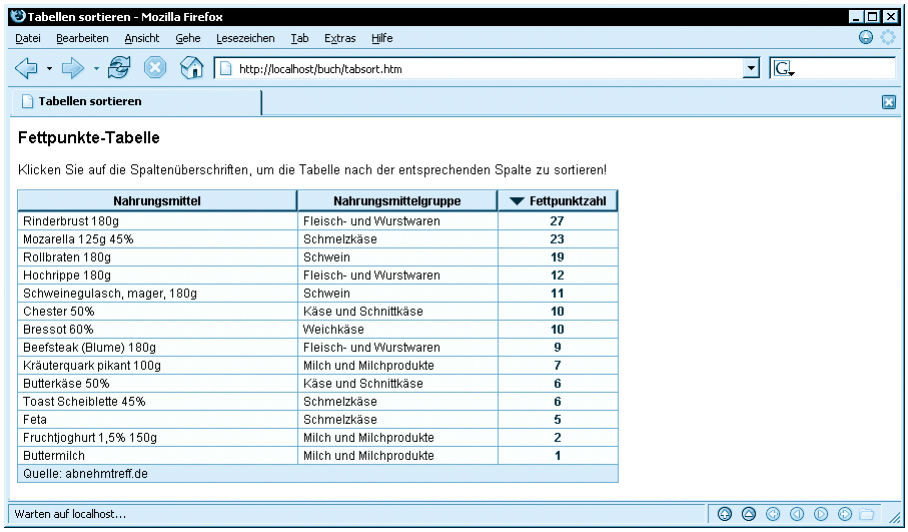


Abbildung 8.4: Tabelle nach letzter Spalte absteigend sortiert

Eine kleine Grafik vor der Spaltenüberschrift der Sortierspalte zeigt an, nach welcher Spalte die Tabelle aktuell sortiert ist, und zwar, ob absteigend oder aufsteigend. Es kann sowohl alphanumerisch als auch numerisch sortiert werden.

Unser Beispielscript kann beliebige Tabellen mit definierten Spalten verarbeiten, auch mehrere unterschiedliche auf einer Seite.

8.3.2 Quelltexte und Erläuterungen

Zunächst der Quelltext des HTML-Dokuments mit der zu sortierenden Tabelle. Das JavaScript haben wir in eine externe Datei ausgelagert.

Listing 8.5: HTML-Datei mit Tabelle zum Sortieren

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Tabellen sortieren</title>
<style type="text/css">
table {
   border-collapse:collapse;
   border:1px solid #999999;
}
th {
   vertical-align:middle;
   text-align:left;
   background-color:#DDDDDD;
   border:1px solid #999999;
   padding:0px;
}
th button {
   width:100%;
   background-color:#DDDDDD;
   font-family:Arial,sans-serif;
   font-size:12px;
   font-weight:bold;
   text-align:left;
   margin:0px;
}
td {
   font-family:Arial,sans-serif;
   font-size:12px;
   text-align:left;
   vertical-align:top;
   background-color:#FFFFDD;
   border:1px solid #999999;
   padding-left:5px;
   padding-right:5px;
   padding-top:1px;
```

```
   padding-bottom:1px;
}
td.fp {
   font-weight:bold;
   color:blue;
   text-align:center;
}
td.tfoot {
   background-color:#DDDDDD;
}
h1 {
   font-family:Arial,sans-serif;
   font-size:16px;
}
p {
   font-family:Arial,sans-serif;
   font-size:13px;
}
</style>
<script type="text/javascript" src="tabsort.js"></script>
</head>
<body onload="init_tabsort('fettpunkttabelle')">

<h1>Fettpunkte-Tabelle</h1>
<p>Klicken Sie auf die Spaltenüberschriften,
um die Tabelle nach der entsprechenden Spalte
zu sortieren!</p>
<form action="javascript:void(0)">
<table id="fettpunkttabelle" border="1"
summary="Tabelle, in der für bekannte Nahrungsmittel und
Nahrungsmittelgruppen Fettpunktzahlen angegeben werden">
<colgroup>
<col style="width:280px">
<col style="width:200px">
<col style="width:120px">
</colgroup>
<thead>
<tr>
<th><button type="button"
onClick="tabsort('fettpunkttabelle','an',0)">
<span id="fettpunkttabelle_sort_img_0"></span>
Nahrungsmittel</button></th>
<th><button  type="button"
onClick="tabsort('fettpunkttabelle','an',1)">
<span id="fettpunkttabelle_sort_img_1"></span>
Nahrungsmittelgruppe</button></th>
<th><button  type="button"
onClick="tabsort('fettpunkttabelle','n',2)">
<span id="fettpunkttabelle_sort_img_2"></span>
Fettpunktzahl</button></th>
</tr>
</thead>
<tfoot>
```

```
<tr>
<td colspan="3" class="tfoot">
Quelle: abnehmtreff.de
</td>
</tr>
</tfoot>
<tbody>
<tr>
<td>Buttermilch</td>
<td>Milch und Milchprodukte</td>
<td class="fp">1</td>
</tr>
<tr>
<td>Fruchtjoghurt 1,5% 150g</td>
<td>Milch und Milchprodukte</td>
<td class="fp">2</td>
</tr>
<tr>
<td>Kräuterquark pikant 100g</td>
<td>Milch und Milchprodukte</td>
<td class="fp">7</td>
</tr>
<tr>
<td>Butterkäse 50%</td>
<td>Käse und Schnittkäse</td>
<td class="fp">6</td>
</tr>
<tr>
<td>Chester 50%</td>
<td>Käse und Schnittkäse</td>
<td class="fp">10</td>
</tr>
<tr>
<td>Bressot 60%</td>
<td>Weichkäse</td>
<td class="fp">10</td>
</tr>
<tr>
<td>Toast Scheiblette 45%</td>
<td>Schmelzkäse</td>
<td class="fp">6</td>
</tr>
<tr>
<td>Feta</td>
<td>Schmelzkäse</td>
<td class="fp">5</td>
</tr>
<tr>
<td>Mozarella 125g 45%</td>
<td>Schmelzkäse</td>
<td class="fp">23</td>
</tr>
<tr>
```

```
<td>Beefsteak (Blume) 180g</td>
<td>Fleisch- und Wurstwaren</td>
<td class="fp">9</td>
</tr>
<tr>
<td>Hochrippe 180g</td>
<td>Fleisch- und Wurstwaren</td>
<td class="fp">12</td>
</tr>
<tr>
<td>Rinderbrust 180g</td>
<td>Fleisch- und Wurstwaren</td>
<td class="fp">27</td>
</tr>
<tr>
<td>Rollbraten 180g</td>
<td>Schwein</td>
<td class="fp">19</td>
</tr>
<tr>
<td>Schweinegulasch, mager, 180g</td>
<td>Schwein</td>
<td class="fp">11</td>
</tr>
</tbody>
</table>
</form>
</body>
</html>
```

Eine Tabelle, welche mit unserem Beispielscript sortiert werden soll, muss im einleitenden <table>-Tag mit `id=` einen id-Namen erhalten. Weiterhin muss sie unterhalb des <table>-Tags mithilfe eines colgroup-Elements alle Spalten der Tabelle in col-Elementen vordefinieren. Die col-Elemente sind auch der beste Ort, um die gewünschten Spaltenbreiten vorzugeben. Daneben muss die zu sortierende Tabelle zumindest die Bereiche `<thead> ... </thead>` und `<tbody> ... </tbody>` explizit durch diese Tags markiert enthalten.

Der Inhalt des thead-Elements besteht aus einer Tabellenzeile mit den Spaltenüberschriften. In unserem Beispiel haben wir es so gelöst, dass der Inhalt der Kopfzellen jeweils aus einem anklickbaren Button (`<button> ... </button>`) besteht. Mit dem Selektor `th button` wird im zentralen Style-Bereich für solche Buttons unter anderem festgelegt, dass sie 100% der verfügbaren Breite einnehmen sollen. So erreichen wir, dass die Buttons die gesamte Tabellenzelle einnehmen. Jedes button-Element enthält einen Event-Handler `onclick=`, der beim Anklicken dafür sorgt, dass die Tabelle sortiert wird. Dazu später noch mehr.

Der Elementinhalt jedes Buttons besteht aus einem leeren span-Element und der Spaltenüberschrift. Das span-Element ist der Behälter für die kleinen Grafiken, welche in der Sortierspalte erscheinen und auf- oder absteigende Sortierung anzeigen. Jedes dieser span-Elemente muss einen id-Namen erhalten, der folgender Konvention genügen muss: Er beginnt mit dem id-Namen der Tabelle, gefolgt von `_sort_img_` und einer Laufnummer.

Die erste Spalte bekommt eine 0, die zweite eine 1, die dritte eine 2 usw. Diese Konvention ist für das Script wichtig, um die Grafiken korrekt zu platzieren.

Ansonsten sind keine weiteren Konventionen erforderlich. Die Tabellendaten können wie üblich notiert werden.

Das Script, das die Tabelle sortiert, ist über ein script-Element in den Kopfdaten des HTML-Dokuments als externe Datei namens ***tabsort.js*** eingebunden. Sein Quelltext lautet:

Listing 8.6: JavaScript für HTML-Tabellensortierung

```
var default_order = "asc"; // "asc" || "desc"
var asc_img_uri = "asc.gif";
var desc_img_uri = "desc.gif";
var asc_img_alt_text =
    "aufsteigend sortiert: ändern in absteigend sortiert";
var desc_img_alt_text =
    "absteigend sortiert: ändern in aufsteigend sortiert";

var tables = new Array();
var ti = 0;

function sort_table(id) {
   this.id = id;
   this.obj;
   this.tbody_obj;
   this.sort_cols = 0;
   this.sort_rows = 0;
   this.img_ids = new Array();
   this.tr_ids = new Array();
   this.td_ids = new Array();
   this.active_sort_col;
   this.active_sort_order = default_order;
}

function init_tabsort(table_id) {
   if(!document.getElementById) return;
   tables[ti] = new sort_table(table_id);
   var t = ti;
   ti += 1;
   tables[t].obj = document.getElementById(table_id);
   for(var i = 0; i < tables[t].obj.childNodes.length; i++) {
    switch(tables[t].obj.childNodes[i].nodeName.toLowerCase()) {
         case 'colgroup':
            for(j = 0;
            j < tables[t].obj.childNodes[i].childNodes.length;
            j++)
if(tables[t].obj.childNodes[i].childNodes[j].nodeName.
toLowerCase() == 'col')
               tables[t].sort_cols += 1;
            break;
         case 'tbody':
```

```
            tables[t].tbody_obj = tables[t].obj.childNodes[i];
            break;
         default: break;
      }
   }
   var tri = 0;
   var tdi = 0;
   for(i = 0; i < tables[t].tbody_obj.childNodes.length; i++) {
if(tables[t].tbody_obj.childNodes[i].nodeName.toLowerCase()
      == 'tr') {
         tables[t].tbody_obj.childNodes[i].id = 'tr_' + tri;
         tables[t].tr_ids.push('tr_' + tri);
         tables[t].sort_rows += 1;
         tdi = 0;
         for(j = 0;
         j < tables[t].tbody_obj.childNodes[i].childNodes.length; j++) {
if(tables[t].tbody_obj.childNodes[i].childNodes[j].nodeName.toLowerCase() ==
'td') {
   tables[t].tbody_obj.childNodes[i].childNodes[j].id
   = 'tr_' + tri + '_td_' + tdi;
   tables[t].td_ids.push('tr_' + tri + '_td_' + tdi);
               tdi += 1;
            }
         }
         tri += 1;
      }
   }
}

function get_sort_table_obj_by_id(id) {
   for(var i = 0; i < tables.length; i++)
      if(tables[i].id == id)
         return(tables[i]);
}

function numsort(a,b) {
   return parseFloat(a) - parseFloat(b);
}

function tabsort(table_id, sort_type, col_n) {
   if(!document.getElementById) return;
   var delimiter = "|||||";
   var to = get_sort_table_obj_by_id(table_id);
   var sort_array = new Array();
   for(var i = 0; i < to.sort_rows; i++) {
      tr_array = new Array();
      tr_id = 'tr_' + i;
      td_id = tr_id + '_td_' + col_n;
      tr_array.push(document.getElementById(td_id).innerHTML);
      for(var j = 0; j < to.sort_cols; j++) {
          if(j == col_n) continue;
          else {
             tr_id = 'tr_' + i;
```

```
            td_id = tr_id + '_td_' + j;
            tr_array.push
            (document.getElementById(td_id).innerHTML);
         }
      }
      tr_str = tr_array.join(delimiter);
      sort_array.push(tr_str);
   }
   if(sort_type == 'an')
      sort_array.sort();
   else if(sort_type == 'n')
      sort_array.sort(numsort);
   if(to.active_sort_order == 'desc') {
      sort_array.reverse();
      to.active_sort_order = 'asc';
   }
   else
      to.active_sort_order = 'desc';
   for(i = 0; i < sort_array.length; i++) {
      tr_array = sort_array[i].split(delimiter);
      first = tr_array[0];
      col = tr_array[col_n];
      tr_array[col_n] = first;
      tr_array[0] = col;
      for(j = 0; j < to.sort_cols; j++) {
         tr_id = 'tr_' + i;
         td_id = tr_id + '_td_' + j;
         document.getElementById(td_id).innerHTML = tr_array[j];
      }
   }
   var img_html;
   if(to.active_sort_order == 'asc')
      img_html = '<img src="' + desc_img_uri + '" alt="' +
      desc_img_alt_text + '">';
   else
      img_html = '<img src="' + asc_img_uri + '" alt="' +
      asc_img_alt_text + '">';
   for(i = 0; i < to.sort_cols; i++) {
      if(i == col_n)
document.getElementById(table_id + '_sort_img_' + i).innerHTML =
img_html + " ";
      else
         document.getElementById(table_id + '_sort_img_' + i).innerHTML = "";
   }
}
```

Zu Beginn des Scripts werden einige globale Variablen deklariert und initialisiert. Mit `default_order` kann festgelegt werden, ob eine Tabelle beim ersten Anklicken einer Spalte nach deren Werten zunächst aufsteigend (`asc`) oder absteigend (`desc`) sortiert werden soll. Danach wird bei jedem weiteren Klick automatisch umgeschaltet. Die beiden URIs der Grafiken, die aufsteigende und absteigende Sortierung signalisieren, lassen sich natürlich anpassen, ebenso wie die zugehörigen Alternativtexte.

Die Variable `delimiter` verdient besondere Aufmerksamkeit. Die zugewiesene Zeichenkette, im Beispiel fünf Senkrechtstriche, darf innerhalb der Tabelle nicht in den Daten vorkommen. Das Script benötigt diese Variable, um einen Tabellenzeilen-Array temporär zu einem String zusammenzusetzen und diesen anhand des Begrenzers (Delimiter) anschließend wieder in seine Einzelwerte zerlegen zu können.

Ferner wird beim Einlesen des Scripts ein Array namens `tables` angelegt sowie eine Array-Zählervariable `ti`. Alle übrigen Anweisungen des Scripts stehen innerhalb von Funktionen und werden daher erst bei Aufruf der Funktionen ausgeführt. Insgesamt kommen folgende Funktionen vor:

- `sort_table():` Diese Funktion dient zur Erstellung eines Objekts. Das Script legt für jede zu sortierende Tabelle eine Instanz dieses Objekts an und speichert es im zuvor deklarierten Array `tables`. Auf diese Weise wird, falls mehrere sortierbare Tabellen auf einer Seite vorkommen, eine saubere Datenhaltung erreicht.
- `init_tabsort():` Diese Funktion initialisiert die Sortierbarkeit einer Tabelle. Aufgerufen werden muss sie, bevor der Anwender einen Sortiervorgang anstößt. Der richtige Ort, um diese Funktion aufzurufen, ist der Event-Handler `onload=` im einleitenden `<body>`-Tag. Für jede sortierbare Tabelle auf einer Seite muss diese Funktion aufgerufen werden. Dabei muss ihr der id-Name der zu sortierenden Tabelle als Parameter übergeben werden.
- `get_sort_table_obj_by_id():` Dies ist eine Hilfsfunktion des Script, mit deren Hilfe eine Anweisung schnell ein im Array `tables` gespeichertes Objekt des Typs `sort_table` auffinden kann.
- `numsort():` Diese Hilfsfunktion wird benötigt, um bei Anwendung der sort()-Methode des Array-Objekts eine numerische Sortierung zu ermöglichen.
- `tabsort():` Diese Funktion führt die eigentliche Sortierung durch. Sie wird über den Event-Handler `onclick=` beim Anklicken der Buttons in den Tabellenüberschriften aufgerufen.

Die beiden »großen« Funktionen sind also `init_tabsort()` und `tabsort()`. In `init_tabsort()` wird, nachdem alle nicht-DOM-fähigen Browser wieder heimgeschickt wurden, in der Array-Variablen `tables[ti]` ein neues Objekt des Typs `sort_table` gespeichert. Dazu wird die Objektfunktion `sort_table()` aufgerufen.

Das sort_table-Objekt hat unter anderem eine Eigenschaft `obj`. In dieser wird das DOM-Objekt der gesamten Tabelle gespeichert, indem `document.getElementById(table_id)` an `tables[t].obj` zugewiesen wird. Über `tables[t].obj` ist dadurch innerhalb der Funktion fortan der DOM-Zugriff auf die Tabelle möglich.

Im weiteren Verlauf analysiert `init_tabsort()` die Tabelle. Nun wird auch klar, warum die Tabelle in HTML ein paar Konventionen einhalten muss, die bereits beschrieben wurden. Das colgroup-Element wird gesucht und die Anzahl der Spalten wird ermittelt. Der gesamte Tabellenkörper, identifizierbar am tbody-Element, wird als Objekt in der Objekteigenschaft `tables[t].tbody_obj` gespeichert.

In der größeren, verschachtelten for-Schleife im zweiten Teil von `init_tabsort()` passiert Folgendes: jedes tr- und jedes td-Element erhält einen eindeutigen id-Namen, damit später ein möglichst einfacher Zugriff darauf möglich ist. Die id-Namensvergabe folgt dabei einem bestimmten Schema. Betrachten wir folgenden HTML-Ausschnitt:

```
<tr>
<td>Buttermilch</td>
<td>Milch und Milchprodukte</td>
<td class="fp">1</td>
</tr>
<tr>
<td>Fruchtjoghurt 1,5% 150g</td>
<td>Milch und Milchprodukte</td>
<td class="fp">2</td>
</tr>
```

Daraus macht `init_tabsort()` im Arbeitsspeicher folgenden Code:

```
<tr id="tr_0">
<td id="tr_0_td_0">Buttermilch</td>
<td id="tr_0_td_1">Milch und Milchprodukte</td>
<td id="tr_0_td_2" class="fp">1</td>
</tr>
<tr id="tr_1">
<td id="tr_1_td_0">Fruchtjoghurt 1,5% 150g</td>
<td id="tr_1_td_1">Milch und Milchprodukte</td>
<td id="tr_1_td_2" class="fp">2</td>
</tr>
```

Die tr-Elemente des sortierbaren Tabellenkörpers werden also mit `tr_0`, `tr_1`, `tr_2` usw. »nummeriert«. Innerhalb einer Tabellenzeile erhalten alle td-Elemente den tr-Namen als vorderen id-Namensteil. Angehängt wird dann für jede Spalte `_td_0` für die erste Spalte, `_td_1` für die zweite Spalte, `_td_2` für die dritte usw. Der anfängliche DOM-Zugriff auf die Tabellenzeilen und -zellen ist etwas komplex. Greifen wir zum besseren Verständnis die folgende if-Abfrage heraus:

```
if(tables[t].tbody_obj.childNodes[i].childNodes[j].nodeName.toLowerCase() ==
'td')
```

Wie weiter oben bereits erwähnt, ist in `tables[t].tbody_obj` der tbody-Elementknoten der sortierbaren Tabelle gespeichert. Über `tables[t].tbody_obj.childNodes` ist der DOM-Zugriff auf alle Kindknoten dieses Elementknotens möglich. Dazu gehören neben anderen Knoten auch die Elementknoten der inneren tr-Elemente. In der herausgegriffenen if-Abfrage, die innerhalb einer for-Schleife steht, ist in `childNodes[i]` ein tr-Element ermittelt worden. Die Abfrage sucht jedoch nach td-Elementen. Also muss sie noch weiter in der Hierarchie herabsteigen und abermals nach `childNodes` suchen, diesmal nach denen des tr-Elements. Die if-Abfrage steht nicht nur innerhalb einer for-Schleife, sondern innerhalb einer zweiten. In `childNodes[j]` (`j` ist der Schleifenzähler der inneren for-Schleife, `i` derjenige der äußeren!) ist ein aktueller Kindknoten des tr-Elements (also des Elementknotens

tbody_obj.childNodes[i]) gespeichert. Über die Objekteigenschaft nodeName wird der Knotenname ermittelt. Durch Anwendung von toLowerCase() wird dieser Wert klein geschrieben interpretiert. Wenn das Kindobjekt td heißt, dann ist die if-Bedingung erfüllt.

Nachdem die Funktion init_tabsort() beendet wurde, kann die betroffene Tabelle nach jeder ihrer Spalten auf- und absteigend, alphanumerisch oder numerisch sortiert werden. Angestoßen wird die Sortierung im Beispiel wie schon angedeutet über onclick-Event-Handler in den button-Elementen der Spaltenüberschriften. Den onclick-Event-Handlern wird dabei als Wert ein Aufruf der Funktion tabsort() zugewiesen. Die Funktion erwartet drei Parameter. Als erster Parameter muss der id-Name der Tabelle übergeben werden – in unserem Beispiel lautet er fettpunkttabelle. Als zweiter Parameter wird entweder an oder n übergeben. Mit an wird eine alphanumerische Sortierung angewiesen und mit n eine numerische. In unserem Beispiel enthält die dritte Spalte nur Zahlenwerte. Damit diese wie gewünscht sortiert werden, wird für diese Spalte eine numerische Sortierung angewiesen. Die beiden anderen Spalten enthalten Zeichenketten als Werte, daher wird für sie an übergeben. Als dritter Parameter muss schließlich noch die Indexnummer der zu sortierenden Spalte übergeben werden. Bei der ersten Spalte ist dies 0, bei der zweiten 1, bei der dritten 2 usw.

In tabsort() wird über die Funktion get_sort_table_obj_by_id() das im Array tables gespeicherte sort_table-Objekt der betroffenen Tabelle geholt und in der Variablen to gespeichert. Über to ist damit der Zugriff auf das sort_table-Objekt der Tabelle und damit auch auf die darin gespeicherten Objekte der Tabelle selbst möglich.

Als Nächstes wird ein Array namens sort_array erzeugt. Was innerhalb der anschließenden for-Schleife passiert, ist Folgendes: Für jede Tabellenzeile des sortierbaren Tabellenkörpers werden die einzelnen Werte aus den Zellen dieser Zeile geholt. Aus den Werten wird eine Zeichenkette zusammengesetzt, die in der Variablen tr_str gespeichert wird. Zwischen den Werten der Zellinhalte wird jeweils die Zeichenfolge eingefügt, die in der globalen Variable delimiter gespeichert ist. Das ist auch der Grund, warum diese Zeichenfolge in den Tabellendaten selbst nicht vorkommen darf.

Angenommen, folgende HTML-Tabellenzeile wird verarbeitet:

```
<tr>
<td>Buttermilch</td>
<td>Milch und Milchprodukte</td>
<td class="fp">1</td>
</tr>
```

Daraus wird nach dem oben Beschriebenen in unserem Beispiel folgende Zeichenkette:

```
Buttermilch|||||Milch und Milchprodukte|||||1
```

Nun soll die so erzeugte Zeichenkette den Zweck erfüllen, in sort_array als Wert übernommen zu werden. Am Ende steht jede Tabellenzeile in dieser Form in sort_array. Durch Aufruf der sort()-Methode für Arrays werden die Werte sortiert. Wenn nun beispielsweise nach der dritten Spalte sortiert werden soll, müssen deren Werte aber vorne stehen, damit

die Sortiermethode greift. Deshalb wird in `tr_str` der Tabellenzeileninhalt nicht einfach nur »der Reihe nach« gespeichert, sondern in der Reihe, die für die gewünschte Sortierung tauglich ist. Wird also beispielsweise eine Sortierung nach Spalte 3 gewünscht, wird Folgendes in `tr_str` erzeugt:

```
1|||||Buttermilch|||||Milch und Milchprodukte
```

Im Anschluss an die for-Schleife wird innerhalb von `tabsort()` der so erzeugte Array `sort_array()` durch Aufruf der Standard-Array-Methode `sort()` sortiert. Falls eine absteigende Sortierung gewünscht ist, muss außerdem noch `reverse()` angewendet werden, was die Elementreihenfolge im Array einfach umkehrt und ebenfalls eine Standard-Array-Methode ist. Falls eine numerische Sortierung gewünscht ist, wird der sort()-Methode der Name der Hilfsfunktion `numsort` übergeben. Diese erzwingt eine Interpretation von zwei zu vergleichenden Werten als Fließkommazahl und bedient die sort()-Methode so, wie sie es benötigt.

Nach erfolgreicher Sortierung muss `sort_array` nun wieder verarbeitet werden. Jedes Array-Element stellt ja eine Tabellenzeile dar, mittlerweile jedoch sortiert. In einer for-Schleife wird der Array abgearbeitet Mithilfe der split()-Methode wird die aktuelle Zeile anhand der Delimiter-Zeichenfolge wieder in Array-Elemente zerlegt (`tr_array`). Die Werte können nun direkt in die Tabelle geschrieben werden. Dabei kann bequem über die von der Funktion `tabsort_init()` erzeugten id-Namen auf die einzelnen Tabellenzellen zugegriffen werden. Jede Tabellenzelle erhält mit `innerHTML` einen neuen Wert, der sich durch die Sortierung ergibt.

Fazit

Das Beispiel ist nur eines dafür, was bei HTML-Tabellen in Verbindung mit DOM und JavaScript möglich ist. Tabellensortierung bringt dem Anwender einen eindeutigen informativen Mehrwert. Mit etwas Phantasie ist mit den gleichen Bordmitteln jedoch noch viel mehr möglich. Angenommen, jede Tabellenzelle besteht aus einem input-Eingabefeld (dieses kann ja mit CSS (`border:none`) rahmenlos gemacht werden, damit es nicht so »formularmäßig« aussieht). Auf diese Weise entsteht eine vollständig editierbare Tabelle. Mit etwas JavaScript/DOM können Sie dann Ansätze einer richtigen Tabellenkalkulation auf Ihrer Webseite realisieren.

Teil 4 – Die Server-Seite

HTML, CSS und JavaScript/DOM sind Basistechnologien zum Erstellen von Webseiten. Doch zum Profiwissen gehört auch, zu wissen, wie eigentlich Webseiten vom Server zum Browser gelangen, was dabei genau geschieht und welche Möglichkeiten es gibt, Vorgänge zu optimieren oder auftretende Probleme zu beseitigen. Dieser Buchabschnitt beschreibt also den Schritt vom Webdesigner zum Webmaster.

9 Hosting und Webserver

Wenn Sie als Anwender im Web eine Seite aufrufen, startet Ihr Browser eine Anfrage an einen entsprechenden Webserver. Welcher Webserver auf Grund einer aufgerufenen oder angeklickten Adresse die Anfrage erhält, welche Protokolle und Stationen dahinter stecken, vermittelt das vorliegende Kapitel ebenso wie Grundkenntnisse in der Konfiguration eines Webservers.

Neben konkreten technischen Informationen behandelt das Kapitel aber auch Themen wie Web-Hosting, die eher zum organisatorisch-praktischen Teil des Betriebs von Websites gehören.

9.1 Routing, IP-Adressen und Domain-Namen

Webentwickler müssen keine Netzwerkspezialisten sein. Doch Basiskenntnisse darüber, wie Anfragen im Internet vom anfragenden Rechner zum angefragten Rechner gelangen und welche Rolle dabei IP-Adressen und Domain-Namen spielen, sind unerlässlich. Dieser Abschnitt vermittelt das nötige Hintergrundwissen in lockerer Form, ohne dabei auf Details der Netzwerktechnik einzugehen.

9.1.1 Szenario eines Routings

Gehen wir von dem Szenario aus, dass ein Anwender in der Adresszeile seines Browsers eine Adresse eingibt:

www.example.org

Das ist zunächst einmal noch kein vollständiger URI. Es fehlen zumindest die Schemaangabe zum Protokoll sowie die gewünschte Pfadangabe zum verlangten Inhalt. Da ein Webbrowser per Default das HTTP-Protokoll verwendet und annimmt, dass die Startseite im Startverzeichnis des Webangebots verlangt wird, ist er so freundlich, die Adresse selbstständig in ein korrektes Standardformat zu bringen:

http://www.example.org/

http:// ist dabei das Schema für das HTTP-Protokoll und / am Ende ist der Pfadname des Web-Wurzelverzeichnisses.

Damit die aktuelle Version der verlangten Seite aufrufbar ist, muss eine Internetverbindung bestehen. Die Internetverbindung ist eine Verbindung zu einem »Einwahlknoten«

eines Zugangsproviders. Dabei spielt es keine Rolle, um welche Art von Internetverbindung es sich handelt. Das Wort »Einwahlknoten« steht jedoch bewusst in Anführungszeichen. Eine Einwahl im engeren Sinne erfolgt nur dort, wo ein Modem am Werk ist, das nicht ständig online ist. Bei echten Standleitungen etwa gibt es keine Einwahl. Dennoch führt auch die Standleitung erst einmal zu einem Provider oder zumindest direkt zu einem so genannten Backbone-Provider. Von dort aus sorgt das Routing im Internet dafür, dass die Anfrage an ihr Ziel gelangt.

Routing

Betrachten wir, um den Vorgang besser zu verstehen, die Ausgabe des Kommandos `tracert`, welches auf praktisch allen internetfähigen Systemen verfügbar ist. Unter MS Windows können Sie es in einem DOS-Fenster oder unter einer Zusatz-Shell wie cygwin ausprobieren.

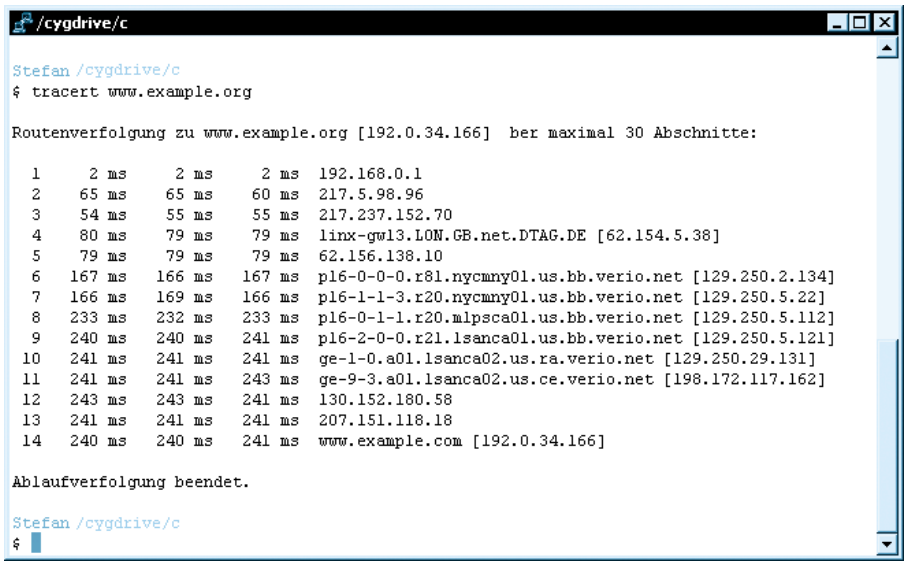


Abbildung 9.1: tracert-Kommando zu www.example.org

In diesem Beispiel wurde eingegeben:

```
tracert www.example.org
```

Hinter dem Kommandonamen `tracert` können Sie einen Domain-Namen oder eine IP-Adresse angeben.

Nachdem die Routenverfolgung abgeschlossen ist, listet das tracert-Kommando sein Ergebnis auf. In der letzten Spalte stehen die IP-Adressen und/oder Hostnamen der Stationen im Netz, welche die Anfrage an die Adresse passiert hat. In der ersten Spalte stehen die Laufnummern und dazwischen die Reaktionszeiten bei der Kommunikation zwischen den beteiligten Rechnern.

In unserm Beispiel verbirgt sich hinter der IP-Adresse in Laufnummer 1 ein *Router*, über den ein lokales Netzwerk (LAN) ans Internet angeschlossen ist. Im Beispiel wird die Internetverbindung letztlich über einen DSL-Anschluss realisiert, doch bei anderen Anschlussarten sieht der Netzzugang auch nicht anders aus. Falls sich ein Einzel-PC über Modem direkt beim Provider einwählt und kein LAN dazwischen ist, wird bei Laufnummer 1 die IP-Adresse des eingewählten PCs angezeigt. Diese wird heutzutage von den meisten Providern dynamisch bei jeder Einwahl neu zugewiesen, d.h., es kann bei jeder Einwahl eine andere IP-Adresse sein.

Die Adresse bei Laufnummer 2 gehört bereits zum Provider. Es handelt sich um den Rechner, den der Provider für die Internetverbindung als Einwahlrechner bereitgestellt hat. Im Beispiel passiert der Weg einen weiteren Rechner im gleichen Subnetz (217 an erster Stelle), bevor er in Laufnummer 4 den Gateway-Rechner erreicht, über den der Provider an den Backbone angeschlossen ist (auf die Begriffe Gateway und Backbone kommen wir später noch zu sprechen).

Der Ausgabe des tracert-Kommandos ist zu entnehmen, dass sich der mit *www.example.org* angefragte Rechner in den USA befindet. Die Adresse in Laufnummer 5 ist offensichtlich die letzte, die noch zum europäischen Backbone gehört. Danach passiert die Anfrage bereits diverse Gateway-Adressen des Backbone-Providers *verio.net* in den USA, bevor sie von dort schließlich ihren Zielrechner erreicht.

Auf dem Zielrechner erhält der Webserver die Anfrage des Browsers und sendet über das HTTP-Protokoll eine Antwort. Sofern erforderlich, werden anschließend die gewünschten Daten übertragen, etwa die Inhalte von HTML-Dateien sowie darin referenzierte Stylesheets, JavaScript-Dateien, Grafiken usw. Diese Daten müssen ebenfalls wieder über diverse Stationen geroutet werden, bis sie beim anfragenden Rechner bzw. beim dort auf Antwort wartenden Browser ankommen.

9.1.2 Gateways und Routing-Tabellen

Dem Beispiel aus dem vorherigen Abschnitt ist zu entnehmen, dass es an mehreren Stellen Rechner im Netz gibt, deren Hauptaufgabe offensichtlich darin besteht, Anfragen und Serverantworten weiterzuleiten. Solche Rechner sind Knotenpunkte im Netz oder, um im Eisenbahn-Jargon zu sprechen, Bahnhöfe mit Stellwerk.

Router und Gateways

Wenn ein PC direkt über Modem bei einem Internet Provider eingewählt wird, benötigt der PC keine eigene Intelligenz zum Routen. Doch bereits ein Router, welcher ein lokales Netzwerk ans Internet anbindet, heißt nicht zufällig so. Seine Aufgabe besteht darin, zuzuordnen, von welchem ihm bekannten Rechner im lokalen Netz eine Anfrage kommt und für welchen dieser LAN-Rechner hereinkommende Antwortdaten aus dem Internet bestimmt sind. Das Zuordnen und entsprechende Weiterleiten wird als *Routing* bezeichnet. Rechner, die solche Routing-Aufgaben übernehmen, werden als *Gateway*-Rechner bezeichnet.

Die »großen« Gateways im Internet tun auch nichts anderes als ein Router, welcher ein LAN mit dem Internet verbindet. Sie verbinden ein ihnen bekanntes Subnetz mit einem übergeordneten oder anderen Subnetz. Das bedeutet, sie können zuordnen, von welchem ihnen bekannten Host-Rechner (genauer: von welcher ihnen bekannten IP-Adresse) eine Anfrage kommt. Sie können sie weiterleiten an ein anderes Subnetz und sie können Antworten oder auch Anfragen, die von anderen Subnetzen kommen, an die richtigen Rechner bzw. IP-Adressen in ihrem Subnetz weiterleiten.

Routing-Tabellen

Damit Rechner mit Gateway-Funktion korrekt routen, benötigen sie Routing-Tabellen. Im Fall eines LAN-Routers, der einfach nur mit einem Internet-Zugangsprovider verbunden ist, muss dieser nach außen hin nur eine einzige Route kennen, nämlich die zum Provider. Zentrale Gateways im Internet kennen jedoch zahlreiche Routen. Ihre Aufgabe besteht darin, möglichst alle Datenpakete, die sie weiterleiten, auf möglichst günstige und schnelle Routen zu schicken. Dazu müssen sie die einzelnen Datenpakete analysieren.

Wie und wo eine Routing-Tabelle gespeichert ist und über welche Kommandos oder Programme sie editierbar ist, ist systemabhängig. Werfen wir einen Blick auf eine typische Routing-Tabelle, um die Funktionsweise zu verstehen:

```
Destination   Gateway         Netmask          Interface
127.0.0.1     127.0.0.1       255.0.0.0        192.168.100.105
0.0.0.0       149.211.17.212  0.0.0.0          eth0
149.211.40.0  149.211.17.212  255.255.255.0    eth0
149.211.32.0  149.211.17.212  255.255.255.0    eth0
149.211.8.0   149.211.17.213  255.255.255.0    eth0
149.211.17.0  149.211.17.213  255.255.255.0    eth0
149.211.1.0   149.211.17.214  255.255.255.0    eth0
149.211.18.0  149.211.17.215  255.255.255.0    eth0
```

Jede Zeile stellt die Definition einer Route dar. Die erste Spalte enthält mögliche Ziel-IP-Adressen (`Destination`) einer Route, also die Adressen anderer »Bahnhöfe mit Stellwerk« (Gateways oder Router). Die Adressen `127.0.0.1` und `0.0.0.0` haben Sonderbedeutung. `127.0.0.1` ist eine so genannte ***Loopback-Adresse***. Das ist eine Adresse, die das eigene System meint, also nicht woanders hin geroutet werden muss. Die Adresse `0.0.0.0` bezeichnet das Standard-Gateway bzw. die Default-Route, die gewählt wird, wenn keine Route bevorzugt wird.

Die zweite Spalte, im Beispiel mit `Gateway` überschrieben, enthält die IP-Adresse des Routers, über den zur Zieladresse der ersten Spalte durch geroutet werden kann.

Die Spalte, die im Beispiel mit ***Netmask*** überschrieben ist, bezeichnet die so genannte ***Subnetzmaske***. Die Subnetzmaske definiert, wie groß der IP-Adressraum des vom jeweiligen Gateway verwalteten Subnetzes ist. Eine typische Subnetzmaske wie 255.255.255.0 bedeutet, dass das Gateway nur die letzte Zahl der IP-Adresse verwaltet, mithin also maximal 255 eigene Unteradressen.

Die Schnittstelle schließlich (`Interface`) ist das Gerät oder die IP-Adresse des Geräts, über das die unter der Spalte `Gateway` genannte IP-Adresse erreichbar ist. Bei der Loopback-Adresse ist das z. B. die IP-Adresse des eigenen Rechners. Andere Gateways sind z. B. über LAN erreichbar, also über Ethernet-Schnittstellen (im Beispiel: `eth0` für die erste LAN-Schnittstelle).

Unter MS Windows können Sie sich übrigens durch das DOS-Kommando `route print` die auf dem eigenen PC gespeicherte Routing-Tabelle anzeigen lassen. Unter Linux und den meisten Unix-basierten Systemen genügt das Shell-Kommando `route` zur Ausgabe der Tabelle.

9.1.3 TCP/IP, IP-Adressen und Internetanbindung

In den vorangegangenen Abschnitten war schon vielfach von IP-Adressen die Rede. Die IP-Adressierung gehört zum IP-Protokoll (IP steht für ***Internet Protocol***). Dieses besorgt in Verbindung mit den TCP-Protokoll den gesamten Datenverkehr im modernen Internet (TCP steht für ***Transmission Control Protocol***). Auch innerhalb lokaler Netze wird immer häufiger nur noch TCP/IP eingesetzt.

Eine heute noch übliche IP-Adresse besteht aus vier Zahlen zwischen 0 und 255 (darstellbar durch 1 Byte oder 8 Bit), die in üblicher Schreibweise durch Punkte getrennt werden. Die Zahlen haben ähnlich wie Vorwahl- und Durchwahlnummern in der Telefonie die Aufgabe, von links her betrachtet Großnetze bis hin zu einzelnen ans Internet angeschlossenen Geräten (Hosts) zu adressieren.

Klasse-A-, Klasse-B- und Klasse-C-Netze

Die größten Netze sind so genannte Klasse-A-Netze. Bei solchen Netzen adressiert die erste Zahl das Netz (sie darf zwischen 1 und 126 liegen). Solche Netze werden vor allem von Backbone-Providern betrieben.

Die drei restlichen Zahlen können innerhalb eines Klasse-A-Netzes frei bestimmt werden – was aber in der Praxis nicht bedeutet, dass ein entsprechender Netzbetreiber alle 16,7 Millionen IP-Adressen verwaltet, die auf diese Weise innerhalb seines Klasse-A-Netzes adressierbar sind.

Neben den Klasse-A-Netzen gibt es auch Klasse-B- und Klasse-C-Netze. Bei Klasse-B-Netzen wird das Netz durch die beiden ersten Zahlen adressiert und die beiden letzten sind »frei bestimmbar«. Bei Klasse-C-Netzen adressieren die ersten drei Zahlen das Netz und nur die letzte ist frei bestimmbar.

Hosts

Jeder Rechner oder ein Gerät, das mit einer eigenen Internetadresse direkt ans Internet angeschlossen ist, wird als ***Host*** bezeichnet. Wenn sich ein PC über Modem direkt bei einem Zugangs-Provider einwählt, bekommt er von diesem eine IP-Adresse zugewiesen. Damit ist der PC selber ein Internet-Hostrechner. In einem LAN dagegen, bei dem alle

Internetzugänge über einen Router laufen, ist nur der Router der Host. Die PCs dahinter sind dagegen nach außen hin gar nicht sichtbar.

Das ist auch der Grund, warum heute vielfach empfohlen wird, in jedem Fall eine LAN-/Router-Lösung als Internetzugang zu wählen, selbst wenn sich nur ein Rechner hinter dem Router befindet. Zahlreiche Späherprogramme, Spam-Bots und Ähnliches sind nämlich stets auf der Suche nach schlecht konfigurierten Hosts, die dann für Datenverkehr missbraucht werden, den der Anwender gar nicht wünscht.

Dynamische und feste IP-Adressen

Zwar lassen sich mithilfe der vier Zahlen des heutigen IP-Adress-Systems bis zu 4 Milliarden Hosts adressieren. Doch in der Praxis wird ein Großteil der möglichen IP-Adressen verschenkt und es herrscht Adressenknappheit. Eine Folge dieser Knappheit ist, dass fast alle Internet-Zugangsprovider IP-Adressen bei der Einwahl dynamisch vergeben. Viele Hosts haben daher keine feste, immer gleiche IP-Adresse, sondern erhalten bei jeder Neueinwahl eine andere Adresse. Nur bei Hosts, die über eine echte Standleitung angebunden sind, bleibt die IP-Adresse gleich. Deshalb müssen Host-Rechner, welche beispielsweise Websites beherbergen sollen, möglichst über Standleitung ans Netz angebunden sein. Denn dabei geht es nicht nur um ausreichende Bandbreite für parallele Datenübertragungen, sondern auch darum, dass die Zuordnung zwischen einer IP-Adresse und einem Domain-Namen stabil bleibt.

Für die meisten Privatanwender und selbst für viele kleine Unternehmen sind echte Standleitungen als Internetzugang zu kostspielig. Meistens erfüllen auch die immer häufigeren DSL-Anschlüsse die Zugangsbedürfnisse. Doch selbst die so genannten DSL-Flatrates sind keine echten Standleitungen. Das DSL-Modem trennt bei solchen Zugangsarten die Verbindung, wenn keine Daten übertragen werden, und selbst wenn durchweg Daten übertragen werden, erzwingt der Zugangsprovider wenigstens einmal täglich eine Verbindungstrennung, damit ein Verbindungsneuaufbau mit dynamischer Zuweisung einer anderen IP-Adresse möglich wird. Aus diesem Grund ist es auch nicht sinnvoll möglich, auf einem eigenen DSL-Routing-Rechner eine eigene Domain zu betreiben. Das Web-Hosting besorgen stattdessen immer noch mehrheitlich Provider, die sich darauf spezialisiert haben. Solche »Hosting-Provider« heißen nicht zuletzt deshalb so, weil sie über Hosts verfügen, die mit einer festen IP-Adresse mit dem Internet verbunden sind und daher Domain-Adressierung ermöglichen.

9.1.4 IP-Adressen und Domain-Namen

Neben dem numerischen Adressraum, der sich durch die IP-Adressen ergibt, hat in den letzten Jahren ein zweiter Adressraum zunehmend an Bedeutung gewonnen: der Namensraum, welcher aus den so genannten Domain-Namen besteht. Dabei handelt es sich nicht um Adressen anderer Hosts, sondern lediglich um eine andere Adressierungsform. Ein einfaches Beispiel verdeutlicht dies. Ein ping-Kommando auf die Adresse www.example.org liefert die IP-Adresse 192.0.34.166. Und tatsächlich liefert auch der

Webbrowser das gleiche Ergebnis, ob man nun *http://www.example.org/* aufruft oder *http://192.0.34.166/*.

Domain Name Service (DNS)

Wenn eine Internetanwendung wie ein Webbrowser eine Domain-Adresse aufrufen soll, muss er den Domain-Namen in eine IP-Adresse auflösen. Zunächst sieht er in der lokalen Hostnamendatei des internetfähigen PCs nach, ob dort eine Zuordnung zwischen dem verlangten Domain-Namen und einer IP-Adresse zu finden ist.

Unter neueren MS Windows-Versionen ist die entsprechende Datei mit dem Namen *hosts* unter *[Windows-Verzeichnis]\system32\drivers\etc* zu finden. Sie kann mit einem Texteditor aufgerufen und bearbeitet werden. Auf unixbasierten Systemen wie Linux findet sich die Datei *hosts* meistens unter */etc*.

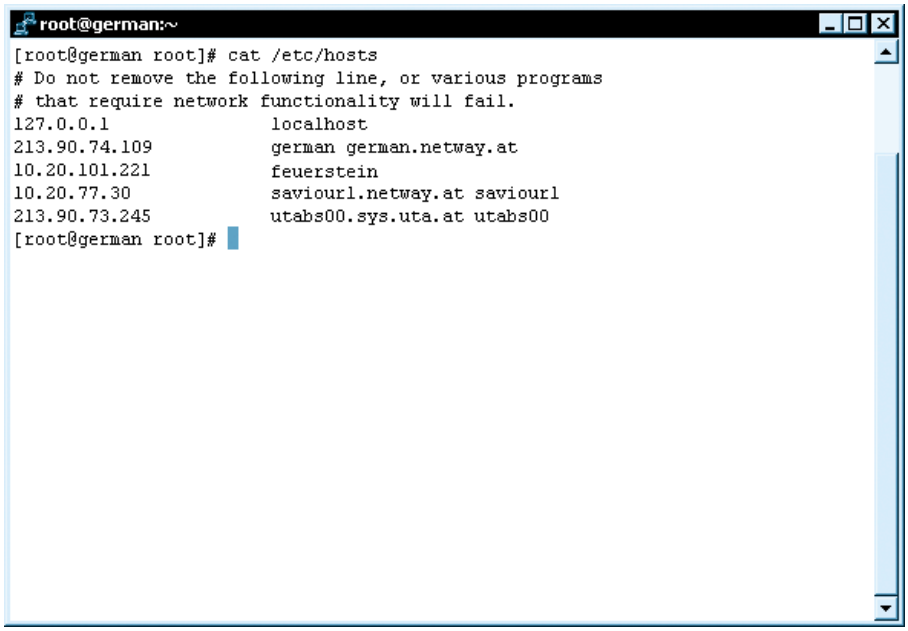


Abbildung 9.2: Typischer Inhalt einer hosts-Datei

Die Datei *hosts* enthält Zuordnungen zwischen IP-Adressen und Domain-Namen. Zumindest eine Zuordnung ist praktisch immer eingetragen und sollte auch eingetragen bleiben: die zwischen der Loopback-Adresse `127.0.0.1` und dem Domain-Namen `localhost`.

Alles, was der Browser in der lokalen *hosts*-Datei findet, kann er selbst auflösen. Für alles, was er dort nicht findet, muss er jedoch einen fremden Service in Anspruch nehmen – den Domain Name Service.

Dazu gibt es Server, die Informationen über die Zuordnung von IP-Adressen zu Domain-Namen auf Wunsch »servieren«. Internet-Zugangsprovider bieten solche DNS-Server an.

Deren IP-Adresse wird normalerweise in den Zugangsdaten der Internetverbindung gleich mit konfiguriert. Der Browser kann dann über eine so konfigurierte Internetverbindung den DNS-Server des Providers befragen.

Das Namensystem

Domain-Namen wie *localhost* oder *feuerstein* in der obigen Abbildung sind durchaus zulässig und üblich – solange es sich um Einträge in der eigenen lokalen *hosts*-Datei handelt. Für die »offiziellen«, öffentlich zugänglichen Domain-Namen gelten dagegen bestimmte Regeln.

Solche Domain-Namen sind hierarchisch aufgebaut. Die Hierarchien werden ähnlich wie bei IP-Adressen durch Punkte getrennt. Der letzte Namensteil ist der allgemeinste, nämlich die so genannte *Top-Level-Domain*. Jedes Land hat eine Top-Level-Domain, z. B. Deutschland die Zeichenfolge *de*, Österreich die Zeichenfolge *at*, Estland die Zeichenfolge *ee* und Djibuti die Zeichenfolge *dj*. Neben den landesspezifischen Toplevel-Domains gibt es diverse Top-Level-Domains für Betreibertypen wie *com* (kommerziell), *net* (für Netzbetreiber im Internet), *org* (für Vereine und vergleichbare Organisationen), neuerdings aber auch solche wie *biz* (für Unternehmen), *info* (für Informationsdienste) oder *name* (für Privatpersonen).

Links von der Top-Level-Domain steht der inhabertypische Domain-Namensteil, also etwa der Firmenname, der Vereinsname, der Projektname usw. So ergeben sich Domain-Namen wie *selfhtml.org* oder *pearson.com*.

Weiter links können weitere Namensteile stehen. Dabei handelt es sich um so genannte *Subdomains*, die der Domain-Inhaber frei vergeben kann. Subdomains können verschiedene Aufgaben haben. So können sie auf einem Rechner laufende, unterschiedliche Internetdienste wie *www* oder *mail* bezeichnen, aber auch auf Unterverzeichnisse im Webangebot verweisen, wie etwa *de.selfhtml.org*. Sogar mehrschichtige Subdomains wie *de.aktuell.selfhtml.org* sind problemlos möglich.

Seit einiger Zeit dürfen Domain-Namen auch Zeichen enthalten, die nicht zum ASCII-Zeichensatz gehören, also etwa deutsche Umlaute. Allerdings kann die Eingabe eines solchen Namens Probleme bereiten, wenn z. B. eine Tastatur nicht entsprechend belegt ist.

Domain-Vergabestellen

Großes Gerangel herrscht um die betreibertypischen Namen. Es kann nun nur einmal *schmidt.de* oder *internet.com* geben, da alle im Internet öffentlichen Domain-Namen internetweit eindeutig sein müssen. Vor diesem Hintergrund sind Abzockergeschäfte wie Domain-Namen-Handel entstanden und in zahlreichen Gerichtsverfahren wurde und wird noch immer darum gestritten, wem ein gleichlautender Name zusteht.

Für die begehrte Vergabe der betreibertypischen Namensteile sind die Verwaltungsstellen der jeweiligen Toplevel-Domains zuständig. So werden die Adressen der deutschen Toplevel-Domain *de* vom Denic (*http://www.denic.de/*) verwaltet. Vergabestellen dieser Art bieten in der Regel auch einen so genannten *Whois-Service* an. Darüber kann nach

Domain-Namen gesucht werden, z. B. um zu prüfen, ob ein gewünschter Domain-Name noch frei ist oder an wen er bereits vergeben wurde.

Wenn Sie ein eigenes Webangebot bereitstellen möchten und dafür – was heute üblich ist – einen eigenen Domain-Namen wünschen, übernimmt Ihr Hosting-Provider die Anmeldeformalitäten für Sie. Er kann jedoch auch nur Namen reservieren, die noch frei sind. Daher sollten Sie vorher über einen whois-Service suchen, welche brauchbaren Namen noch frei sind.

Weiterführende Links:

Aufbau des Internets:
http://www.netplanet.org/aufbau/
Grundlagen Computernetze:
http://www.netzmafia.de/skripten/netze/
Whois-Abfrage für *.aero*, *.arpa*, *.biz*, *.com*, *.coop*, *.edu*, *.info*, *.int*, *.museum*, *.net* und *.org*:
http://www.internic.org/whois.html
Whois-Abfrage für *.de*:
http://www.denic.de/
Whois-Abfrage für *.at*:
http://www.atnic.de/
Whois-Abfrage für *.ch*:
http://www.switch.ch/id/
Universal Whois Service
http://www.uwhois.com/domains.html

9.2 Web-Hosting

Wer mit technischer Sorgfalt professionelle Websites erstellt, benötigt auch ein seriöses Website-Hosting. Dafür gibt es Hosting-Provider. Einige Internet-Zugangsprovider versuchen sich auch im Hosting-Geschäft und bieten Produkte aus einer Hand an, also Internetzugang plus Hosting von Webseiten. In vielen Fällen besteht jedoch bereits ein Internetzugang und es geht nur darum, eine eigene Site ins Netz zu bringen. In diesem Fall genügt ein reiner Hosting-Provider, auch »Hoster« genannt.

Jeder seriöse Hosting-Provider bietet auf seinen eigenen Webseiten detaillierte Informationen zu den von ihm angebotenen »Hosting-Paketen«. In diesem Abschnitt sollen ein paar wichtige Begriffe geklärt werden, die bei solchen Angeboten immer wieder auftauchen.

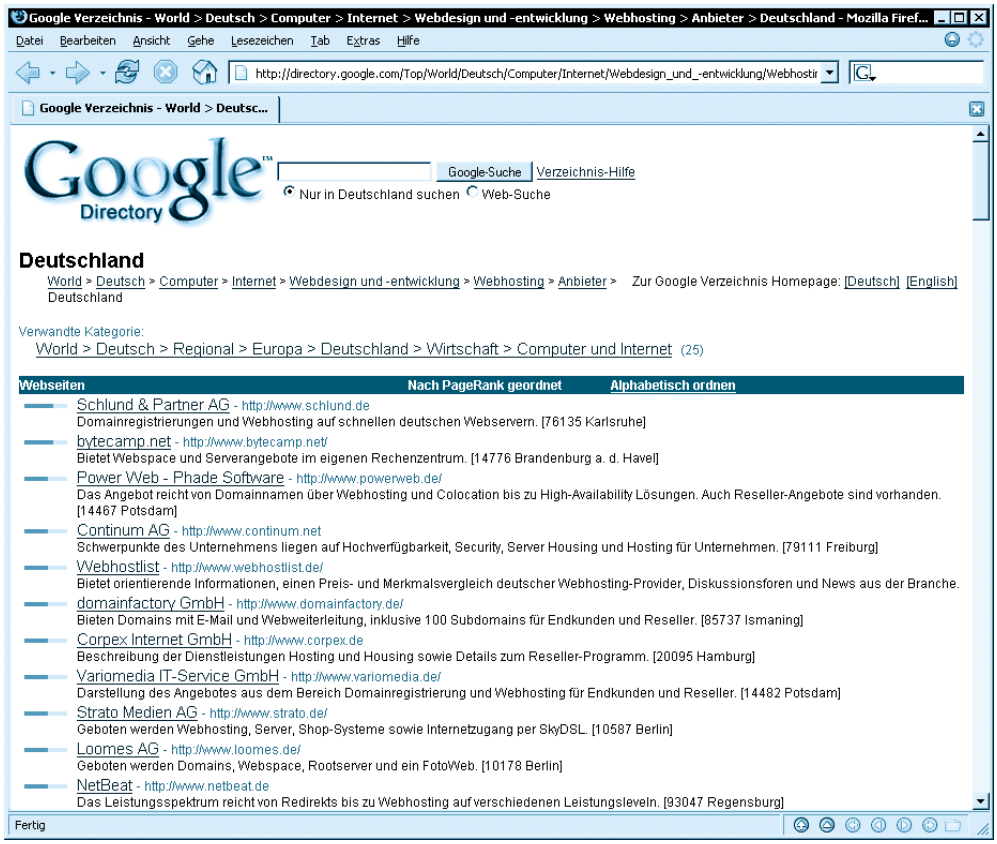


Abbildung 9.3: Eine mögliche Anlaufstelle für die Provider-Suche: das Google-Verzeichnis

Links für die Providersuche:

Webhostlist:
http://www.webhostlist.de/
Google-Liste von Webhosting-Anbietern in Deutschland, Österreich und der Schweiz:
http://directory.google.com/Top/World/Deutsch/Computer/Internet/Webdesign_und_-entwicklung/Webhosting/Anbieter/
Heise Internettarife Web-Hosting:
http://www.heise.de/itarif/view.shtml?function=hosting

9.2.1 Verhältnismäßigkeit von Angeboten und eigenen Zielen

Beim Vergleich von Hosting-Angeboten fällt es oft schwer, die eigene Website richtig einzuschätzen. Am realistischsten ist eine solche Einschätzung dann, wenn die Website bereits online ist und sich ermitteln lässt, welche Server-Belastung die Site darstellt. Solange eine Site jedoch vorher noch nicht online war, hilft nur ungefähres Kalkulieren, welche Kapazitäten sie benötigt und welches Hosting-Angebot demzufolge das beste Preis-Leistungs-Verhältnis bietet. Die Angebotspalette reicht vom »kostenlosen Web-

space«, der sich durch Einblendung von Werbung finanziert, bis hin zu kompletten Server-Farmen mit Clustern und Load-Balancing für viele tausend Euro im Monat.

Kostenloser Webspace

Kostenloser, werbefinanzierter Webspace ist nur für »meine ersten Gehversuche mit HTML« und für private Spaßseiten (»Homepage von Pfiffi, unserem Cocker-Spaniel«, »Liebeserklärung an Michaela«) zu empfehlen. Die Features bei solchen Angeboten sind sehr beschränkt. Die Angebote werden auf Servern mit Hunderten anderer Angebote gespeichert. Die Werbeeinblendungen, die der Provider einfach dynamisch in den HTML-Code der Seiten einfügt, bevor diese an einen aufrufenden Browser übertragen werden, werden von Anwendern meist als penetrant und nervig empfunden. Die meisten Suchmaschinen-Robots meiden Server solcher Anbieter wie der Teufel das Weihwasser, weil dort fast nur Schund zu finden ist. Die Seiten sind dementsprechend schlecht auffindbar.

Einsteiger-/Starter-Pakete/Virtual Hosting

Angebote dieser Kategorie kosten zwischen 10 und 100 Euro monatlich. Geboten wird eine für die meisten Fälle brauchbare Menge an Webspace, also Festplattenspeicher auf einem Server-Rechner, und zahlreiche Einzel-Features wie CGI, PHP, eine oft begrenzte Anzahl von Datenbanken auf einem DBS-System wie MySQL und häufig auch die Nutzung von Fertigscripts, webbasierte Verwaltungsoberfläche, automatische Suchmaschineneinträge usw. In diesem Segment regiert der Feature-Overkill, denn nicht wenige Interessenten für dieses Segment wünschen für möglichst wenig Geld möglichst viele Features, auch wenn sie nicht mal alle davon verstehen.

Die Netzanbindung solcher Angebote ist meist befriedigend bis sehr gut. In der Regel sind auch ein oder mehrere Domain-Namen inklusive. Dabei können auch Mail-Postfächer unter dem Domain-Namen eingerichtet werden. Und schließlich hat der Kunde Zugriff auf Statistiken, die ihm Besucherzahlen und Besucherverhalten auf den eigenen Seiten transparent machen.

Die Seiten liegen allerdings nicht auf einem eigenen Server-Rechner, sondern mehrere Kunden des Providers teilen sich einen Rechner. Dies wird als ***Virtual Hosting*** bezeichnet, da mehrere Domain-Namen der gleichen IP-Adresse zugeordnet werden und erst in der Konfiguration des Webservers unterschieden werden. Der Server-Rechner ist bei seriösen Providern selbstverständlich so konfiguriert, dass Kunden nicht bei ihren Nachbarn spähen oder gar wildern können.

Geeignet sind solche Lösungspakete für Auftritte kleinerer Unternehmen, für Vereine, kleinere Organisationen und für ambitionierte Privatprojekte. Das Unterhalten kleinerer Webanwendungen wie Gästebuch, Umfragen oder ein kleines Shop-System sind im Rahmen solcher Pakete ebenfalls kein Problem.

Server-Hosting/Eigener Root-Server

Während Ihre Webseiten bei den zuvor genannten Lösungen auf dem gleichen Server-Rechner liegen wie diejenigen anderer Kunden, mieten Sie bei dieser Kategorie einen eigenen Server-Rechner inklusive vorinstallierter Software. Die Preise in dieser Kategorie schwanken meist zwischen 30 und 300 Euro, je nach Hardwarequalität und Zusatzleistungen wie 24-Stunden-Support usw.

Server-Rechner dieser Art sind üblicherweise flache Kisten, ähnlich wie Pizzaschachteln, die der Provider entweder selbst oder in einem Rechenzentrum in so genannte Racks (Rechnerregale) einbaut. Die Hardware besteht vorwiegend aus PC-üblichen Teilen, abgesehen von Bauteilen, die auf die flache Bauweise der Schachteln abgestimmt sein müssen.

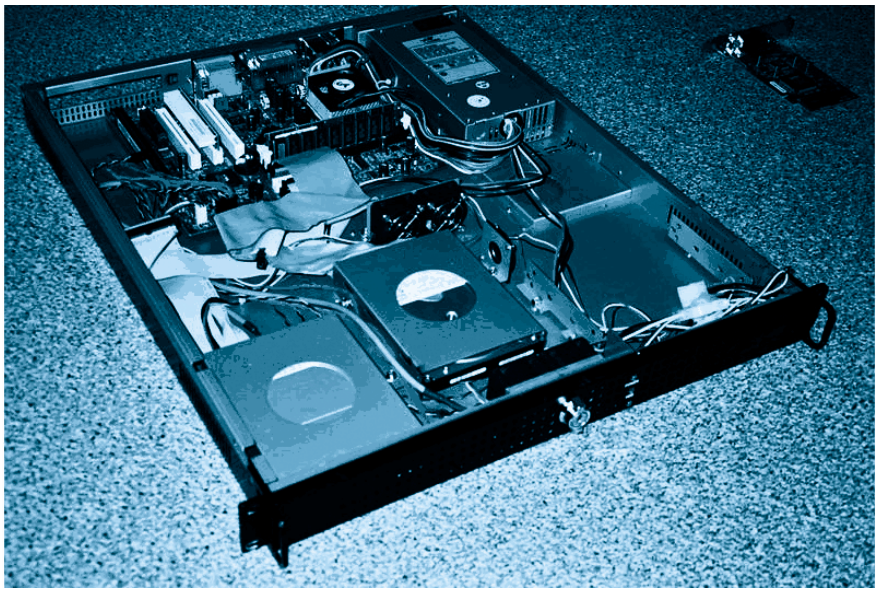


Abbildung 9.4: Typischer Root-Server in moderner Flachbauweise für Einbau in 19«-Racks

Als Betriebssystem ist meistens eine vom Provider bevorzugte Linux-Distribution installiert. Ebenso ist in der Regel alles vorinstalliert, was zum Betrieb einer Website nötig ist, also ein Webserver, Scriptsprachen wie PHP und Perl, ein Datenbanksystem wie MySQL, Webstatistik-Tools wie der Webalizer usw. Das System ist auch schon so eingerichtet, dass Sie sofort loslegen können.

Sie erhalten vollen Root-Zugang zu dem Rechner und können somit alles an dem Rechner selbst verwalten, neue Software installieren, Betriebssystem und Webserver ohne Einschränkung konfigurieren usw.

Auch in diesen Angeboten sind in der Regel mehrere Domain-Namen, Mail-Postfächer und diverse Zusatzfeatures enthalten. Besonders achten sollten Sie in diesem Segment jedoch auch auf die Angaben zum Support. Wenn beispielsweise der Server-Rechner abstürzt und über die Online-Verbindung nicht mehr erreichbar ist, sollte der Provider ihn

schnellstmöglich neu booten und gegebenenfalls vor Ort über Konsolenzugang inspizieren können. Manche Provider bieten auch Möglichkeiten wie die an, einen Kaltstart-Reboot via Handy-Anruf auszulösen.

Da sich die Preise in diesem Segment durchaus mit denen des Virtual Hostings überlappen, lohnt sich ein Vergleich, ob nicht gleich das Mieten eines eigenen Server-Rechners attraktiver ist. In jedem Fall ist diese Kategorie erforderlich, wenn Sie mehrere Webangebote unterhalten, wenn Sie eine größere Menge an Seitenbesuchern haben oder zumindest erwarten und wenn Sie anspruchsvollere Webanwendungen entwickeln, bei denen auch mal zusätzliche Software installiert werden muss oder tiefere Eingriffe in die System- oder Webserver-Konfiguration nötig sind.

Solange Sie keine zu großen Veränderungen am System vornehmen und kein Spielkind sind, das zwanghaft alle Schräubchen verdrehen muss, genügt das Wissen eines Webmasters, wie es in diesem und den folgenden Buchkapiteln vermittelt wird, um einen eigenen Root-Rechner, dezidierten Server oder wie auch immer die Angebote in dieser Kategorie bezeichnet werden, am Laufen zu halten. Allerdings kann es nicht schaden, einen Systemspezialisten zu kennen, der in der Lage ist, das System auf die spezifischen Anforderungen des eigenen Webangebots hin zu optimieren.

Die Lösung des dezidierten Servers, Root-Servers oder ähnlich bezeichnet empfiehlt sich auf jeden Fall für alle Unternehmen, die neben ihrem Firmenauftritt auch Webanwendungen für Business-to-Business-Kommunikation unterhalten. Sie empfiehlt sich ferner für alle Projekte mit attraktiven Inhalten oder mit anspruchsvolleren Anwendungen wie Board- oder Forensystemen, größeren Shop-Systemen mit Datenbankanbindung usw.

Server-Housing/Betrieb eigener Rechner

Wenn Ihnen die von den meisten Providern angebotenen Root-Server-Lösungen nicht zusagen, weil Hard- und Software nicht Ihren Vorstellungen oder Erfordernissen entsprechen, bleibt noch die Angebotsstufe, selbst finanzierte Hardware mit selbst installiertem System und eigener Software hosten zu lassen. In diesem Fall steht die Hardware nur deshalb beim Provider oder in einem von ihm gemieteten Platz in einem Rechenzentrum, weil dort die erforderliche Internetanbindung mit hoher Bandbreite eingerichtet ist. Der Service, den der Kunde vom Provider verlangen kann, ist bei solchen Lösungen natürlich beschränkter als bei Root-Server-Lösungen. Stattdessen werden häufig Serviceverträge mit dem Hardwarehersteller oder Drittanbietern abgeschlossen, die im Notfall Zugang zum Server-Rechner haben und ihn warten können.

Die Vorteile in diesem Angebotssegment sind, dass Sie auch edlere, PC-unübliche Hardware einsetzen können wie 64-Bit-Systeme, Parallelprozessoren, abgestimmte RAID-Systeme usw. Bis hin zum Betrieb ganzer Server-Farmen ist in diesem Segment eigentlich alles möglich. Provider, die dieses Segment anbieten, sind jedoch häufig selbst Betreiber eines Rechenzentrums oder Backbone-Provider. Die Angebote kommen meist nicht von der Stange, sondern werden individuell aufgrund einer Anfrage erstellt. Die Preise sind dabei natürlich auch andere als beim Hosting von Standard-Root-Servern – sie beginnen bei 300 Euro und sind nach oben offen, je nachdem, was gewünscht wird.

9.2.2 Traffic und Transfervolumen

Die physikalische Infrastruktur des Internets, also der Ausbau und Unterhalt weltweiter Leitungsverbindungen und Rechenzentren mit zentralen Gateway-Großrechnern kostet viel Geld. Finanziert wird diese Infrastruktur unter anderem dadurch, dass die Netzteilnehmer für die Datenmengen, die sie verursachen, bezahlen. In der Praxis sieht das so aus, dass Provider giga- und terabyteweise Datenvolumen einkaufen. Die so entstehenden Unkosten fließen in unterschiedlichen Kalkulationsmodellen in die Endkundenangebote mit ein.

Eines der gängigsten Modelle besteht darin, ein Volumenlimit festzulegen. In der Praxis eines Webhosting-Angebots liest sich das dann in Ausdrücken wie »100 Gigabyte Traffic inklusive«. Doch was bedeutet das aus Kundensicht genau?

Auch bei vielen Internet-Zugangsangeboten gibt es Volumenlimits, Traffic-Begrenzungen oder wie immer das Charakteristikum bezeichnet wird. Jede Online-Aktion im Internet, egal ob Surfen im Web, Abrufen oder Versenden von E-Mails, Chatten oder ICQ, File-Sharing oder Internetradio hören, verursacht Datenübertragungen. Die Menge der übertragenen Daten kann sehr unterschiedlich sein und hängt von zahlreichen Faktoren ab. Eine typische einzelne Webseite mit einigen Grafiken kommt vielleicht auf 60 Kbyte, was aber nicht zwangsläufig bedeutet, dass diese 60 Kbyte in jedem Fall übertragen werden – abhängig vom Browser-Cache oder Features wie gzip-Komprimierung bei der Übertragung kann es auch weniger sein. Dafür kommen wieder einige Bytes dazu, wenn Browser und Server kommunizieren. Ebenso verhält es sich bei anderen Diensten.

Ist es schon beim eigenen Internetzugang schwer abschätzbar, wie viel Datenvolumen tatsächlich benötigt wird, so wird dies beim Web-Hosting noch ungleich problematischer. Denn hier entsteht Ihnen der Traffic nicht bei dem, was Sie selbst abrufen, sondern bei dem, was die Besucher Ihrer Seiten abrufen (und natürlich auch bei allen Arbeiten, die Sie selbst am Server durchführen, wie z.B. beim Upload von Daten). Kurz: Jedes Byte, das vom Server angefordert oder an ihn übertragen wird, fließt ins verbrauchte Datenvolumen ein.

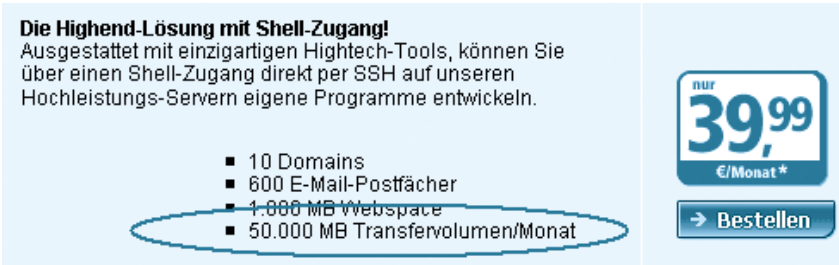


Abbildung 9.5: Angebot mit Volumenlimit

Übersteigt die Menge des übertragenen Datenvolumens die festgelegte Grenze, wird der Server jedoch nicht einfach abgeschaltet. Stattdessen müssen Sie dann nachzahlen. Die Provider bieten dafür unterschiedliche Modelle an. In einigen Fällen zahlen Sie pro mehr

verbrauchtem Mega- oder Gigabyte ein paar Cent und in anderen Fällen gibt auch es größere Klassen wie etwa »bis 50 Gigabyte überschrittenem Datenvolumen«.

Die Volumenbegrenzungen sind zwar bei seriösen Providern so realistisch hoch angesetzt, dass nicht gleich jeder kleine Firmenauftritt zu einem finanziellen Risiko wird. Doch wenn Sie datenintensive Angebote wie eine Download-Site unterhalten, kann das dadurch verbrauchte Datenvolumen schnell in ungeahnte Höhen schießen. Oder Sie erleben einen so genannten *Slashdot-Effekt*: Ihre Site wird beispielsweise in einer zuschauerreichen Fernsehsendung genannt oder von einer Mega-Site wie Yahoo prämiert. Dann kommen »über Nacht« plötzlich sehr große Besuchermengen auf Ihre Seiten – und das dabei entstehende Datentransfervolumen geht auf Ihre Kosten.

Je mehr Besucher eine Site also anzieht, desto eher sollten Sie darauf achten, ein Angebot zu wählen, bei dem keine Volumenbegrenzung besteht. Auch solche Angebote gibt es – die Provider kalkulieren in diesem Fall so, dass einem Kunden mit hohem Traffic viele andere gegenüber stehen, deren Angebote vergleichsweise wenig Traffic verursachen. Allerdings ist dringend zu empfehlen, bei Angeboten ohne Volumenlimit auf das Kleingedruckte im Mietvertrag zu achten: Oftmals wird dort quasi durch die Hintertür doch noch ein Volumenlimit festgelegt.

9.2.3 Webspace und Speicher

Wenn Sie ein Angebot mit Virtual Hosting mieten, gehört dazu auf jeden Fall die Angabe der verfügbaren Webspace-Menge. Gemeint ist damit, wie viel Festplattenspeicher Ihnen eingeräumt wird, um darauf die Daten Ihres Webangebots abzulegen. Mehrere hundert Mbyte sind hierbei auf jeden Fall geboten, auch wenn HTML und Grafikdateien des ersten Firmenauftritts gerade mal ein paar hundert Kbyte groß sind. So ist nämlich in der Speichermenge meist auch diejenige zum Unterhalt der domainspezifischen E-Mail-Postfächer eingeschlossen. Außerdem wächst fast jedes Webprojekt im Laufe der Zeit und der Server beherbergt dann meist auch ältere, inaktive Versionen oder er wird zum privaten Datenaustausch über FTP benutzt. Es gibt jedenfalls viele Gründe, warum der verfügbare Webspace um ein Mehrfaches höher sein sollte als der aktuell benötigte Bedarf.

Bei Angeboten für eigene Root-Server wird weniger mit Webspace geworben. Stattdessen muss beim Angebotsvergleich die Festplattenkapazität des Server-Rechners in Kombination mit dem Plattenverbrauch der vorinstallierten Software berücksichtigt werden. Ebenfalls berücksichtigt werden muss, dass bei eigenen Servern auch die Logfiles aller laufenden Server Festplattenspeicher verbrauchen, was je nach Traffic und Logging-Konfiguration erhebliche Speicherkapazitäten benötigt. Der Plattenspeicher des Server-Rechners sollte in jedem Fall dem eines besseren PCs entsprechen.

CPU und Arbeitsspeicher/RAM

Bei Virtual Hosting werden Sie dazu keine sinnvollen Angaben erhalten, da der Server und seine Rechenkapazitäten ohnehin mit anderen Kunden geteilt wird. Root-Server-Angebote dürfen jedoch nur als seriös bezeichnet werden, wenn der Anbieter detailliert Auskunft über die technischen Daten der eingesetzten Hardware gibt.

Glaubenskriege über den besten Prozessor gehören nicht in dieses Buch. Root-Server werden meist mit PC-üblichen Prozessoren angeboten. Vergleichen Sie im Zweifelsfall, welche CPU-Ausstattung aktuell bei normalen PCs der Ober- und Spitzenklasse angeboten wird. Ein ordentlicher Root-Server sollte mindestens über die gleiche Ausstattung verfügen. Besser und natürlich teurer sind Angebote mit zwei oder mehreren Prozessoren.

Beim Arbeitsspeicher sollten die Werte deutlich über den aktuell üblichen Durchschnittswerten von PCs liegen – das Doppelte bis Vierfache davon ist Minimum. Denn wenn beispielsweise 20 Besucher und »Besucher« wie Suchmaschinen-Robots gleichzeitig auf Ihren Webseiten surfen, werden 20 Webserver-Prozesse gestartet, von denen jeder zweistellige Megabyte-Zahlen im Arbeitsspeicher verbraucht. Dazu kommen andere laufende Server und andere regelmäßige Prozesse, ausgeführte Scripts, Datenbankaufrufe usw. Schon ein einziger Seitenbesucher kann auf diese Weise schnell mal 50 Mbyte Arbeitsspeicherbelegung verursachen. Zwar beherrschen alle modernen Betriebssysteme das Swapping, also das Auslagern von Arbeitsspeicher auf die Festplatte, falls der Speicher mal nicht ausreicht. Doch der Rechner wird dadurch zu langsam und Seitenbesucher müssen mit merklichen Verzögerungen rechnen.

9.2.4 Serverzugänge über FTP, SCP/SFTP, Telnet und SSH

Wichtig bei allen Hosting-Angeboten ist, wie Sie Ihre Webseiten auf den Server bekommen und wie Sie auf dem Server gespeicherte Daten verwalten können. Hierbei ist ein klarer Fingerzeig nötig: Meiden Sie von vornherein alle Angebote, bei denen ausschließlich FTP und Telnet angeboten werden! Dies sind veraltete Internetdienste aus einer Zeit, als das Internet noch eine friedliche Wiese für ein paar wenige, vernünftige und wohlmeinende Menschen war. Heute jedoch, im Zeitalter immer dreisterer Spam-Methoden, Massenviren, Phishing, Spoofing und anderen kriminellen Erscheinungen im Netz sollten Sie für den Zugang zu einem Server nur noch »sichere« Zugangsverfahren einsetzen. Und das ist heute vor allem der Zugang über SSH, wobei SFTP ein Teil davon ist.

SSH steht für *Secure Shell* und bezeichnet sowohl ein Datenübertragungsprotokoll als auch dessen konkrete Implementierung. Das Protokoll erlaubt das Anmelden auf einem entfernten Host im Internet, das Arbeiten auf dem entfernten Rechner mit einem dort verfügbaren Kommando-Interpreter/einer dort verfügbaren Shell und das Übertragen von Dateien zwischen lokalem Rechner und entferntem Rechner. Letzteres wird auch als SCP bezeichnet. Seit der Spezifikationsversion 2.0 wird neben SCP, was nur einfaches Kopieren zwischen den entfernten Rechnern ermöglicht, auch eine FTP-Variante angeboten, nämlich SFTP. Damit sind auch erweiterte Arbeiten wie die Vergabe von Rechten für Dateien, das Auflisten von Verzeichnisinhalten usw. möglich.

Da die technischen Spezifikationen der Versionen 1.0 und 2.0 von SSH teilweise inkompatibel sind, sollte der Provider auf jeden Fall den Standard 2.0 anbieten.

Das entscheidende Charakteristikum an SSH ist, dass alle Übertragungen, egal ob vom lokalen Rechner zum entfernten Server-Rechner oder umgekehrt, verschlüsselt übertragen werden. Dabei werden standardisierte und als sicher geltende Verschlüsselungsmethoden wie IDEA, Triple DES, Blowfish usw. angewendet. Die Authentifizierung als

registrierter Benutzer am Server erfolgt RSA-basiert. Ohne die Details dieser Algorithmen und Verfahren kennen zu müssen, sollten Sie wissen, dass bei den veralteten Varianten FTP und Telnet beispielsweise Passwörter, mit denen Sie sich am Server anmelden, im Klartext übertragen werden und auf jedem Gateway-Rechner, den sie passieren, abgefangen werden können. Hat sich also beispielsweise ein Hacker Zugang zu einem solchen Gateway-Rechner verschafft, kann er sich mühelos massenweise Zugangsdaten verschaffen, die dort im Klartext weitergeleitet werden.

Aus dem gleichen Grund, weshalb also etwa bei Online-Banking und Seiten, auf denen Sie Ihre Kreditkartennummer übertragen, nie etwas anderes als das »sichere« HTTP-Protokoll HTTPS zum Einsatz kommen sollte, sollten Sie für den Arbeitszugang zu Ihrem Server-Rechner den Zugang über SSH (2.0) verlangen.

Zwei wichtige Windows-Programme: PuTTY und WinSCP

Die früher üblichen Telnet- und FTP-Clientprogramme können Sie im Zusammenhang mit SSH nicht mehr benutzen. Für Anwender von MS Windows gibt es jedoch zwei attraktive Clients, die alles bieten, was Sie zum effizienten Arbeiten mit SSH benötigen.

PuTTY nennt sich ein Terminal-Programm, welches neben dem veralteten Telnet auch SSH in den Versionen 1.0 und 2.0 unterstützt. Das Programm ist klein, sehr laufstabil und verzichtet auf angenehme Weise auf überflüssigen Schnickschnack. Es erlaubt das Speichern vieler verschiedener Server-Zugänge. Dabei besteht jede Zugangskonfiguration aus einem umfangreichen Set an einstellbaren Konfigurationsdaten, zu denen die verwendeten Verschlüsselungsverfahren gehören, Einstellungen zur Zeichenkodierung, aber auch Einstellungen zur Fenstergröße, Farben und Erscheinungsbild der Arbeitsfläche.

PuTTY können Sie von der folgenden Originaladresse downloaden:

http://www.chiark.greenend.org.uk/~sgtatham/putty/

PuTTY ist Open Source. Es fallen keine Lizenzgebühren an, auch bei kommerzieller Nutzung und Mehrfachkopien innerhalb eines Unternehmens nicht. Informieren Sie sich jedoch über die Rechtmäßigkeit der Verwendung der von PuTTY bzw. SSH eingesetzten Verschlüsselungsverfahren in Ihrem Land. Ohne dabei zum Staatsfeind werden zu wollen, sei jedoch an dieser Stelle ausdrücklich dazu geraten, im Zweifelsfall Zivilcourage zu beweisen. Es kann nicht angehen, dass die Sicherheitsinteressen des Staates so weit gehen, dass seine Bürger sich schutzlos allen Betrügern im Internet ausliefern müssen.

Beim Start von PuTTY erscheint ein Dialogfenster, das, mit einer explorerartigen Menüstruktur versehen, alles in einem bietet: Es erlaubt das schnelle Direktanmelden an einem entfernten Server-Rechner, das Anlegen von konfigurierten Zugängen sowie das Laden, Ändern und Löschen bereits konfigurierter Zugänge.

Die Default-Einstellungen, welche PuTTY verwendet, reichen zur Anmeldung an den meisten SSH-Servern aus. In der Regel genügt es, neben den Zugangsdaten ein paar Einstellungen zum Erscheinungsbild vorzunehmen, zumindest dann, wenn mit einem Zugang regelmäßig und/oder länger gearbeitet werden soll.

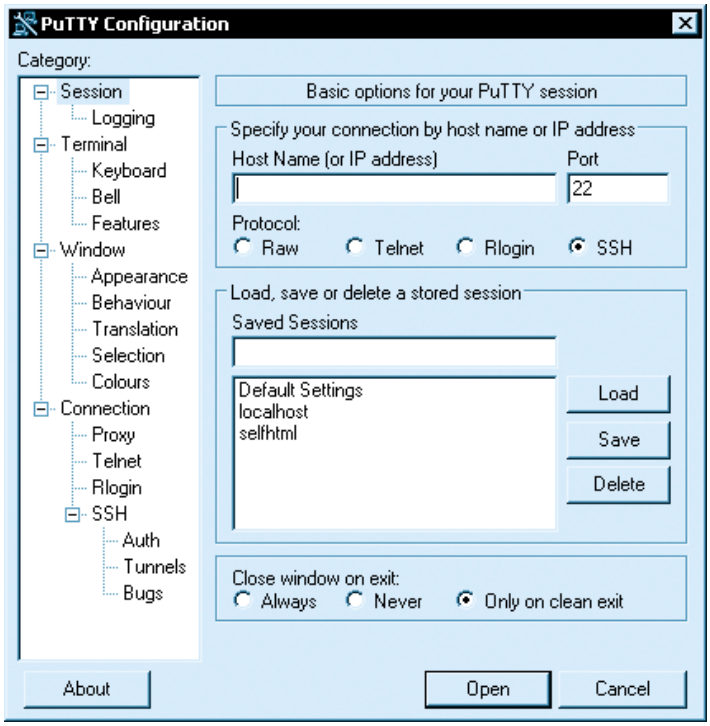


Abbildung 9.6: Das All-in-One-Dialogfenster von PuTTY

Um einen neuen Zugang anzulegen, geben Sie Ihren Domain-Namen oder die IP-Adresse des Server-Rechners oben im Feld für ***Host Name*** ein. Bei Port muss normalerweise 22 stehen – dies ist der Standard-Port für SSH-Verbindungen. Unterhalb davon klicken Sie ***SSH*** an. Geben Sie nun im Eingabefeld ***Saved Sessions*** einen frei wählbaren, sprechenden Namen für den Zugang ein. Klicken Sie anschließend auf SAVE, um den Zugang zu speichern. Wenn Sie möchten, können Sie nun durch die Menüeinträge links wandern und sich weitere Einstellmöglichkeiten ansehen. Unterhalb des Zweigs WINDOW können Sie beispielsweise Einstellungen zur optischen Erscheinung des Fensters der Arbeitsfläche einstellen. Nach Änderungen klicken Sie auf SAVE, damit die Änderungen wirksam werden.

Um einen konfigurierten Zugang zu starten, klicken Sie unten auf OPEN. Das Terminal-Fenster wird gestartet. Der SSH-Server auf dem Server-Rechner fordert zur Eingabe von Benutzername und Passwort auf. Die Passworteingabe erscheint aus Sicherheitsgründen ohne Echo am Bildschirm.

Welche Kommandos Sie nach erfolgreicher Anmeldung eingeben und welche Programme Sie auf dem Server-Rechner starten können, hängt von der dort verwendeten Shell (also vom Kommandointerpreter) und von den Rechten ab, die Sie am Server haben. In Kapitel 10 dieses Buchs lernen Sie wichtige Arbeitskommandos für Linux-Systeme kennen, da die meisten von Providern angebotenen Server-Rechner eine Linux-Variante als Betriebssystem installiert haben.

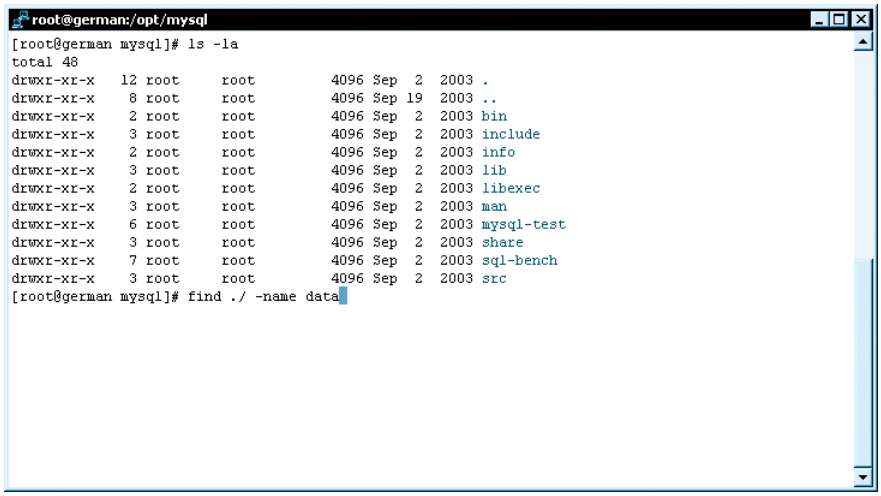


Abbildung 9.7: Das Terminal-Fenster von PuTTY

PuTTY eignet sich hervorragend zum Arbeiten auf dem Server-Rechner. Für Datenübertragungen zwischen lokalem PC und entferntem Server-Rechner sollten Sie jedoch ein speziell dafür ausgerüstetes Client-Programm bevorzugen. Ein unter MS Windows beliebter Client für SCP und SFTP nennt sich ***WinSCP***.

WinSCP ist ebenso wie PuTTY Open Source und ohne Lizenzgebühren benutzbar. Ansonsten gelten die gleichen Hinweise wie bei PuTTY.

WinSCP bietet eine attraktive grafische Oberfläche an, die sich, was das Look and Feel betrifft, wahlweise (im Programm einstellbar) am Windows Explorer oder am Stil des Norton Commander orientiert. Das Programm erlaubt es, bequem durch Maus- oder Tastaturaktionen Dateien und ganze Verzeichnisse vom lokalen PC oder LAN zum entfernten Server-Rechner zu übertragen (Upload) oder umgekehrt (Download).

WinSCP können Sie von folgender Adresse beziehen:

http://winscp.sourceforge.net/

Beim Programmstart erscheint ähnlich wie bei PuTTY ein All-in-One-Dialog zum Konfigurieren, Speichern und Laden von Konfigurationsdaten und zum Starten von gespeicherten Zugängen.

Um einen neuen Zugang einzurichten, klicken Sie auf NEW. Anschließend können Sie die Zugangsdaten eingeben. Klicken Sie am Ende auf SAVE, um den Zugang zu speichern. Um einen vorhandenen Zugang zu starten, wählen Sie ihn aus der Liste in der Mitte des Dialogfensters aus und klicken Sie dann auf LOGIN.

Die Arbeitsoberfläche von WinSCP ist weitgehend intuitiv. Das Programm eignet sich für jede Art von Up- und Download. Bei Dateiübertragungen sind zahlreiche Optionen einstellbar wie Binär- oder Textmodus, Beibehalten des Zeitstempels usw. Auf dem Server lassen sich Verzeichnisinhalte nach verschiedenen Spalten sortiert auflisten, Dateirechte setzen, Dateien löschen usw.

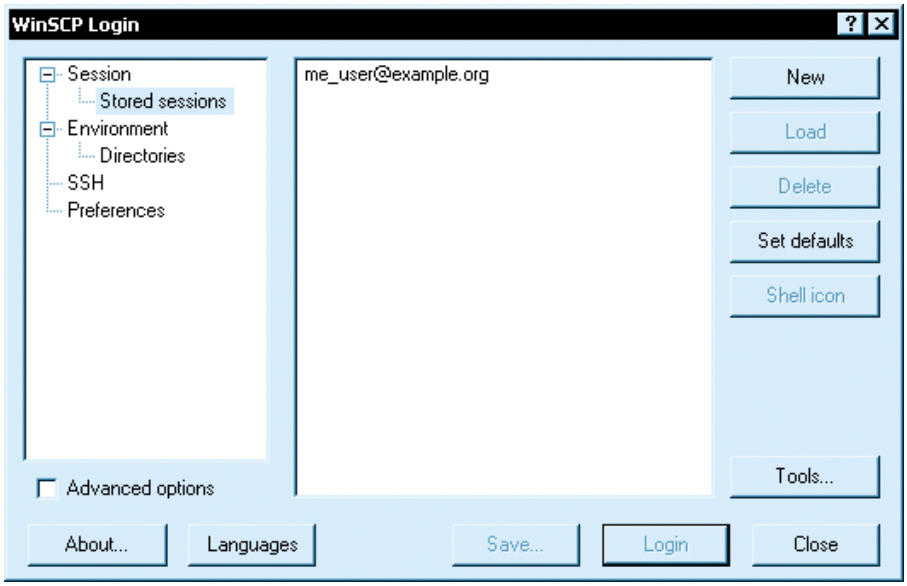


Abbildung 9.8: Das All-in-One-Dialogfenster von WinSCP

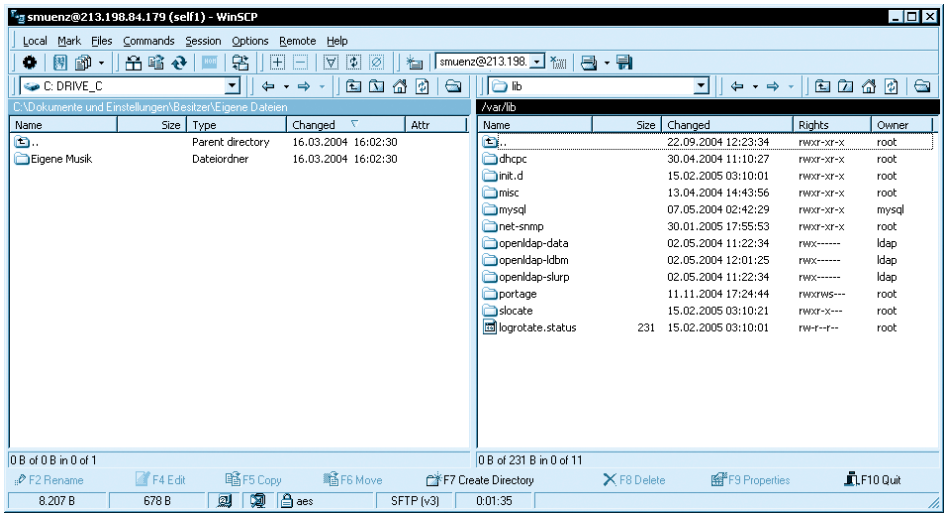


Abbildung 9.9: Das Arbeitsfenster von WinSCP (hier im Norton-Commander-Stil)

WinSCP arbeitet übrigens auch direkt mit PuTTY zusammen. Über den Menübefehl COMMANDS/OPEN in PuTTY kann ein aktuell angezeigtes Server-Verzeichnis direkt in PuTTY geöffnet werden, um z.B. Arbeiten auf Shell-Ebene vorzunehmen, die über WinSCP nicht möglich sind.

SSH und SCP/SFTP unter Linux und Macintosh

Das Shell-Werkzeug OpenSSH, beziehbar von *http://www.openssh.com/*, bietet für alle Arten von Unix-Shells eigentlich alles, was der Benutzer eines Shell-Interpreters benötigt. Das Softwarepaket besteht unter anderem aus den Kommandoprogrammen ***ssh*** (für Shell-Verbindungen zu einem entfernten Rechner), ***scp*** (für SSH-1.0-kompatibles Kopieren zwischen lokalem Dateisystem und entferntem Rechner) sowie ***sftp*** (für SSH-2.0-kompatiblen FTP-Ersatz).

Da SSH ohnehin nur eine Kommando-Shell auf dem entfernten Rechner öffnet, spricht auch nichts dagegen, das Kommando gleich von einer lokalen Shell aus zu starten. Bei SFTP dagegen bevorzugen viele Anwender jedoch grafische Oberflächen, die ein bequemes Up- und Downloaden von Dateien erlauben. Ein an fast alle grafischen Oberflächen anpassbares, für Unix-basierte Systeme geschaffenes Programm ist gFTP (*http://gftp.seul.org/*).

Für Macintosh-Rechner ist beispielsweise Interarchy ein ausgereifter grafischer SFTP-Client (*http://www.interarchy.com/*).

Weiterführende Links:

Wikipedia-Artikel zu SSH:
http://de.wikipedia.org/wiki/SSH
Einführung in die Secure Shell (SSH):
http://www.rrzn.uni-hannover.de/secure_shell.html

9.2.5 Domain-Namen und Umziehen von Domains

Domain-Namen, etwa des Top-Level-Typs ***.de***, kosten heute nur noch Cent-Beträge. Wenn Sie eine neue Website etablieren möchten, empfiehlt es sich auf jeden Fall, dafür einen geeigneten Domain-Namen zu reservieren. Eine ordentliche Webadresse wertet jedes Projekt auf.

Das größere Problem besteht darin, noch freie Namen zu finden. Fast alle interessanten Namen, Wörter oder Wortkombinationen sind bereits belegt. Es ist schon etwas Einfallsreichtum notwendig, um einen griffigen, noch nicht reservierten Namen zu finden. Die whois-Abfragen der Top-Level-Vergabestellen unterstützen Sie bei der Suche nach freien Namen.

Bei der Namenswahl ist außerdem geltendes Namens- und Markenrecht zu beachten. Im Zweifelsfall ist es ratsam, einen Anwalt damit zu betreuen, für einen freien Wunschnamen eine Markenrecherche durchzuführen. Denn zumindest offensichtliche Markenverletzungen können bei Entdecken durch den Markeninhaber richtig teuer werden.

Die meisten Hosting-Provider bieten im Rahmen ihrer Web-Hosting-Angebote gleich das Reservieren eines oder mehrerer Domain-Namen mit an. Sie können Domain-Namen auch providerunabhängig direkt bei der Toplevel-Domain-Vergabestelle beantragen, also

etwa *.de*-Domains beim Denic. Dabei müssen Sie sich jedoch selbst darum kümmern, alle geforderten Daten zu erbringen.

Falls Sie bereits die gewünschten Domains reserviert haben und nur zu einem anderen Provider wechseln möchten, ist das Handling im Prinzip zwar festgelegt und einfach. Doch leider kommt es immer mal wieder zu Problemen, weil manche Provider nicht richtig spuren.

Das Problem bei einem Wechsel besteht aus technischer Sicht darin, dass dem Domain-Namen bei einem anderen Provider auf einem anderen Server-Rechner natürlich eine andere IP-Adresse zugewiesen werden muss. Der neue Provider kann die Änderung zwar über seinen DNS-Server anstoßen, jedoch nur dann, wenn er dazu eine Freigabemitteilung des alten Providers erhält.

KK-Anträge

Das Verfahren dazu ist der so genannte *KK-Antrag* (die Abkürzung *KK* steht für Konnektivitäts-Koordination). KK-Anträge sind an keine spezielle Form gebunden. Sie müssen lediglich alle erforderlichen Daten enthalten und dokumentenecht sein, d.h. vom Domain-Inhaber oder einem von ihm Bevollmächtigten unterschrieben sein.

Der Domain-Inhaber kann den Antrag entweder selber stellen oder der neue Provider übernimmt das für ihn. Fragen Sie jedoch in jedem Fall den neuen Provider, welches Verfahren er bevorzugt. Falls Sie selbst den KK-Antrag stellen, genügt folgender Beispieltext den Anforderungen:

> *Hiermit teile ich Ihnen mit, dass die Domain mein-domainname.de von folgendem Provider übernommen werden soll: Name, Anschrift des neuen Providers. Der Umkonnektierung (KK) der o.g. Domain stimme ich als Admin-C hiermit ausdrücklich zu.*

Der alte Provider muss auf einen KK-Antrag innerhalb einer angemessenen Frist, die jedoch nicht mehr als ein paar Werktage dauern sollte, reagieren. Reagiert er mit einem so genannten ACK (*acknowlegdement*), bedeutet dies, dass er die Domain freigibt. Der neue Provider kann den Domain-Namen in der Konfiguration seines DNS-Servers der IP-Adresse des Server-Rechners zuweisen, auf dem Ihre Webseiten liegen werden. Reagiert der alte Provider dagegen mit einem so genannten NACK (*no acknowlegdement*), bedeutet dies, dass er die Domain nicht freigibt. Gründe dafür können Zahlungsrückstände oder Vertragsverletzungen Ihrerseits sein oder aber, dass der KK-Antrag nicht die erforderlichen Voraussetzungen erfüllt, also z.B. nicht oder nicht vom Domain-Inhalber selbst unterschrieben ist.

Bei *.com*, *.net* und anderen Toplevel-Domains funktioniert das Handling ähnlich. Fragen Sie einfach den neuen Provider.

9.2.6 Server-Software und Features

Typische Virtual-Hosting-Angebote und Root-Server-Angebote sollten genau angeben, welche Software vorinstalliert ist. Die typische Standardkonfiguration heutiger Anbieter

ist eine so genannte *LAMP-Konfiguration*. *LAMP* steht für *L*inux *A*pache *M*ySQL *P*HP. Da die Kombination dieser Produkte vielfach erprobt ist und von vielen Webentwicklern nicht von ungefähr als »Lieblingsumgebung« bevorzugt wird, ist die entsprechende Kombination auf jeden Fall nicht verkehrt. Da die Kombination ausschließlich aus Open-Source-Produkten besteht, fallen auch keine Nutzungsgebühren für Software an.

Während Sie bei Virtual-Hosting-Angeboten jedoch kaum Einfluss auf die vorinstallierte Software haben, können Sie bei einem eigenen Server-Rechner mit root-Zugang vorinstallierte Software löschen und/oder andere installieren. Sie können (und sollten allerdings auch) selbstständig auf regelmäßige Produkt-Upgrades achten. Denn auch Produkte wie Linux oder PHP sind nicht frei von Sicherheitslücken und oftmals enthalten numerisch harmlose Zwischenversionen wichtige Schließungen von bekannt gewordenen Sicherheitslücken.

Neben der Basis-Software werden in typischen Web-Hosting-Angeboten immer wieder diverse »Features« angepriesen. Dazu gehören beispielsweise SSI, CGI-Scripts, Verzeichnisschutz mit .htaccess oder cron-Jobs. Wenn Sie einen eigenen Root-Server mieten, sind solche Dinge sowieso selbstverständlich, weil es sich dabei eigentlich gar nicht um unglaublich tolle Features handelt, sondern lediglich um ein paar Einstellungen im System oder in der Konfiguration des Webservers. Auf Server Side Includes (SSI), CGI-Schnittstelle und Verzeichnisschutz werden wir in Kapitel 11 noch näher eingehen. Kapitel 10 befasst sich unter anderem auch mit dem Thema cron-Jobs.

Was ebenfalls zu einer ordentlichen Vorkonfiguration eines Root-Servers gehören sollte, ist eine brauchbare Webstatistik. Der Webserver loggt zwar alle Zugriffe mit, doch die Logfile-Auswertung sollte besser einer darauf spezialisierten Software überlassen werden.

Sehr beliebt und ebenfalls ein Open-Source-Produkt ist der *Webalizer* (*http://www.webalizer.org/*). Über eine webbasierte Oberfläche – also im Browser – meist über ein passwortgeschütztes Unterverzeichnis erreichbar, können Sie dabei täglich aktualisierte Statistiken abrufen, die sowohl tabellarisch als auch grafisch aufbereitet werden. Sollte ein Angebot keine Angabe zu Statistiken enthalten, fragen Sie auf jeden Fall beim Provider deswegen nach. Ordentliche Benutzungsstatistiken sind eine wichtige Basis zur Überprüfung der Website hinsichtlich Beliebtheit, Server-Belastung usw.

9.2.7 Datensicherung, Überwachung und Support

Früher wurde in Web-Hosting-Angeboten häufig mit einem Prozentwert für »garantierte Verfügbarkeit« geworben, der je nach Mut des Providers irgendwo zwischen 98% und 99,99% lag. Heute liest man diese Zahlen seltener, da sie häufig unhaltbare Versprechen waren, die von Kunden eingeklagt wurden, was am Ende teuer für die Provider war.

Fakt ist, dass auch stabil konfigurierte Server-Rechner einmal so abstürzen können, dass nur noch ein Neustart von der Konsole aus hilft. Darüber hinaus gibt es natürlich auch Fälle, die noch gravierender sind, beispielsweise wenn am Server-Rechner eine Festplatte crasht oder eine CPU durchbrennt.

Entscheidend dabei ist, wie schnell der Provider solche Fälle erkennt und für Abhilfe sorgt. Manche Provider unterhalten richtige Überwachungsmechanismen und entsprechendes Personal, das selbstständig bei auftretenden Problemen reagiert. Andere Provider reagieren erst, wenn Sie als Site-Betreiber anrufen, um die Nichtverfügbarkeit des Webangebots zu melden. Und manche Provider brauchen sogar dann noch unangemessen lange, bis eine Reaktion erfolgt. In den Web-Hosting-Angeboten ist davon natürlich nichts zu erfahren. Was diese Seite des Hostings betrifft, hilft nur, sich umzuhören und Erfahrungsberichte von anderen Kunden zu studieren.

Meinungen und Erfahrungen dieser Art können Sie beispielsweise aufstöbern, indem Sie auf der Website von Google über dem Suchfeld auf Groups klicken und anschließend den Namen eines Providers als Suchbegriff eingeben. Als Ergebnis erhalten Sie dann, was in allen möglichen Newsgroups über diesen Provider geredet wurde und wird.

Wenn tatsächlich mal eine Festplatte crasht, ist neben dem schnellen Auswechseln natürlich auch das Thema Datensicherung von Bedeutung. Gerade bei Websites, deren Inhalt teilweise oder sogar hauptsächlich auf Benutzereingaben basiert, wie etwa bei Board- oder Foren-Systemen, bei Auktionen oder Kleinanzeigen usw., ist ein tägliches und schnell wieder einspielbares Komplett-Backup Pflicht. Schildern Sie im Zweifelsfall einem potenziellen Provider ein denkbar schlimmes Szenario: Sie unterhalten einen elektronischen Kleinanzeigenmarkt, bei dem Kunden für das Schalten von Kleinanzeigen bezahlen. Nun crasht die Festplatte mit den Daten. Wie schnell kann der Provider das Angebot wieder zum Laufen bringen und wie viele Daten für welchen Zeitraum können maximal verloren gehen? An den Reaktionen des Providers können Sie hoffentlich bereits erkennen, ob er klare Aussagen zu seiner Backup-Strategie machen kann oder nicht.

Ein Hosting-Provider ist ein wichtiger Vertragspartner, ähnlich wie ein Wohnungsvermieter. Wie weit die vertraglichen Leistungen reichen, ist variabel. Nicht wenige Provider bieten neben dem Hosting auch gleich Webdesign und Webprogrammierung mit an. Wenn ein schneller Webauftritt her muss und Know-how oder Manpower zur Ersterstellung fehlt, ist eine solche Lösung aus einer Hand durchaus erwägenswert. Im Normalfall ist es jedoch besser, wenn sich der Provider auf den technischen Support beschränkt. Wie weit dieser Support geht, ist natürlich ebenfalls flexibel auslegbar. Einige Provider übernehmen das Updaten oder Nachinstallieren von Basis-Software nur gegen Bezahlung auf Stundenbasis, andere wiederum bieten fixkostenbasierte Wartungszusatzverträge für Kunden an, die zwar einen eigenen Root-Server wünschen, denen jedoch das Know-how fehlt, diesen zu pflegen.

9.3 Client, Server und Protokolle

Webseiten, die weltweit und unterbrechungslos übers Web aufrufbar sein sollen, müssen, so ist nun klar geworden, auf einem Rechner »gehostet« werden, der selbst ein »vollwertiger Host« ist, also ein Rechner mit permanenter, breitbandiger Internetanbindung für Up- und Downstream und fester IP-Adresse. Damit das Webangebot namentlich erreichbar ist, sollte ein geeigneter Domain-Name registriert werden. Im DNS-Server ist die Zuordnung

zwischen Domain-Name und IP-Adresse einstellbar. Da dies alles die üblichen netz- und computertechnischen Umgebungsbedingungen der Anbieter von Webseiten übersteigt, gibt es Hosting-Provider, die das »Hosten« übernehmen.

Ist der Mietvertrag beim Hosting-Provider unterschrieben, erhält der Kunde normalerweise umgehend ein paar Daten. Dazu gehören die IP-Adresse des Rechners, auf dem er seine Webseiten ablegen kann, sowie ein Benutzername und ein Passwort für Zugänge über SSH und SFTP und manchmal noch weitere Benutzernamen-/Passwort-Kombinationen für Zugänge zu einer webbasierten, also im Browser aufrufbaren Verwaltungsoberfläche oder für den Aufruf von grafischen Verwaltungsoberflächen für PHP und/oder MySQL.

Egal ob Virtual Hosting oder eigener Root-Server: Der Provider hat die Systeme so vorkonfiguriert, dass Sie sich mit entsprechenden Clients wie PuTTY, WinSCP oder mit Ihrem Webbrowser direkt anmelden und loslegen können. Tatsächlich bedeutet das: Auf dem Host-Rechner sind bereits diverse Softwareserver gestartet, welche es ermöglichen, dass Sie sich mit einem zugehörigen Client-Programm dort anmelden und arbeiten. Und nur wegen dieser gestarteten Softwareserver verdient der Host-Rechner selber die Bezeichnung »Server«.

In diesem Abschnitt gehen wir näher auf den Zusammenhang zwischen Client, Server und Protokoll ein. Das Verständnis für diesen Zusammenhang ist auch wichtig, um serverseitiges Programmieren mit PHP und clientseitiges Programmieren in JavaScript/DOM im Kopf sauber auseinander halten zu können.

9.3.1 TCP/IP und UDP

TCP/IP ist die bekannteste und am häufigsten genutzte Basis für Datenübertragungen im Internet. TCP/IP ist eine Kombination von zwei Datenübertragungsprotokollen: Das IP (*Internet Protocol*) regelt, welche Informationen ein Datenpaket enthalten muss und wie der Datensatz innerhalb eines Datenpakets aufgebaut sein muss. Datenpakete, die nicht den Regeln entsprechen, werden aussortiert.

Das IP ist jedoch verbindungslos, d.h., abgesendete Datenpakete können damit auf Korrektheit überprüft werden, aber nicht daraufhin, ob sie auch beim Empfänger angekommen sind. Die Aufgabe der Empfangskontrolle übernimmt stattdessen das TCP (*Transmission Control Protocol*).

Bei TCP/IP gilt eine Datenübertragung nur dann als erfolgreich abgeschlossen, wenn alle Datenpakete den IP-Regeln entsprechen und TCP den erfolgreichen Empfang aller gesendeten Datenpakete meldet.

Aus diesem Grund eignet sich TCP/IP für alle Datenübertragungen, die verlustfrei sein müssen und bei denen es nicht darauf ankommt, ob sie eine hundertstel Sekunde früher oder später eintreffen. Beim Übertragen einer Mail oder einer Webseite sollen ja alle Zeichen korrekt übertragen werden. Anders verhält es sich bei UDP (*User Datagram Protocol*). Dieses ist wie TCP ein Übertragungsprotokoll, jedoch mit dem Unterschied, dass eine Datenübertragung nicht vom erfolgreichen Empfang von Datenpaketen abhängig ist.

UDP eignet sich daher vorwiegend für Übertragungen, bei denen Datenverluste in akzeptablem Umfang hinnehmbar sind, wie beispielsweise bei Voice over IP (VoIP, Internettelefonie).

Die hier relevanten Internetdienste, allen voran der Transport von Webseiten, basieren auf TCP/IP. Jeder Rechner, der am Internet teilnehmen und entsprechende Dienste nutzen oder anbieten soll, muss über eine Netzwerkschnittstelle verfügen und das TCP/IP-Protokoll unterstützen. Die Netzwerkschnittstelle stellt die physikalische Verbindung zum Netz her. Das TCP/IP-Protokoll überwacht ein- und ausgehende Daten der Netzwerkschnittstelle, um aus dem Datenstrom nach seinem Standard unterscheidbare Datenübertragungen zu identifizieren.

Das IP-Protokoll regelt wie bereits erwähnt den Aufbau von Datenpaketen. Datenpakete bestehen aus einem IP-Header und dem eigentlichen Dateninhalt. Es hilft zum besseren Verständnis vieler später behandelter Themen, den Aufbau eines IP-Headers einmal näher zu betrachten. Nachfolgende Tabelle listet Felder eines IP-Datenpakets und deren Bedeutung/Inhalt auf:

Feld	Bedeutung/Inhalt
Version	Version des verwendeten IP-Protokolls. Standardversion ist derzeit Version 4. Geplant ist eine Version 6, die unter anderem einen größeren IP-Adressraum ermöglicht.
IHL (IP Header Length)	Gibt an, wie lang der IP-Header insgesamt ist. Nur so kann der IP-Header von den nachfolgenden Nutzdaten sauber getrennt werden.
Type of Service	Durch Kombinieren von Bits lässt sich entscheiden, welche Prioritäten dieses Datenpaket haben soll, ob eine schnelle Übertragung wichtiger ist oder ob eine möglichst sichere Ankunft beim Empfänger wichtiger ist.
Total Length	Gibt die gesamte Länge des Datenpakets inklusive des Headers an. Nur so ist im Datenstrom kontrollierbar, wo ein Datenpaket aufhört und wo ein neues beginnt.
Identification Flags Fragment Offset	Eine Identifikationsummer, die innerhalb einer Serie von Datenpaketen, die zu einer Datenübertragung gehören, eindeutig ist, sorgt in Verbindung mit Angaben zum Behandeln unterschiedlich spät eintreffender Datenpakete dafür, dass Datenpakete auf Empfängerseite wieder zu den Daten zusammengesetzt werden, die eigentlich übertragen wurden.
Time to Live	Zahl, die von jedem Gateway, den das Datenpaket passiert, um 1 verringert wird. Ist der Wert 0, wird das Datenpaket nicht mehr weiterverarbeitet und gelöscht. So wird verhindert, dass nicht zustellbare Datenpakete »ewig« weiter im Netz herumgeistern.
Protocol	Enthält die Portnummer des Internetdienstes, zu dem die Datenübertragung gehört.
Header Checksum	Prüfsumme, die für die Datenintegrität bei der Weiterleitung des Datenpakets sorgt.

Tabelle 9.1: Felder eines IP-Headers

Feld	Bedeutung/Inhalt
Source Address	IP-Adresse des sendenden Host-Rechners.
Destination Address	IP-Adresse des empfangenden Host-Rechners.
Options & Padding	Diverse Zusatzinformationen.

Tabelle 9.1: Felder eines IP-Headers (Fortsetzung)

Gut erkennbar ist, dass jedes IP-Datenpaket in seinem Header Angaben dazu enthält, woher das Paket kommt, wohin es soll, wie groß es ist, zu welchem Internetservice es gehört und das wievielte Paket einer Datenübertragung es ist. Auch wenn Datenpakete in einer anderen Reihenfolge beim Empfangs-Host eintreffen als sie beim Sender-Host abgesendet wurden, können sie auf Grund dieser Angaben korrekt wieder zusammengesetzt werden. Dies ist das entscheidende Charakteristikum von TCP/IP-Datenübertragungen.

Zum Zerlegen in Datenpakete beim Senden und zum Zusammenfügen beim Empfangen bedarf es natürlich einer entsprechenden Softwareintelligenz bei den Host-Rechnern. Diese wird als so genannter TCP/IP-Socket bezeichnet. Jeder PC, aber auch jeder Server-Rechner, der am Internet teilnehmen will, muss über einen solchen Socket verfügen. Unter MS Windows ist dies beispielsweise eine DLL-Datei namens *winsock.dll*.

9.3.2 Daemons, Server und Ports

Der Ausrichtung des Buchs folgend beschränken wir uns auf die Linux-Welt, da die meisten Server-Rechner für Web-Hosting heute auf Linux-Varianten oder ähnlichen, Unix-basierten Betriebssystemen basieren. Das bedeutet aber nicht, dass Server-Software nur unter solchen Systemen lauffähig ist. Auch auf jedem moderneren Windows-PC laufen von Haus aus einige Server und andere Server lassen sich jederzeit installieren und betreiben.

In der Unix-/Linux-Welt versteht man unter einem *Daemon* ein Programm, das nach seinem Start permanent im Hintergrund geladen bleibt und in der Lage ist, von anderen Programmen/Prozessen angesprochen zu werden. Unter Windows NT/2000/XP entspricht dies den so genannten *Diensten* (engl.: *services*). »Daemon« ist übrigens eine Akronym für *disk and execution monitor*. Der Ausdruck hat keinen religiösen oder übernatürlichen Background.

inetd, Portnummern und die übrigen Internetserver

Neben diversen anderen Programmen gehören vor allem Server zur Gattung der Daemons. Server können verschiedenste Aufgaben wahrnehmen. So gibt es beispielsweise auch Druckerserver. In unserem Zusammenhang interessieren jedoch nur diejenigen Server, die für Internetdienste von Bedeutung sind.

Dabei fungiert ein Server als eine Art Super-Server. Sein Name lautet in der Unix-Welt *inetd*. Generell übrigens sind Server in der Unix-/Linux-Welt leicht an dem abschließenden »d« für »Daemon« im Programmnamen erkennbar.

Der Super-Server *inetd* erhält alle TCP- und UDP-Anfragen, die am Rechner eintreffen, ermittelt, welchem Typ sie angehören, und startet dann denjenigen Server, der für die Abarbeitung zuständig ist. Um entscheiden zu können, welchem Typ ein Verbindungswunsch entspricht, verwaltet der Super-Server eine Konfigurationstabelle mit Nummern, denen entsprechende Server-Programme zugeordnet sind. Die Nummern werden als *Portnummern* oder einfach als Ports bezeichnet. Im Datensatz von IP-Paketen wird, wie wir im vorherigen Abschnitt gesehen haben, die Portnummer mit übertragen. Die gängigen Portnummern sind natürlich standardisiert. Der Standard basiert auf den Vorschlägen in der RFC 790 (nachlesbar unter *http://www.faqs.org/rfcs/rfc790.html*). Die heute maßgebliche Liste wird unter *http://www.iana.org/assignments/port-numbers* von der Internetorganisation IANA veröffentlicht.

Für hochfrequentierte Server ist diese Arbeitsweise von *inetd* jedoch keine gute Lösung, da sie andauernd wieder neu gestartet und geladen werden müssten. Deshalb werden Server-Typen wie Webserver, Datenbankserver, oft auch FTP-Server, Mailserver usw. in der heutigen Praxis als separate Daemons gestartet. Sie überwachen auch teilweise selbstständig bestimmte Ports. Auch dabei gelten die standardisierten Portnummern. Standard-Port für Webverbindungen ist beispielsweise die Nummer 80 und für SSH-Verbindungen die Nummer 22. In der Konfiguration der jeweiligen Server lassen sich teilweise abweichende oder zusätzliche Portnummern einrichten. So kommt es bei Webservern häufig vor, dass diese an zusätzlich eingerichteten Portnummern wie beispielsweise 8081 »lauschen«.

Die nachfolgende Tabelle listet typische Portnummern und damit verbundene Internetdienste auf:

Portnummer	Internetdienst
21	File Transfer Protocol (FTP)
22	SSH (Secure Shell)
23	Telnet
25	Simple Mail Transfer Protocol (SMTP)
42	Hostname-Service
43	Whois-Service
53	Domain Name Service
70	Gopher-Protokoll
79	finger-Protokoll
80	HTTP-Protokoll (WWW)
110	POP3-Protokoll (Mailempfang)
119	Network News Transfer Protocol (NNTP) für Newsgroups
123	Network Time Protocol (NTP) für netzwerkübergreifenden Zeitabgleich
143	Interactive Mail Access Protocol (IMAP)
194	Internet Relay Chat (IRC)
3306	MySQL-Datenbankserver

Tabelle 9.2: Wichtige typische Portnummern und ihre Dienste

Weiterführender Link:

Client-/Server-Architektur und Internet:
http://www.server-client.com/

9.3.3 Das HTTP-Protokoll

Das Internet ist vorwiegend auf Client-Server-Kommunikation ausgelegt. Für alle entsprechend strukturierten Dienste gibt es Client-Programme, die mit einem geeigneten Server kommunizieren können. Beim Internetdienst »World Wide Web« ist das beispielsweise der Webbrowser, der mit einem Webserver kommunizieren kann. Beim Internetdienst »E-Mail« ist es ein E-Mail-Programm (Client), das mit einem Mailserver kommuniziert. Die Arbeit eines Servers besteht darin, Client-Anfragen für »seinen« Internetdienst zu verarbeiten und den Client mit Antworten oder angeforderten Daten zu beliefern.

Für den Kommunikationsaustausch zwischen Client und Server gibt es so genannte ***Anwendungsprotokolle***. Das Standard-Anwendungsprotokoll für den Dienst »World Wide Web« ist das HTTP-Protokoll.

Das alles geschieht oberhalb von TCP/IP. Aus Sicht der IP-Datenpakete gehören Kommandos des HTTP-Protokolls bereits zu den »Nutzdaten«. Damit Browser und Server überhaupt erkennen, was von den über TCP/IP übertragenen Nutzdaten zum Anwendungsprotokoll HTTP gehört und was zu den damit übertragenen Inhalten, gibt es eine Regel. Am Anfang einer Datenübertragung zwischen Browser und Server stehen stets die HTTP-Protokollinhalte. Erst wenn zwei Zeilenumbrüche in Folge vorkommen, signalisiert dies das Ende des HTTP-Protokollbereichs. Dahinter folgen die Daten des Inhalts, also z. B. der HTML-Code einer angeforderten Webseite.

Ursprünglich entworfen wurde HTTP ebenso wie HTML von Web-Gründervater Tim Berners-Lee. Und ebenso wie bei HTML hat schließlich das W3-Konsortium die Weiterentwicklung des Standards in die Hand genommen. Aus dem Engagement gingen die Standards 1.0 und 1.1 des HTTP-Protokolls hervor. Beide Versionen werden noch aktiv im Web verwendet. Die aktuellere Protokollversion 1.1 wird jedoch immer häufiger eingesetzt und die Version 1.0 immer seltener. Die HTTP-1.1-Spezifikation ist in der RFC 2616 festgelegt.

Links zum HTTP-Protokoll:

RFC 2616 (HTTP-Protokollspezifikation 1.1):
http://www.faqs.org/rfcs/rfc2616.html
RFC 1945 (HTTP-Protokollspezifikation 1.0):
http://www.faqs.org/rfcs/rfc1945.html
RFC 2660 (HTTPS-Protokollspezifikation):
http://www.faqs.org/rfcs/rfc2660.html
HTTP Working Group:
http://ftp.ics.uci.edu/pub/ietf/http/
HTTP-Dokumentation:
http://www.html-world.de/program/http_ov.php

Szenario, Teil 1: Browser ruft Webseite auf

Der HTTP-Protokollbereich besteht aus einem HTTP-Kommando. Im Anschluss daran können diverse Angaben, die aus einem Namen und einem Wert bestehen, angehängt werden.

Wenn ein Webbrowser beispielsweise den URI *http://www.example.org/* aufrufen soll, baut er zunächst über TCP die Verbindung zu *example.org* auf und versendet dann über das IP-Protokoll folgende Daten:

```
GET / HTTP/1.1
```

Wenn der Webserver diese Daten empfängt, weiß er, dass es HTTP-Protokolldaten sind, denn Am Anfang aller empfangenen Daten steht immer der HTTP-Protokollbereich. Er beginnt folglich damit, das Kommando zu interpretieren, und kann anschließend wie gewünscht reagieren.

Das Kommando besteht aus dem Kommandonamen GET, dem Webverzeichnispfad der verlangten Daten (im Beispiel nur `/`, was einfach die Startseite im Webverzeichnis bedeutet), und der HTTP-Version, die der Browser zum Absetzen dieses Kommandos benutzt. Das GET-Kommando bedeutet: Der Browser fordert Daten an, hat aber selbst keine Daten zu senden.

Meistens bleibt es allerdings nicht beim reinen Kommando. In weiteren Zeilen kann ein so genannter HTTP Message Header folgen. Dieser besteht aus Feldern, die den typischen Aufbau `Bezeichner: Wert` haben. Der Browser gestaltet einen GET-Request an einen Webserver in der Praxis üblicherweise eher so:

```
GET / HTTP/1.1
ACCEPT-ENCODING: gzip,deflate
CONNECTION: keep-alive
ACCEPT:text/xml,application/xml,application/xhtml+xml,text/html;
       q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
ACCEPT-CHARSET: ISO-8859-1,utf-8;q=0.7,*;q=0.7
USER-AGENT: Mozilla/5.0 (Windows; U; Windows NT 5.1; de-DE;
            rv:1.7.5) Gecko/20041122 Firefox/1.0
ACCEPT-LANGUAGE: de,en-us;q=0.7,en;q=0.3
KEEP-ALIVE: 300
HOST: www.example.org
```

Alle Felder des Message Header beginnen mit dem groß geschriebenen Bezeichner. Hinter dem Doppelpunkt folgt der zugewiesene Wert. Welche Bezeichner möglich sind, legt das HTTP-Protokoll fest. Ebenso legt es für diverse Bezeichner fest, welchem Schema der zugewiesene Wert entsprechen muss.

Im Beispiel ist unter anderem erkennbar, dass der anfordernde Browser bereit ist, die angeforderten Daten auch in komprimierter Form (nach gzip- oder deflate-Standard) zu erhalten, dass er die angeforderten Daten in Form von Text, HTML, XHTML oder XML direkt verarbeiten kann und dass er als Zeichenkodierung den Zeichensatz ISO-8859-1 oder die Kodierung nach UTF-8 akzeptiert. Der Browser gibt sich aus als ein Mozilla 5.0,

der unter Windows NT 5.1 läuft und eine deutschsprachige Benutzeroberfläche hat. Weiter hinten verrät er dann noch, dass er der Gecko-Familie angehört, am 22.11.2004 kompiliert wurde und eigentlich ein Firefox 1.0 ist. Diese Angaben zum User-Agent sind historisch gewachsenes Kraut-und-Rüben-Durcheinander. Einige Browser erlauben es dem Anwender auch, über die Browser-Bedienoberfläche oder in der Browser-Konfiguration eine User-Agent-Angabe nach Belieben einzugeben. Entsprechend lustige bis hirnrissige Daten finden sich dann in den HTTP-Protokolldaten wieder, die der Browser erzeugt.

Szenario, Teil 1: Webserver liefert Antwortdaten an Browser

Der empfangende Webserver analysiert Kommando und Header. Auf Grund der Anfrage im Beispiel ist klar, dass die Webseite auf jeden Fall gesendet werden muss. Mit Hilfe eines Header-Felds namens `IF-MODIFIED-SINCE` hätte der Browser auch darauf hinweisen können, dass er die aktuelle Version der Webseite nur möchte, wenn deren Daten nach dem angegebenen Datum geändert wurden. Das könnte er beispielsweise tun, weil er seit jenem Zeitpunkt noch eine Version der Seite in seinem Cache-Speicher hat.

Der Server lädt die angeforderte Seite. Nehmen wir an, es handelt sich um eine statische HTML-Datei namens *index.htm*, die sich im Webverzeichnis befindet und folgenden Inhalt hat:

```
<html>
<head>
<title>Example Web Page</title>
</head>
<body>
<p>You have reached this web page by typing &quot;example.com&quot;,
&quot;example.net&quot;,
or &quot;example.org&quot; into your web browser.</p>
<p>These domain names are reserved for use in documentation and are not
available for registration. See
<a href="http://www.rfc-editor.org/rfc/rfc2606.txt">RFC
2606</a>, Section 3.</p>
</body>
</html>
```

Wie bereits erwähnt werden die HTTP-Protokolldaten vor dem ausgelieferten Inhalt eingefügt. Was der Browser am Ende vom Webserver von *example.org* empfängt, könnte etwa so aussehen:

```
RESPONSE HTTP/1.1 200 OK
DATE: Wed, 16 Feb 2005 16:21:14 GMT
SERVER: Apache/1.3.27 (Unix) (Red-Hat/Linux)
LAST-MODIFIED: Wed, 08 Jan 2003 23:11:55 GMT
ETAG: "3f80f-1b6-3e1cb03b"
ACCEPT-RANGES: bytes
CONTENT-LENGTH: 438
CONTENT-TYPE: text/html

<html>
```

```
<head>
<title>Example Web Page</title>
</head>
<body>
<p>You have reached this web page by typing &quot;example.com&quot;,
&quot;example.net&quot;,
or &quot;example.org&quot; into your web browser.</p>
<p>These domain names are reserved for use in documentation and are not
available for registration. See
<a href="http://www.rfc-editor.org/rfc/rfc2606.txt">RFC
2606</a>, Section 3.</p>
</body>
</html>
```

Das GET-Kommando wird vom Webserver mit einem RESPONSE-Kommando beantwortet. Das RESPONSE-Kommando nennt hinter dem Kommandonamen die verwendete HTTP-Version und dahinter einen so genannten ***HTTP-Statuscode***. Der Code `200 OK` bedeutet, dass die angeforderte Seite gefunden und geladen wurde und dass die Daten im Anschluss an den HTTP-Protokollbereich übertragen werden.

Die HTTP-Statuscodes sind sehr wichtig, da sie die möglichen Reaktionen des Webservers abdecken. Einer der bekanntesten Statuscodes, den auch viele Anwender im Web aus leidvoller Erfahrung kennen, ist der Statuscode `404 NOT FOUND`, der geliefert wird, wenn eine angeforderte Seite nicht existiert. Nachfolgende Tabelle listet wichtige HTTP-Statuscodes auf:

Code-Nr.	Code-Meldung	Bedeutung
200	OK	Client-Anfrage war in Ordnung, angeforderte Daten werden nachfolgend übertragen.
204	NO CONTENT	Client-Anfrage war in Ordnung. Es werden jedoch keine Daten an den Client gesendet. Im Browser bleibt die zuletzt aufgerufene Seite am Bildschirm stehen.
301	MOVED PERMANENTLY	Die angeforderten Daten liegen unter einer neuen Adresse. Die Adresse teilt der Server im Message Header mit. Der Browser kann z. B. eine erneute Anfrage für die andere Adresse starten.
302	MOVED TEMPORARILY	Die angeforderten Daten liegen vorübergehend unter einer neuen Adresse. Die Adresse teilt der Server im Message Header mit. Der Browser kann z. B. eine erneute Anfrage für die andere Adresse starten.
304	NOT MODIFIED	Die angeforderten Daten haben sich seit dem vom Browser bei der Anfrage angegebenen Zeitpunkt nicht geändert und werden deshalb nicht gesendet. Der Browser kann die Daten stattdessen aus seinem Cache holen. In der Praxis ist dies der häufigste Code neben 200.
400	BAD REQUEST	HTTP-Syntaxfehler in der Client-Anfrage.

Tabelle 9.3: Wichtige HTTP-Statuscodes

Code-Nr.	Code-Meldung	Bedeutung
401	UNAURHORIZED	Die angeforderten Daten sind zugangsgeschützt. Der Server kann die Daten nur senden, wenn eine gültige Zugangskennung, bestehend aus Benutzername und Passwort, bei der Anfrage mit gesendet wird.
403	FORBIDDEN	Die angeforderten Daten sind für den anfordernden Client oder generell nicht zugänglich.
404	NOT FOUND	Die angeforderten Daten wurden nicht gefunden.
505	INTERNAL SERVER ERROR	Der Server kann die angeforderten Daten nicht senden, weil auf dem Server ein Fehler aufgetreten ist. Beispielsweise konnte ein aufgerufenes Script nicht gestartet werden oder es hat keine Daten zum Senden erzeugt.

Tabelle 9.3: Wichtige HTTP-Statuscodes (Fortsetzung)

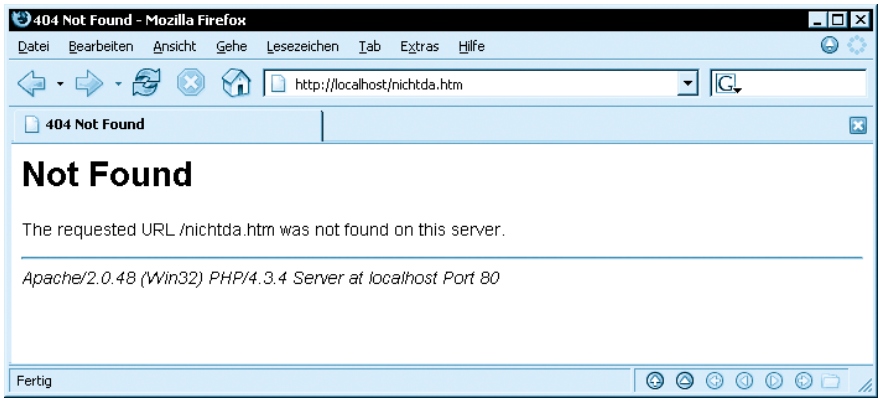


Abbildung 9.10: Typische HTTP-Statusmeldung 404 NOT FOUND im Browser

Die HTTP-Statuscodes der Serien 400 und 500 stellen vom Webserver erzeugte Fehlermeldungen dar. Zur Konfiguration des Webservers gehört ein Set von HTML-Dateien, in denen die Fehlermeldungen HTML-formatiert enthalten sind. Der Server liefert bei Auftreten eines Fehlers die entsprechende HTML-Datei anstelle der eigentlich angeforderten Daten an den Browser. Auf diese Weise erscheinen im Browser Seiteninhalte wie der oben abgebildete.

Es gehört übrigens zum guten Stil einer professionellen Website, die Fehlermeldungsseiten für den Browser so zu gestalten, dass sie im Seitenlayout des Projekts erscheinen. Wie das genau funktioniert, werden wir in ***Abschnitt 11.6*** zeigen.

HTML und HTTP

Wenn Sie bis hierhin durchgedrungen sind, sind Sie bereits in der Lage, HTML einmal aus serverseitigem Blickwinkel zu betrachten. Folgende Punkte sind wichtig zu wissen:

- **Erneute HTTP-Kommunikation für jede Ressource:** Nachdem ein Browser vom Webserver die Daten einer angeforderten Adresse erhält, handelt es sich dabei in der Regel erst einmal um HTML-Code. Der Browser parst den HTML-Code. Dieser kann jedoch

etliche Grafikreferenzen, Referenzen von externen CSS- oder JavaScript-Dateien, referenzierte Flash-Movies usw. enthalten. Oder es handelt sich um eine Frameset-Datei, die nur Angaben enthält, welche anderen Inhalte in die einzelnen Frame-Fenster geladen werden sollen. Nachdem der Browser den HTML-Code geparst hat (oder noch während er ihn parst), startet er für jede referenzierte Datei eine erneute GET-Anfrage an den Webserver. Wenn in einem HTML-Dokument also zehn verschiedene Grafikdateien referenziert sind, sind zehn weitere vollständige GET-RESPONSE-Kommunikationen zwischen Browser und Server erforderlich. Aus dieser Sicht wird möglicherweise verständlicher, warum Experten immer wieder vor Verwendung unnötig vieler Grafiken warnen.

- **RESPONSE-Beeinflussung in Meta-Tags:** In Zusammenhang mit HTML wurden auch Meta-Angaben des Typs `<meta http-equiv=... content=...>` behandelt. Was auf diese Weite im HTML-Dokument notiert ist, sollte ein Webbrowser, der das Dokument von einem Webserver als Antwort erhält, als zusätzliche Angaben zum Message Header interpretieren, den er vom Server mit dem Dokument erhalten hat. Nehmen wir folgendes HTML-Markup an:

```
<meta http-equiv="expires" content="Wed, 26 Feb 2005 09:12:21 GMT">
```

Der Browser wird damit aufgefordert, dies so zu interpretieren, als ob er vom Server folgenden HTTP-Response erhalten hätte:

```
HTTP/1.1 200 OK
EXPIRES: Wed, 26 Feb 2005 09:12:21 GMT
```

Meta-Tags mit http-equiv-Attribut sollen also dem HTML-Autor die Möglichkeit bieten, die Anweisungen des Servers an den Browser beim Ausliefern des Dokuments zu beeinflussen. Es liegt jedoch an den Browsern, ob sie http-equiv-Anweisungen so ernst nehmen wie echte HTTP Message Header. Viele Browser tun das leider nicht oder zumindest nicht bei allen http-equiv-Meta-Tags oder nur mit Einschränkungen. Um als Anbieter von Webseiten echte HTTP Message Header an den Browser zu übertragen, bleibt nur der Weg über Scriptsprachen wie PHP. Diese erlauben es, vor dem eigentlichen HTML-Code auch HTTP Message Header zu senden.

- **GET- und POST-Methode:** Im einleitenden `<form>`-Tag können Sie beim method-Attribut festlegen, mit welcher Methode die Formulardaten an den Server übertragen werden. Möglich sind die Angaben `get` oder `post`. Wenn Sie `get` angeben, fordert der Browser die im action-Attribut angeforderte Seite beim Webserver nach Absenden des Formulars mit dem GET-Kommando von HTTP an. Wenn Sie `post` wählen, benutzt er stattdessen das POST-Kommando. Der Unterschied ist, dass beim GET-Kommando nichts anderes übertragen wird als das HTTP-Kommando selbst und die Message Header. Deshalb müssen die Formulardaten in diesem Bereich untergebracht werden. Wie bekannt, werden sie im URI der angeforderten Adresse als so genannter GET-String untergebracht. Bei der POST-Methode dagegen können die Daten im Anschluss an HTTP-Kommando und Message Header gesendet werden, genauso, wie umgekehrt im Anschluss an den HTTP-Protokollbereich vom Server HTML-Code oder Inhalte anderer Dateien an den aufrufenden Browser gesendet werden.

9.4 Der Webserver Apache

Webserver sind wie alle Server Softwareprodukte – auch wenn sie meistens keine eigene Anwendungsoberfläche besitzen. Während manche Server jedoch vergleichsweise schlicht sind, ist ein Webserver ein recht komplexes Produkt, das etliche Aufgaben zu erfüllen hat und an das extrem hohe Anforderungen gestellt werden, was die Laufstabilität betrifft. Die reine HTTP-Kommunikation ist längst nicht das einzige, was ein moderner Webserver zu beachten hat. Er muss auch serverseitige Scripts ausführen können, er muss Softwaremodule von Drittanbietern einbinden können und er muss dem Websiteanbieter zahlreiche und verzweigte Konfigurationsmöglichkeiten anbieten, die helfen, das Webangebot zu sichern und zu optimieren. Der Softwareentwicklungsaufwand für ein führendes Webserver-Produkt ist daher mindestens so hoch wie der für ein Office-Paket oder eine anspruchsvolle Grafik-Software.

Gemeinsam mit dem Boom, der durch die ersten Versionen des Netscape-Browsers Mitte der 90er Jahre ausgelöst wurde, begannen auch die Webserver von Netscape zu dominieren. Die Lizenzpolitik von Netscape sowie technische Eigenmächtigkeiten sorgten jedoch dafür, dass die Server der so genannten Enterprise-Serie bald wieder an Bedeutung verloren. Wie auch auf dem Browser-Markt, schien damit die Stunde für Microsoft geschlagen zu haben. Doch während am Browser-Markt der Internet Explorer alsbald zum meist verwendeten Produkt wurde, entwickelte sich der Server-Markt in eine andere Richtung.

Ein Grund dafür ist, dass Microsoft-Betriebssysteme, so beliebt sie für Desktop-PCs sind, unter Experten, die im Server-Bereich tätig sind, weniger Achtung genießen. Ein Server-Rechner sollte so weit wie möglich aus »transparenter« Software bestehen. Und genau das verhinderte und verhindert Microsofts Lizenzpolitik bis heute. Ganz anders dagegen das Linux-Projekt. Vom Betriebssystemkern über die zahlreichen Dienstprogramme bis hin zu wichtigen Anwendungen basiert in der Linux-Welt die Mehrzahl der Software auf dem Open-Source-Prinzip und jeder Funktionsaufruf ist offen dokumentiert. Außerdem steht Linux in der Tradition der Unix-Philosophie, deren typische und zuverlässig funktionierende Basis-Features wie Ein-/Ausgabekanäle, Prozess-Forking, Signalisierung, Piping usw. gerade im Server-Betrieb sehr gefragt sind.

So konnte sich im Laufe der Jahre ein betriebssystemübergreifendes, ambitioniertes Webserver-Projekt immer stärker etablieren: das Apache-Projekt. Auch der Apache ist ein Open-Source-Produkt. Mit zwei Dritteln Marktanteilen unter allen im heutigen Web ansprechbaren Webservern (Tendenz steigend) dominiert er klar und setzt die Standards.

Der Erfolg des Apache Webservers hängt allerdings auch mit dem Elan seiner Entwicklergemeinde zusammen. Nur wenige Open-Source-Projekte werden so fieberhaft und gewissenhaft weiterentwickelt und ständig optimiert wie der Apache Webserver. Seinen zahlreichen Zwischenversionen (***patches*** genannt) soll er inoffiziellen Quellen zufolge sogar seinen Namen verdanken (***a patchy server*** ...).

Links zum Apache Webserver:

Apache Software Foundation:
http://www.apache.org/
Dokumentation Apache 1.3 (en):
http://httpd.apache.org/docs/
Dokumentation Apache 2.0 (de):
http://httpd.apache.org/docs-2.0/de/
Dokumentation Apache 2.1 (de):
http://httpd.apache.org/docs-2.1/de/

9.4.1 Versionen, Bezug und Installation von Apache

Wenn Sie ein typisches Web-Hosting-Angebot unterschreiben, brauchen Sie sich um die Installation eines Webservers normalerweise keine Gedanken mehr zu machen. Im Fall von Virtual-Hosting-Angeboten haben Sie ohnehin wenn überhaupt nur Einfluss auf dezentrale Konfigurationsmöglichkeiten des Webservers, bei Root-Servern finden Sie auf jeden Fall eine Standardinstallation vor, die in der Regel auch schon so konfiguriert ist, dass sie mit anderen installierten Produkten wie PHP zusammenarbeitet.

Gerade wegen der intensiven Weiterentwicklung des Apache kann es jedoch für Mieter eines Root-Servers sinnvoll sein, hin und wieder ein Upgrade auf eine aktuellere Version aufzuspielen. Außerdem empfiehlt es sich bei der Entwicklung professioneller Sites dringend, lokal oder auf einem Testserver im Web das gesamte Webprojekt unter Server-Bedingungen lauffähig zu bekommen. Selbst wenn Produktiv- und Entwicklungsumgebung nicht hundertprozentig gleich sind, so hilft es auf jeden Fall, in der Entwicklungsumgebung einen Apache Webserver zu installieren, sofern in der Produktivumgebung ebenfalls ein solcher Server am Werk ist.

Download und Versionen

Die Download-Adresse für den Apache lautet *http://httpd.apache.org/download.cgi*. Darüber hinaus bieten fast alle Linux- Distributionen und andere Anbieter von Unix-Derivaten eigene Download-Möglichkeiten an. Auch Anbieter von Paketen wie LAMP (Apache-MySQL-PHP für Linux) oder WAMP (Apache-MySQL-PHP für Windows) bieten eigene Downloads an.

Vom Apache Webserver existieren derzeit zwei Versionenzweige: eine 1.3-Serie und eine 2.0-Serie. Beide werden noch aktiv im Web verwendet. Einige Server-Betreiber haben auch gute Gründe, nicht auf die neuere Versionslinie 2.0 umzusteigen. Gründe sind beispielsweise installierte Module, die es für die Version 2.0 noch nicht gibt. Da aber das Modul-Handling in Version 2.0 anders ist als in Version 1.3, würden die bisherigen Module unter der neueren Apache-Version nicht mehr laufen.

Beachten Sie auch, wenn Sie außerdem noch PHP als Scriptsprache in Verbindung mit Apache einsetzen wollen, einen Warnhinweis der PHP-Entwickler: Dort wird dringend davon abgeraten, PHP in Verbindung mit Apache 2.0 zu verwenden. Der Grund ist, dass

Apache 2 so genannte *Threads* benutzt, um seine Arbeit effizienter zu verrichten. Wenn PHP als Apache-Modul installiert wird und damit Speicherbereiche des Apache nutzt, kann es zu Speicherkonflikten kommen, da viele PHP-Funktionen nicht »thread-save« sind. In der Praxis berichten jedoch viele Anwender, dass das Zusammenspiel zwischen Apache 2 und PHP problemlos funktioniert. Auf vielen öffentlichen Webservern mit PHP läuft jedoch wegen dieser Warnung meist noch eine Apache-1.3-Version.

Download und Installation unter Unix/Linux

Vom Apache-Server gibt es für Unix-/Linux-Derivate eine unkompilierte Version (auf der Download-Seite im Archiv-und-gZip-Format *tar.gz* angeboten). Daher muss der Server auf solchen Plattformen nach dem Download nicht nur installiert, sondern auch für die Betriebssystemumgebung kompiliert werden. Das standardisierte Verfahren der Makefiles macht es jedoch einfach, die Installation mit den eingestellten Defaults durchzuführen.

Die nachfolgende Beschreibung setzt voraus, dass Sie sich auf dem Rechner, auf dem der Apache installiert werden soll, eingeloggt haben auf Shell-Ebene arbeiten können und hinreichend Rechte besitzen, um neue Software zu installieren. Auf einem entfernten Server-Rechner genügt es, sich mit einem SSH-Client wie PuTTY eingeloggt zu haben.

Um den Download direkt von der Shell-Ebene zu starten, wechseln Sie z. B. in Ihr Home-Verzeichnis. Dann können Sie das wget-Kommando benutzen. Beispiel:

```
wget http://apache.engram.de/apache.org/httpd/httpd-2.0.NN.tar.gz
```

Das `NN` steht hierbei für eine Zwischenversion des Apache. Den genauen Download-URI der tar.gz-Datei sollten Sie vorher im Browser ermitteln.

Beachten Sie, dass der Apache 2.0 während der Installation wenigstens 50 Mbyte freien Festplattenspeicher benötigt. Überprüfen können Sie den belegten und verfügbaren Speicherplatz durch Eingabe des folgenden Kommandos:

```
df -h
```

Stellen Sie vor der Installation sicher, dass ein Ansi-C-Compiler auf dem System installiert ist. Das Standardprodukt auf Linux-Distributionen ist der GNU C-Compiler GCC. Über die Shell können Sie beispielsweise eingeben:

```
gcc --help
```

Wenn dann Versionsinformationen und Aufrufmöglichkeiten zu GCC erscheinen, ist alles in Ordnung.

Nach dem Download kann die tar.gz-Datei durch folgendes Kommando dekomprimiert werden (NN steht für eine Zwischenversionsnummer des Apache):

```
gzip -d httpd-2_0_NN.tar.gz
```

Anschließend kann die so entpackte Archivdatei *httpd-2_0_NN.tar* entpackt werden:

```
tar xvf httpd-2_0_NN.tar
```

Als Nächstes sind grundsätzliche Konfigurationen für die Installation nötig. Die meisten Einstellungen können jedoch ebenso gut auch später vorgenommen werden. Entschieden werden sollte vor der Installation jedoch der Ablageort. Per Default wird Apache 2.0.x in das Verzeichnis */usr/local/apache2* installiert. Wenn Sie dies ändern möchten, können Sie beispielsweise Folgendes eingeben:

```
./configure --prefix=/usr/local/bin/apache
```

In diesem Beispiel wird der Apache im Verzeichnis */usr/local/bin/apache* installiert.

Wenn Sie gar keine Optionen konfigurieren möchten, starten Sie einfach nur `./configure` ohne Parameter. Das Konfigurationsscript benötigt eine Weile für seine Arbeit. Anschließend kann der Kompilierungsvorgang gestartet werden:

```
make
```

Nachdem dieser Vorgang abgeschlossen ist, kann die Installation gestartet werden:

```
make install
```

Neben dieser unkompilierten Standarddistribution gibt es zahlreiche vorkompilierte oder in umgebungsspezifischen Paketformaten gespeicherte Distributionen für bestimmte Linux-Varianten, FreeBSD usw. Informationen und Downloads sind meist über die Webseiten zur Distribution erreichbar.

Download und Installation unter MS Windows

Es kommt in der Praxis nur selten vor, dass ein Apache-Webserver auf einem Windows-Betriebssystem für einen öffentlichen Webserver installiert wird. Es wird auch ausdrücklich davor gewarnt, dies mit älteren Windows-Versionen zu tun, die nicht auf NT-Technologie basieren, da die Sicherheitslücken dort einfach zu gravierend sind.

Wenn Sie Ihre Webseiten unter MS Windows entwickeln, ist es jedoch ratsam, sich Apache auch lokal zu installieren. Das hat verschiedene Vorteile:

- Sie können alle Seiten unter HTTP-Bedingungen testen.
- Sie können in HTML die absolute lokale Referenzierung nutzen (wie z.B. `src="/pics/fotos/bild1.png"`) und sich dabei wie in der Produktivumgebung auf das Web-Wurzelverzeichnis beziehen statt auf das Wurzelverzeichnis des aktuellen Laufwerks.
- Sie schaffen damit die Umgebungsbasis, die Sie für Webseiten mit PHP und MySQL ohnehin benötigen werden.

Über die Download-Seite von Apache können Sie »Binaries«, also vorkompilierte Versionen für MS Windows, herunterladen. Sollte dort beispielsweise keine aktuelle Version für Apache 2.0 verfügbar sein, finden Sie aktuelle Versionen auch über Download-Sites wie die von ZDNet (*http://www.zdnet.de/downloads/*).

Das Installationsprogramm (.exe oder .msi) kann ausgeführt werden. Zur Installation genügt es, der dialogorientierten Benutzerführung zu folgen. Wenn keine besonderen Gründe dagegen sprechen, ist es zweckmäßig, alle im Dialogverlauf angebotenen Defaults zu akzeptieren.

9.4.2 Start, Stopp und Neustart von Apache

Unter Unix-/Linux-Systemen ist im Normalfall root-Berechtigung erforderlich, um den Webserver zu starten. Der Grund ist, dass Apache den als wichtig erachteten Port 80 belegt. Die ausführbare Programmdatei heißt auf Unix-/Linux-Systemen *httpd* und unter Windows *Apache.exe*.

Unter Unix/Linux wird jedoch empfohlen, zum Starten, Stoppen und Neustarten des Servers nicht direkt die ausführbare Serverdatei zu verwenden, sondern das mit installierte Shellscript *apachectl*. Es befindet sich im gleichen Verzeichnis wie die ausführbare httpd-Datei, auffindbar über folgendes Shell-Kommando:

```
find / -name apachectl
```

Wechseln Sie in das entsprechende Verzeichnis oder starten Sie das Script mit vollem Pfadnamen. Die Kommandos zum Starten, Stoppen und Neustarten lauten:

```
[Pfad/]apachectl start
[Pfad/]apachectl stop
[Pfad/]apachectl restart
```

Wenn nach Eingabe des Kommandos nach kurzer Zeit wieder der Shell-Eingabeprompt erscheint, war die Aktion erfolgreich.

Unter MS Windows wird Apache seit Version 2.0 in aller Regel gleich bei der Installation als Service (Dienst) eingerichtet. Ein kleines Monitor-Tool wird dabei installiert, das als Icon im System Tray (kleine Symbole in der Taskleiste neben der Uhrzeitanzeige) erscheint:



Abbildung 9.11: Icon des Apache 2.0 Monitor Tools unter MS Windows

Ein Doppelklick auf das Icon öffnet den Monitor. Die Schaltflächen *Start*, *Stop* und *Restart* lösen die drei entsprechenden Operationen aus.

9.4.3 Test von Webseiten unter Apache

Auf typischen Server-Rechnern im Internet, zu dem Sie über ein Web-Hosting-Vertrag Zugang haben, werden Sie bereits alles fertig eingerichtet vorfinden. Der Provider sollte Ihnen im Vertrag auch mitteilen, wo im Verzeichnisbaum sich die so genannte Document Root befindet, also das Startverzeichnis für eigene Webseiten.

Wir nehmen hier deshalb an, Sie haben einen Apache lokal auf Ihrem Windows-PC installiert und möchten nun testen, ob alles funktioniert. Gehen Sie dazu wie folgt vor:

1. Legen Sie ein gewünschtes Verzeichnis an, unterhalb dessen Sie HTML-Dateien und andere Dateien Ihres lokalen Webs ablegen wollen. Es ist egal, wo sich dieses Verzeichnis befindet. Es kann sich auch im LAN auf einem über Laufwerksbuchstaben ansprechbaren Verzeichnis befinden.
2. Suchen Sie im Programmverzeichnis von Apache (z. B. *C:\Programme\Apache\Apache 2*) im Unterverzeichnis *conf* nach der Datei *httpd.conf*.
3. Öffnen Sie die *httpd.conf* mit einem Texteditor (z. B. mit dem Windows Notepad-Editor).
4. Suchen Sie nach einer Zeile, die mit `DocumentRoot` beginnt. Notieren Sie dahinter durch Leerzeichen getrennt den vollständigen Pfad zu dem Verzeichnis, in dem die Startseite Ihres lokalen Webs liegen soll. Trennen Sie dabei Verzeichnisangaben durch einfachen Schrägstrich. Erlaubt sind Angaben wie `/Web` oder `f:/daten/hannes/http`.
5. Speichern Sie die Änderungen in *httpd.conf*.
6. Starten Sie den Apache Webserver wie im vorherigen Abschnitt beschrieben neu.
7. Erzeugen Sie nun im angegebenen Web-Startverzeichnis eine HTML-Datei oder kopieren Sie eine vorhandene HTML-Datei in das Verzeichnis und nennen Sie sie *test.htm*.
8. Starten Sie einen Webbrowser und geben Sie in der Adresszeile ein:

```
http://127.0.0.1/test.htm
```

Wenn alles korrekt eingerichtet ist, erscheint die angegebene HTML-Datei als über HTTP aufgerufene Webseite im Browserfenster. Damit haben Sie einen funktionierenden lokalen Apache Webserver auf Ihrem PC.

9.4.4 Die zentrale Konfigurationsdatei httpd.conf

Nach jedem Start oder Neustart liest Apache zuerst seine zentrale Konfigurationsdatei ein. Das bedeutet andersherum, dass der Server nach jeder Änderung in dieser Datei neu gestartet werden muss, damit die Konfigurationsänderung wirksam wird.

Die zentrale Konfigurationsdatei von Apache hat den Namen *httpd.conf*. Sie ist im Installationsunterverzeichnis *conf* abgelegt. Wenn Sie unter Linux auf Shell-Ebene nach der Datei suchen möchten, können Sie eingeben:

```
find / -name httpd.conf
```

Um sich erst einmal den Inhalt anzeigen zu lassen, ohne etwas zu ändern, können Sie das more-Kommando oder, wenn verfügbar, das noch komfortablere less-Kommando verwenden:

```
more [Pfad/]httpd.conf
less  [Pfad/]httpd.conf
```

Zum Bearbeiten der Konfigurationsdatei müssen Sie einen auf dem System verfügbaren Texteditor starten. Fast überall verfügbar ist auf jeden Fall der vi-Editor:

```
vi [Pfad/]httpd.conf
```

Unter MS Windows wechseln Sie in das Programmverzeichnis des Apache (z.B. *C:/Programme/Apache/Apache 2*) und dort ins Unterverzeichnis *conf*. Die Datei *httpd.conf* können Sie mit jedem Texteditor öffnen (Datei selektieren, mit rechter Maustaste Kontextmenü öffnen und *Öffnen mit...* wählen).

Aufbau der zentralen httpd.conf

Die meisten Zeilen der Datei beginnen mit einem Gatterzeichen (#). Das sind Kommentarzeilen, die vom Apache Webserver ignoriert werden. Da die Konfigurationsmöglichkeiten nicht selten weitreichende Konsequenzen haben, ist die Datei üppig mit erklärenden Kommentaren ausgestattet.

Der genaue Inhalt der Datei variiert zwischen verschiedenen Apache-Versionen. Generell ist sie jedoch stets in drei entsprechend ausgewiesene Hauptabschnitte (Sections) unterteilt.

Section 1 betrifft das »global Environment«, also die allgemeine Server-Umgebung, Section 2 die 'Main' server configuration, also Konfigurationsdaten, die für alle Apache-Prozesse von Bedeutung sind, und Section 3 die Virtual Hosts. Die einzelnen Konfigurationsanweisungen werden als *Direktiven* (*directives*) bezeichnet. Diese haben folgenden Aufbau:

```
Bezeichner Wert
```

Das heißt, sie stehen in einer eigenen Zeile. Die Bezeichnung der Direktive und ihr zugeordneter Wert werden durch ein Leerzeichen voneinander getrennt (es dürfen auch mehrere Leerzeichen sein).

Es gibt auch Gruppen von Direktiven. Diese werden in einer HTML-ähnlichen Weise in Tags eingeschlossen. Schema:

```
<Gruppenbezeichner>
    Bezeichner Wert
    Bezeichner Wert
</ Gruppenbezeichner>
```

Es würde den Rahmen dieses Buchs sprengen, auf alle Einstellmöglichkeiten detailliert einzugehen. Die wichtigsten sollten Sie jedoch kennen und selbstständig korrekt ändern können.

Root-Verzeichnis des Servers

Die Direktive `ServerRoot` hilft dem Server, seine Konfigurationsdaten, Fehlermeldungsdateien, Grafikressourcen usw. zu finden. Ändern Sie den Eintrag nie ohne besonderen Grund! Der Eintrag hat nichts mit der bekannteren Direktive `DocumentRoot` zu tun!

Persistente Verbindungen

Eine wichtige Rolle für die Performance des Apache spielen die persistenten Verbindungen (*persistent connections*). Jeder Browser, der auf einem Apache Webserver eine Webseite aufruft, erzeugt beim Erstaufruf einen neuen Apache-Prozess. Wenn die Direktive `KeepAlive` auf `Off` steht, wird nach jeder Kommunikation Browser-Anfrage > Server-Antwort der Prozess wieder beendet. Steht `KeepAlive` dagegen auf `On`, wird der gestartete Prozess erst beendet, wenn länger keine Kommunikation mit dem gleichen Client stattfindet. Wie viele Sekunden damit gewartet wird, ist bei `KeepAliveTimeOut` einstellbar. Der Wert bedeutet die Anzahl Sekunden, bis ein Apache-Prozess bei persistenter Verbindung wegen Inaktivität wieder beendet wird.

Weiterhin wichtig ist für dieses Feature der Wert zu `MaxKeepAliveRequests`. Der Wert legt fest, wie viele Browser-Anfragen innerhalb einer persistenten Verbindung maximal entgegengenommen werden dürfen, bevor der Prozess beendet wird. Ein Wert von 0 bedeutet unbegrenzt viele erlaubte Requests.

Der Vorteil persistenter Verbindungen ist eine deutlich schnellere Auslieferung von angeforderten Daten von bis zu 50%. Der Nachteil ist, dass die Apache-Prozesse dadurch länger im Arbeitsspeicher bleiben und bei hohen Besucherzahlen den Speicher stärker belasten.

Allgemeiner Timeout

Bei der Direktive `TimeOut` können Sie die Anzahl Sekunden einstellen, die ein Apache-Prozess warten soll, bevor er sich wegen erwarteter, aber bislang nicht eingetroffener Daten beenden soll. Je höher der Wert, desto länger bleibt der Prozess im Arbeitsspeicher. Je niedriger, desto höher die Wahrscheinlichkeit, dass z. B. Browser-Anfragen im »Nirwana« landen, weil ihr zugeordneter Server-Prozess bereits beendet wurde. Dennoch ist es in der Praxis stark belasteter Websites durchaus sinnvoll, diesen Wert vom Standardwert 300 deutlich zu verringern, z. B. auf 60. Dadurch kann der Server-Rechner entlastet werden.

Module

Im ersten Abschnitt der *httpd.conf* findet sich eine längere Liste von `LoadModule`-Einträgen. Viele davon sind per Default mit # auskommentiert, was bedeutet, dass sie beim Starten des Servers nicht geladen werden und nicht verfügbar sind. Jedes nicht auskommentierte Modul wird dagegen geladen und ist verfügbar.

Apache unterscheidet statische und dynamische Module. Statische Module sind vor dem Kompilieren des Servers festlegbar. Was Sie in der *httpd.conf* finden, sind dagegen dynamische Module.

Dynamische Module für Apache 2.0 und 1.x sind inkompatibel. Die Standardmodule, die zur Apache-Distribution gehören, sind an der Endung .so erkennbar. Aktivieren Sie durch Entfernen des Kommentarzeichens # nur diejenigen Module, die Sie wirklich benötigen. Durch Beschränkung auf die wirklich nötigen Module kann ein Apache-Prozess schneller erzeugt werden und belegt weniger Speicher.

Sie können auch selbst Module für Apache schreiben und diese einbinden. Zum Programmieren können Sie beispielsweise Perl benutzen, wenn diese Scriptsprache auf dem Rechner installiert ist. Apache-Module müssen allerdings so geschrieben sein, dass sie konform sind zur Apache-API (*Application Interface*).

Nachfolgende Tabelle listet besonders wichtige und interessante der dynamischen Standardmodule auf:

Modul	Bedeutung
`access_module`	Stellt die Anweisungen »Order allow,deny« bereit, mit denen sich der Zugriff auf Verzeichnisinhalte regeln lässt. Wichtig, wenn Sie z. B. IP-Adressbereiche oder Domains vom Zugriff auf Seiten ausschließen wollen. Ansonsten kann dieses Modul auch deaktiviert werden.
`alias_module`	Erlaubt die Verwendung von Verzeichnissen außerhalb der DocumentRoot, ansprechbar über virtuelle Pfadnamen sowie den Einsatz von ScriptAlias-Namen beispielsweise für CGI- oder PHP-Scripts. Wenn keine Aliasnamen dieser Art benötigt werden, kann dieses Modul deaktiviert werden.
`auth_module`	Ermöglicht Benutzerauthentifizierung und liest Benutzer- und Gruppendaten ein. Zwingend erforderlich, wenn Sie mit passwortgeschützten Seiten arbeiten. Wenn nicht, kann das Modul deaktiviert werden.
`autoindex_module`	Erlaubt Optionen für die Optik von Verzeichnislistings, die vom Server generiert werden. Ist nur von Bedeutung, wenn in einem Verzeichnis Verzeichnis-Browsing erlaubt ist und nur das Verzeichnis angefordert wird, keine Datei darin.
`cgi_module`	Ermöglicht CGI-Scripts in CGI-Verzeichnissen wie z. B. cgi-bin. Wenn keine CGI-Scripts benötigt werden, weil z. B. nur mit PHP als Modul gearbeitet wird, kann dieses Modul deaktiviert werden.
`dir_module`	Ermöglicht das Definieren von Default-Dateinamen für Verzeichnisse, also dass z. B. index.htm ausgeliefert wird, wenn kein Dateiname angegeben wird.
`env_module`	Stellt Umgebungsvariablen für CGI-Scripts und Server-Side-Include-Seiten (SSI) zur Verfügung.
`expires_module`	Erzeugt HTTP Message Header der Typen EXPIRES und CACHE-CONTROL, wenn diese in HTML-Meta-Daten angegeben sind.
`file_cache_module`	Hält eine festlegbare Liste immer wieder benötigter Dateien im Arbeitsspeicher, um sie schneller ausliefern zu können. Ein interessantes Performance-Optimierungs-Modul.
`include_module`	Ermöglicht Server Side Includes (SSI).

Tabelle 9.4: Wichtige und typische dynamische Standardmodule des Apache Webservers

Modul	Bedeutung
log_config_module	Ermöglicht das Mitloggen von Server-Zugriffen. Das Abschalten verhindert die Basis für Webstatistiken.
mime_module	Verwaltet die Zuordnung zwischen Dateinamenerweiterungen und Mime-Typen. Sollte auf keinen Fall deaktiviert werden.
mime_magic_module	Ist in der Lage, den Mime-Type unbekannter Dateinamenerweiterungen zu erkennen, indem der Anfang solcher Dateien eingelesen wird. Der Server versucht dann, anhand des Dateianfangs zu ermitteln, welchem Mime-Type die Datei angehört. Interessantes Feature, das jedoch Performance kosten kann.
negotiation_module	Ermöglicht das Aushandeln von Übertragungsbedingungen zwischen Server und Client. Sollte auf jeden Fall aktiviert sein.
rewrite_module	Ermöglicht es, angeforderte URIs mit anderen URIs auf Basis von festlegbaren Regeln zu überschreiben. Interessantes Feature, wenn z.B. ein Website-Relaunch zu neuen Dateinamen führt. Links oder Bookmarks auf alte Seiten werden dann automatisch auf andere, neue Seiten gelenkt.
setenvif_module	Erlaubt das Setzen von Umgebungsvariablen abhängig von der Art einer Client-Anforderung.
userdir_module	Ermöglicht User-spezifische Verzeichnisse (eine alte NCSA-Server-Spezialität, es handelt sich dabei um Verzeichnisse, deren Namen mit Tilde ~ beginnen).
ssl_module	Ermöglicht den Einsatz der Protokolle Secure Sockets Layer (SSL) und Transport Layer Security (TLS). Basis für verschlüsselte Datenübertragungen zwischen Server und Browser und umgekehrt mit HTTPS-Seiten.

Tabelle 9.4: Wichtige und typische dynamische Standardmodule des Apache Webservers (Fortsetzung)

Bei einigen der Module werden die erforderlichen Detail-Einstellungen in weiteren Einträgen der *httpd.conf* oder in verzeichnisspezifischen *.htaccess-Dateien* vorgenommen, d.h., das Aktivieren des Moduls ist nur ein Schritt – die zugehörige Konfiguration der andere.

Es gibt weitere wichtige Module, die nicht zum Standardumfang von Apache gehören. Für PHP-Programmierer ist das allen voran das Modul, welches es Apache erlaubt, PHP-Scripts direkt und überall (und nicht als CGI-Scripts) auszuführen. Zwar benötigen die einzelnen Apache-Prozesse dadurch einiges mehr an Arbeitsspeicher, doch die Performance ist deutlich höher.

Zentrale Server-Direktiven

Diese Direktiven sollten von jedem Webmaster ordentlich gepflegt und an die Erfordernisse angepasst werden.

Dazu gehört zunächst die Direktive `ServerAdmin`. Ihr Wert ist die E-Mail-Adresse des für den Server verantwortlichen Webmasters. Bei Fehlermeldungen wird diese Adresse mit ausgegeben.

Die Direktive `ServerName` erwartet die Angabe eines Hosts und, getrennt durch Doppelpunkt, den Default-Port, also `host:Portnummer`. Bei registrierten Domains ist dies der Haupt-Domain-Name, z.B. `ichundmeinserver.de`, oder, wenn kein solcher Domain-Name

existiert, die IP-Adresse des Rechners oder ein Hostname. Bei lokalen Installationen kann hier z.B. die Loopback-Adresse `127.0.0.1` oder der Default-Hostname `localhost` angegeben werden. Default-Portnummer ist `80`.

Die Direktive `UseCanonicalName` wird von Apache verwendet, wenn es nötig wird, aus einer Ressource des vom Server selbst verwalteten Domain-Bereichs einen vollständigen URI zu erzeugen. Mögliche Werte sind `On` und `Off`.

Ganz wichtig und fast in jeder Apache-Konfiguration zu ändern ist die Direktive `DocumentRoot`. Hier wird der absolute Pfad des Verzeichnisses angegeben, unterhalb dessen die Startdatei des Webangebots liegen soll. Sie können also Ihre Webdaten ablegen, wo Sie möchten. Wichtig ist nur, dass Sie an dieser Stelle angeben, wo die Daten liegen. Der Verzeichnispfad wird in Unix-Schreibweise notiert, also durch einfache Schrägstriche, nicht durch die in der Windows-Welt üblichen Backslashes. Wenn Sie einen Apache unter MS Windows installiert haben und einen Laufwerksbuchstaben mit angeben wollen, können Sie das in der Form wie im Beispiel `f:/daten/web` tun. Wichtig ist außerdem, dass das Verzeichnis am Ende keinen abschließenden Schrägstrich erhält.

Ebenfalls Pflicht ist es für jeden Webmaster, sich über grundsätzliche Verzeichnisoptionen Gedanken zu machen. Die Verzeichnisoptionen gehören zu einer Gruppe von Verzeichnisdirektiven. Diese werden in `<Directory> ... </Directory>` eingeschlossen. Die Direktive für Verzeichnisoptionen lautet innerhalb dieser Tags `Options`. Der Direktive können mehrere Werte zugewiesen werden. Beispiel:

```
<Directory />
   Options Indexes FollowSymLinks Includes
</Directory>
```

Das Beispiel bedeutet: Setze für das bei `DocumentRoot` angegebene Wurzelverzeichnis für Webseiten (bezeichnet durch `/` im einleitenden <Directory>-Tag) sowie für alle Verzeichnisse unterhalb davon die Optionen `Indexes`, `FolllowSymLinks` und `Includes`. Alle anderen verfügbaren Optionen werden nicht gesetzt. Nachfolgende Optionen sind möglich:

Directory-Option	Bedeutung
`All`	Es werden alle Optionen außer `MultiViews` gesetzt. Eine weitere Angabe von Einzeloptionen ist dann nicht erforderlich.
`ExecCGI`	Wenn gesetzt, können in diesem Verzeichnis und seinen Unterverzeichnissen CGI-Scripts abgelegt und ausgeführt werden.
`FollowSymLinks`	Wenn gesetzt, folgt der Server in diesem Verzeichnis und seinen Unterverzeichnissen symbolischen Links. Symbolische Links sind auf Unix-/Linux-Systemen ein weiteres Mittel, um ehemalige Dateinamen auf neuere automatisch weiterzuleiten.
`Includes`	Wenn gesetzt, können in diesem Verzeichnis und seinen Unterverzeichnissen Server-Side-Include-Dateien (also SHTML-Dateien) platziert werden. Der Server beachtet darin enthaltende SSI-Direktiven.

Tabelle 9.5: Zentrale Verzeichnisoptionen in der Apache-Konfiguration

Directory-Option	Bedeutung
`IncludesNOEXEC`	Wie Includes, jedoch ohne die SSI-Direktiven #exec und #exec cgi. Das bedeutet, es können in SHTML-Dateien zwar beispielsweise Inhalte aus Dateien dynamisch eingebunden werden, nicht aber dynamisch Scripts ausgeführt werden.
`Indexes`	Wenn eine von einem Client angeforderte URI ein Verzeichnis ist und in dem Verzeichnis keine Datei mit einem der Namen liegt, die bei der Direktive `DirectoryIndex` genannt sind, dann wird ein Listing des Verzeichnisinhalts gesendet.
`MultiViews`	Wenn gesetzt, ist in diesem Verzeichnis und seinen Unterverzeichnissen die so genannte Content Negotiation möglich. Darunter versteht man das flexible Ausliefern von Inhalten, abhängig von clientspezifischen Bevorzugungen, z. B. für deutschsprachige Versionen einer Webseite an Browser mit deutschsprachiger Benutzeroberfläche.
`SymLinksIfOwnerMatch`	Wie `SymLinks`, jedoch mit der Einschränkung, dass nur symbolischen Links gefolgt wird, wenn Zieldatei und Zielverzeichnis der gleichen Benutzerkennung angehören wie Verzeichnis und Datei mit dem symbolischen Link.

Tabelle 9.5: Zentrale Verzeichnisoptionen in der Apache-Konfiguration (Fortsetzung)

Weitere mögliche Direktiven innerhalb von `<Directory> ... </Directory>` sind `AllowOverride` (*.htaccess*-Dateien erlauben oder nicht) oder `Deny from` (IP-Bereiche oder Domains ausschließen). Auf diese Möglichkeiten gehen wir jedoch erst im nächsten Abschnitt in Zusammenhang mit *.htaccess*-Dateien näher ein. Es ist durchaus erlaubt, in der *httpd.conf* für jeden gewünschten Verzeichnisunterbaum separate Angaben mit `<Directory> ... </Directory>` zu notieren, z. B.:

```
<Directory /fotos>
   Options Indexes
</Directory>
```

Da es sich jedoch um verzeichnistypische Einstellungen handelt, können diese alternativ auch in *.htaccess*-Dateien im Verzeichnis selbst notiert werden.

Eine wichtige zentrale Direktive ist *DirectoryIndex*. Geben Sie hier durch Leerzeichen getrennt alle Dateinamen an, die bei Anforderung eines Verzeichnis-URI wie `/` oder `/fotos/` automatisch ausgeliefert werden, wenn eine oder mehrere davon sich in einem Verzeichnis befinden. Beispiel:

```
DirectoryIndex index.html index.htm index.php
```

Die Reihenfolge ist ebenfalls von Bedeutung. Sollten sich im Beispiel im Verzeichnis eine *index.htm* und eine *index.php* befinden, wird die *index.htm* ausgeliefert, weil sie zuerst notiert ist.

Die übrigen Direktiven der zentralen Server-Konfiguration sollten Sie nur ändern, wenn Sie besondere Gründe dafür haben.

9.4.5 Dezentrale Konfigurationsmöglichkeiten über .htaccess

In der Unix-/Linux-Welt haben Dateinamen, die mit einem Punkt beginnen, besondere Bedeutung. Es handelt sich in der Regel um verzeichnisspezifische Konfigurationsdateien. Es kann auch so eingestellt sein, dass Benutzer diese Dateien überhaupt nicht sehen können, wenn sie sich den Verzeichnisinhalt auflisten. Um solche Dateien bearbeiten zu können, müssen Sie sich auf der Shell-Ebene also entweder mit dem Administratorzugang (root-Kennung) oder mit einem Benutzernamen anmelden, für den solche Dateien sichtbar sind.

Auch der Apache Webserver verwaltet solche mit Punkt beginnende Dateien. Per Voreinstellung (in der *httpd.conf* bei der Direktive `AccessFileName` festlegbar) heißen diese Dateien *.htaccess*.

.htaccess-Dateien sind reine Textdateien. Am besten bearbeiten Sie sie vor Ort auf dem Server. Melden Sie sich dazu über SSH am Server-Rechner an. Wechseln Sie auf Shell-Ebene mit dem Kommando `cd` (change directory) in das gewünschte Verzeichnis, z. B. durch Eingabe von `cd /usr/home/web/fotos`. Erzeugen Sie dann beispielsweise mit dem vi-Editor eine .htaccess-Datei durch Eingabe von `vi .htaccess`.

In .htaccess-Dateien lassen sich z. B. folgende Optionen definieren:

- Verzeichnisschutz, d.h., nur Benutzer mit definiertem Benutzernamen und Passwort können Inhalte in diesem Verzeichnis und seinen Unterverzeichnissen aufrufen.
- Directory-Browsing: Für die Optik lassen sich Optionen festlegen, wie der Verzeichnisinhalt aufgelistet wird, falls eine typische Indexdatei fehlt, und ob Verzeichnis-Browsing erlaubt ist.
- Verzeichnisoptionen: Es sind die gleichen Angaben möglich wie bei der Direktive `Options` in der zentralen Konfiguration von Apache. Ebenso sind Angaben zu Direktiven wie `DirectoryIndex` und `ErrorDocument` erlaubt.
- Bei Content Negotiation können Optionen eingestellt werden, unter welchen Bedingungen welche Inhalte ausgeliefert werden.

Ein ausführliches Beispiel mit Anleitung zum Einrichten eines Verzeichnisschutzes werden wir in ***Abschnitt 11.4*** behandeln.

9.4.6 Virtuelle Hosts mit Apache

Die Webstartseite eines lokal eingerichteten Apache Webservers ist über eine der beiden URIs *http://127.0.0.1/* oder *http://localhost/* erreichbar, bei LANs auch unter der LAN-IP-Adresse des Rechners wie z. B. *http://192.168.101.4/*. Auf einem Host-Rechner, der mit eigener IP-Adresse im Internet ansprechbar ist, kann die Webstartseite mit dessen IP-Adresse erreicht werden, z. B. mit *http:// 192.0.34.166/*.

Damit eine Namensadresse wie beispielsweise *http://www.example.org/* weltweit aufrufbar ist, muss ein richtiger DNS-Server vorgeschaltet sein, der über den weltweiten Ver-

bund der DNS-Gateways in der Lage ist, die zu einem Domain-Namen gehörende IP-Adresse ausfindig zu machen.

Wenn Sie bei einem Provider ein Web-Hosting-Angebot des Typs Virtual Hosting (also keinen eigenen Root-Server) wahrnehmen, dann ist es jedoch häufig so, dass die Webseiten verschiedener Domain-Namen (wie etwa *duseldamp.de*, *fondelkauz.de* und *kulabersitz.de*) auf dem gleichen Server-Rechner untergebracht, also im DNS der gleichen IP-Adresse zugeordnet sind. Treffen entsprechende Anfragen beim Webserver eines solchen Rechners ein, muss dieser selbst wissen, in welchem Startverzeichnis beispielsweise die Webseiten jeder einzelnen Domain liegen. Auch andere Konfigurationen können ja kundenspezifisch sein, wie etwa die Namen von Verzeichnis-Default-Dateien. Um solche »dezentralen« Konfigurationen für unterschiedliche Domains zu ermöglichen, verfügen Webserver wie Apache über das Feature, virtuelle Hosts einzurichten.

Was in der *httpd.conf* in ***Abschnitt 2***, also der Main Server Configuration, eingestellt wird, gilt global für den Default-Host, den der Apache verwaltet. In ***Abschnitt 3*** der Konfigurationsdatei können Sie jedoch innerhalb der Direktivengruppe `<VirtualHost> ... </VirtualHost>` beliebig viele weitere Hosts einrichten. Innerhalb der umschließenden Gruppen-Tags können fast alle Direktiven, die in der *httpd.conf* vorkommen, nochmals notiert werden, wobei die dort angegebenen Werte dann aber nur für diesen speziellen virtuellen Host gelten.

Für Hosting-Provider ist dieses Feature also der Schlüssel, um mehrere Kunden mit jeweils eigenen Domains auf dem gleichen Rechner zu hosten. Diese Einsatzmöglichkeit soll uns hier jedoch weniger interessieren. Hier soll es darum gehen, zu zeigen, wie Sie mithilfe von Virtual Hosting bequem verschiedene Webprojekte lokal pflegen können. Wenn Sie nämlich mehrere verschiedene Webprojekte pflegen, die im Web auf verschiedenen Servern liegen und/oder über verschiedene Domain-Namen zugänglich sind, dann ist es sinnvoll, sich in der lokalen Entwicklungsumgebung am eigenen PC für jedes Webprojekt eine eigene Namensadresse einzurichten. So können Sie Ihre unterschiedlichen Projekte anschließend lokal über Adressen wie *http://moebelhaus.localhost/* oder *http://verein.test/* aufrufen. In den Apache-Konfigurationsdirektiven können Sie dann für jeden virtuellen Host die Konfigurationseinstellungen mit denen am jeweiligen »echten« Server des Projekts abstimmen.

Um in der lokalen Entwicklungsumgebung einen virtuellen Host einzurichten, können Sie wie folgt vorgehen:

1. Geben Sie in der *httpd.conf* des Apache am Ende der Datei folgendes Minimalgerüst ein:

```
<VirtualHost *:80>
  ServerName kunde.meier.localhost
  DocumentRoot /daten/web/meier/htdocs
  ErrorLog /daten/web/meier/logs/error.log
  TransferLog /daten/web/meier/logs/access_log
</VirtualHost>
```

Damit richten Sie einen Beispiel-Host ein, der wie üblich über den HTTP-Port 80 erreichbar ist und über *http://kunde.meier.localhost/* im Browser aufgerufen werden kann. Die Webinhalte des Projekts liegen unterhalb des Verzeichnisses */daten/web/meier/htdocs*. Ebenfalls angegeben werden die Dateipfade für die beiden wichtigen Server-Log-Dateien für Webzugriffe und Fehler.

Wählen Sie als Servernamen für den virtuellen Host am besten einen gewünschten Namen und hängen Sie *.localhost* oder *.test* an. Dies sind offiziell für solche Zwecke vorgesehene Top-Level-Domain-Bezeichnungen.

2. Speichern Sie die Änderungen in *httpd.conf*.
3. Starten Sie den Apache Webserver neu wie in ***Abschnitt 9.4.2*** beschrieben.
4. Legen Sie die erforderlichen Verzeichnisse im Dateisystem an, also z.B. die Pfade */daten/web/meier/htdocs* (Startverzeichnis für Webinhalte in unserem Beispiel) und */daten/web/meier/logs* (Verzeichnis für Log-Dateien des Webservers für diesen virtuellen Host). Falls bereits Dateien für das Webprojekt existieren, legen Sie diese in das angegebene Startverzeichnis für Webinhalte. Wenn noch keine Dateien existieren, erzeugen Sie zum Testen eine kleine HTML-Datei, z.B. *test.htm*, und legen diese im Verzeichnis ab.
5. Suchen Sie auf dem lokalen Rechner die hosts-Datei, in der lokale Hostnamen vergeben werden können. Öffnen Sie die Datei mit einem Texteditor zum Bearbeiten.

   Bei neueren Versionen von MS Windows (2000/NT/XP) liegt die hosts-Datei unterhalb des Windows-Programmverzeichnisses unter *system32\drivers\etc*. Bei Linux-Rechnern liegt sie unter */etc*.

   Innerhalb der Datei können Sie IP-Adressen Hostnamen zuordnen. Das Schema lautet:

   ```
   IP-Adresse   Hostname [Hostname2 ...]
   ```

   IP-Adresse und Hostname(n) werden durch ein oder mehrere Leerzeichen getrennt.
6. Ordnen Sie in der hosts-Datei der lokalen IP-Adresse, also der Loopback-Adresse 127.0.0.1, den Hostnamen zu, den Sie auch in der *httpd.conf* bei `<VirtualHost> ... </VirtualHost>` hinter der Direktive *ServerName* vergeben haben. Im Beispiel könnte der Eintrag also lauten:

   ```
   127.0.0.1  localhost  kunde.meier.localhost
   ```

   Sie können die Datei *hosts* übrigens auch nutzen, um Ihren privaten kleinen DNS-Server zu fahren und damit Webzugriffe auf häufig aufgerufene Seiten zu beschleunigen. Der Webbrowser sucht nämlich bei jeder Namensadresse erst einmal in dieser Datei, bevor er beim DNS-Server der Internetverbindung anfragt. Ein Beispiel:

   ```
   213.198.84.179  de.selfhtml.org
   ```

Damit kann der Browser die Dokumentation SELFHTML viel schneller auffinden als wenn er erst den DNS-Server des Providers befragen muss. Interessant sind solche Einträge beispielsweise für die Produktivseiten selbst betreuter Webprojekte, da man diese häufig aufrufen muss. Die IP-Adressen von Domain-Namen bekommen Sie heraus, indem Sie auf Shell-Ebene oder unter Windows in einem DOS-Fenster ein ping-Kommando auf den Domain-Namen absetzen, wie z. B. `ping de.selfhtml.org`.

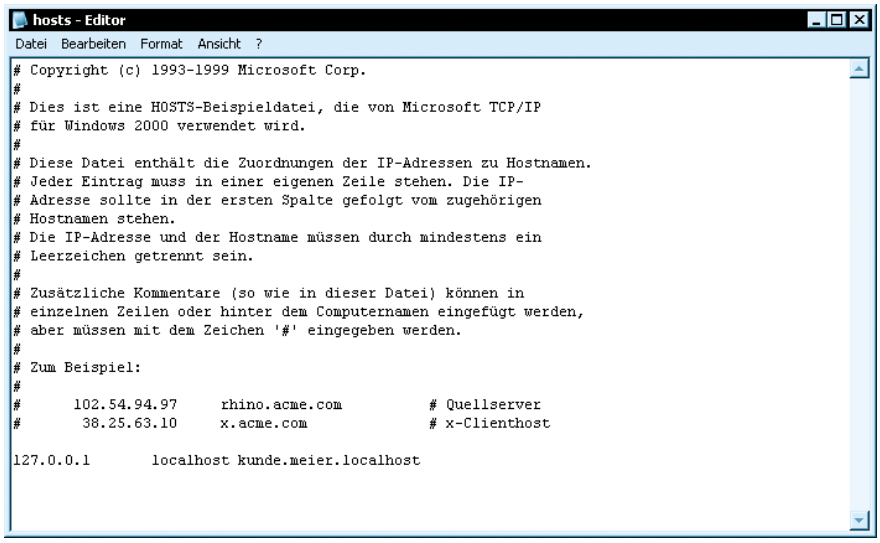


Abbildung 9.12: hosts-Datei unter MS Windows im Notepad-Editor

7. Speichern Sie die Datei *hosts*.
8. Testen Sie den angelegten virtuellen Host nun im Browser.

   In unserem Beispiel können Sie eingeben: `http://kunde.meier.localhost/`.

Neben dem Anlegen eines virtuellen Hosts in der Apache-Konfiguration ist also eine Anpassung auf Systemebene, nämlich in der hosts-Datei, erforderlich.

9.4.7 Log-Dateien des Apache Webservers

Per Voreinstellung protokolliert der Webserver alle Zugriffe und alle dabei aufgetretenen Fehler in so genannten Log-Dateien. Die Log-Dateien bilden die Basis für spätere Statistiken, wie sie etwa von Tools wie dem Webalizer erstellt werden.

Der typische Dateiname für die Log-Datei mit Fehlern heißt *error_log*, und der für die Log-Datei mit Zugriffen heißt *access_log* (unter Windows *error.log* und *access.log*). Beide Dateinamen und ihr Ablageort sind in der *httpd.conf* konfigurierbar und zwar in den Direktiven *ErrorLog* (für die Fehler-Log-Datei) und *CustomLog* (für die Zugriffs-Log-Datei). Auf Windows-Systemen wird dazu per Default ein Unterverzeichnis *logs* im Pro-

grammverzeichnis des Apache Webservers verwendet. Auf Unix/Linux-Systemen werden die Dateien in der Regel in einem Verzeichnispfad namens */var/log/httpd* abgelegt.

Für die Fehler-Log-Datei wird ferner eine Direktive namens `LogLevel` angeboten. Darin kann festgelegt werden, welche Arten von Meldungen in der Fehler-Log-Datei protokolliert werden sollen. Folgende Zuweisungen sind möglich:

LogLevel-Wert	Bedeutung
`debug`	Alle nur erdenklichen Ereignisse werden protokolliert. Nur für Testzwecke geeignet, da die Log-Datei ansonsten sehr schnell sehr groß werden kann.
`info`	Alle Fehler, Warnungen und Hinweise werden protokolliert und daneben auch reine Informationen.
`notice`	Alle Fehler und Warnungen werden protokolliert und daneben auch Hinweise auf bemerkenswerte Systemzustände, die jedoch keinen Fehler darstellen.
`warn`	Default-Einstellung: Alle Fehler werden protokolliert und daneben auch Warnungen bei Ereignissen, die zwar aus Sicht des Webservers kein unmittelbares Problem darstellen, aber aus Systemsicht durchaus problematisch sein können, z. B. wenn Zombie-Prozesse entstehen.
`error`	Nur die klassischen Fehlermeldungen des Webservers werden protokolliert. Dazu gehören diejenigen der Fehlertypen crit, alert und emerg, aber auch Fehler wie »File not found« (404).
`crit`	Nur Fehler, die für die Stabilität des Webservers selbst kritisch sind, werden protokolliert, z. B. wenn eine Socket-Verbindung nicht geöffnet werden konnte.
`alert`	Nur Fehler, aufgrund deren der Webserver selbst nicht mehr ordnungsgemäß agieren kann, werden protokolliert, z. B. wenn es Probleme auf Benutzerebene gibt.
`emerg`	Nur Systemfehler, die den Webserver zum Beenden bringen, werden protokolliert, z. B. wenn der Server wichtige Dateien nicht mehr schreiben kann, weil z. B. die Festplatte voll ist.

Tabelle 9.6: Loglevel für Fehler-Log-Datei

Jede Zeile in einer der Log-Dateien steht für einen protokollierten Eintrag. Das Format der Einträge in der Fehler-Log-Datei ist vorgegeben. Das der Zugriffs-Log-Datei können Sie dagegen aus Einzelbausteinen selbst zusammensetzen, je nachdem, welche Informationen Sie darin wünschen.

Ein Beispieleintrag aus der *httd.conf*:

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\""
combined
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
```

Jede Definition zur Direktive `LogFormat` legt ein mögliches Eintragsformat fest. Am Ende wird für das Eintragsformat ein Name vergeben, wie im Beispiel *combind*, *common*, *referer* und *agent*. Diese Namen können dann bei der Direktive *CustomLog* angegeben werden. Beispieleintrag:

```
CustomLog /var/log/httpd/access_log common
```

Im Beispiel wird bestimmt, dass die Zugriffe in der Datei *access_log* im Verzeichnis */var/log/httpd* gespeichert werden sollen und dass dabei das unter dem Namen *common* definierte Log-Format verwendet werden soll. Falls Sie übrigens das Default-Log-Format des Apache verwenden möchten, können Sie die Direktive `TransferLog` anstelle von `CustomLog` verwenden. Beispiel:

```
TransferLog /var/log/httpd/access_log
```

Der Unterschied zwischen `TransferLog` und `CustomLog` besteht also nur darin, dass bei *CustomLog* zusätzlich der Name des gewünschten Log-Formats angegeben wird.

Das Log-Format wird aus Bausteinen zusammengesetzt, deren Bezeichner durch ein Prozentzeichen eingeleitet wird. Der gesamte »Formatstring« wird in Anführungszeichen gesetzt. Nachfolgende Tabelle listet die Bedeutung von Prozentbezeichnern auf:

Bezeichner	Bedeutung
`%a`	IP-Adresse des anfragenden Hostrechners.
`%A`	Lokale IP-Adresse.
`%B`	Anzahl Bytes der Server-Antwort abzüglich HTTP-Header.
`%b`	Wie %B, nur dass bei 0 Byte nicht »0«, sondern »-« geschrieben wird.
`%C`	Inhalt eines Cookies in der Anfrage. Der Name des Cookies muss in geschweiften Klammern stehen. Beispiel: `%{Expire}C`.
`%D`	Benötigte Antwortzeit des Servers in Mikrosekunden.
`%e`	Inhalt einer Umgebungsvariable bei der Anfrage. Der Variablenname muss in geschweiften Klammern stehen. Beispiel: `%{HTTP_ACCEPT}e`.
`%f`	Angeforderter Dateiname.
`%h`	Hostname des anfragenden Hostrechners.
`%H`	Verwendetes Protokoll (z. B. HTTP oder HTTPS).
`%i`	Inhalt eines HTTP-Headers bei der Anfrage. Der Feldname muss in geschweiften Klammern stehen. Beispiel: `%{EXPIRES}i`.
`%m`	Anfragemethode (z. B. GET oder POST).
`%o`	Inhalt eines HTTP-Headers bei der Antwort. Der Feldname muss in geschweiften Klammern stehen. Beispiel: `%{CONTENT-TYPE}o`.
`%p`	Kanonischer Port des Webservers.
`%P`	Prozess-ID (PID) des Kindprozesses, der die Anfrage behandelt.
`%q`	GET-String im URL-Request.

Tabelle 9.7: Prozentbezeichner für Apache-Direktive »LogLevel«

Bezeichner	Bedeutung
%r	Erste Zeile des HTTP-Headers.
%s	Status der Anfrage.
%t	Zeitpunkt der Anfrage.
%T	Vom Server benötigte Zeit zum Ausliefern der Anfrage.
%u	Benutzername des Anfragenden, sofern dieser authentifiziert ist.
%U	Angeforderter Pfadname.
%v	Kanonischer Name des Servers.
%X	Verbindungsstatus (in der Log-Datei steht dann x für »abgebrochen«, + für »wird gehalten« oder – für »beendet«).
%I	Anzahl der bei der Anfrage empfangenen Bytes inklusive HTTP-Header.
%O	Anzahl der bei der Antwort gesendeten Bytes inklusive HTTP-Header.

Tabelle 9.7: Prozentbezeichner für Apache-Direktive »LogLevel« (Fortsetzung)

Die Log-Dateien liegen im Klartext vor und können mit einem Texteditor oder einem Textanzeigeprogramm betrachtet werden. Allerdings können die Log-Dateien sehr groß werden und von manchen Programmen wegen ihrer Größe möglicherweise nicht verarbeitet werden.

Provider in Deutschland sind verpflichtet, Log-Dateien über einen gewissen Zeitraum hinweg aufzubewahren. Das sollte das Backup-System regeln. Außerdem müssen die Log-Dateien Rückschlüsse darüber erlauben, wer wann was aufgerufen hat. Achten Sie deshalb darauf, sofern Ihr Server in Deutschland steht, bei einem CustomLog-Format zumindest die Bezeichner %a, %h, %t und %U zu verwenden – auch im eigenen Interesse. Denn mithilfe dieser Daten können Sie z.B. schwere Hausfriedensbrecher in Foren oder Gästebüchern erfolgreich zur polizeilichen Anzeige bringen.

10 Basiswissen Linux für Webworker

Dieses Kapitel richtet sich an Anbieter von Websites, die sich beim Web-Hosting für ein Root-Server-Angebot entschieden haben, also vom Provider einen vorkonfigurierten Server-Rechner zur Verfügung gestellt bekommen, der im Server-Raum oder Rechenzentrum des Providers untergebracht ist. Für den Rechner erhalten Sie eine so genannte root-Kennung, d.h., Sie können sich mit maximalen Administratorrechten am Rechner anmelden.

Da die meisten der von Hosting-Providern angebotenen Root-Server mit einem Linux-Betriebssystem vorkonfiguriert sind, konzentrieren wir uns auf diese Systemfamilie. Viele der hier behandelten Konzepte und Kommandos usw. sind auch auf anderen Unix-Derivaten verfügbar. Jedoch kann es bei anderen Unix-Systemen durchaus gravierende Abweichungen und Unterschiede geben.

Immer mehr Anwender installieren auch am eigenen PC ein Linux-Betriebssystem. In der Regel ist es kein Problem, mehrere Betriebssysteme wie etwa MS Windows oder Linux parallel zu installieren und beim Starten des PCs auszuwählen, welches System geladen werden soll. Dabei kann auch auf die gleichen Verzeichnisse und Dateien zugegriffen werden.

Der Vorteil einer lokalen Linux-Installation ist nicht zu verachten: Nur dadurch können Sie die Entwicklungsumgebung wirklich optimal auf die spätere Produktivumgebung abstimmen. Einige Möglichkeiten, die auch bei Websites zum Einsatz kommen, wie z.B. cron-Jobs, sind nun einmal in dieser Form unter Windows nicht möglich bzw. nur durch entsprechende andere Systemprogramme nachahmbar. Auch manche Features, die bei serverseitigem Scripting von Bedeutung sein können, wie z.B. das Sperren von Dateien für den Zugriff durch andere Prozesse (File-Locking), sind unter Windows gar nicht oder nicht in der unter Unix/Linux typischen Form verfügbar.

Egal ob Sie sich also über einen SSH-Client wie PuTTy (siehe ***Abschnitt 9.2.4***) an ihrem frisch gemieteten Root-Server angemeldet haben und nun voller Fragezeichen vor dem Shell-Prompt sitzen oder ob Sie sich etwas Mut zum Installieren eines lokalen Linux-Systems anlesen möchten – dieses Kapitel führt in die Welt von Linux ein, ohne dabei in den arroganten Abgrenzungsjargon mancher Hardliner abzugleiten. Das vermittelte Wissen ist darauf ausgerichtet, was ein ambitionierter Webmaster kennen sollte, nicht aber auf die Anforderungen, die an einen Systemspezialisten gestellt werden.

10.1 Linux als Server-Betriebssystem

Linux basiert auf Unix. Unix ist mittlerweile mehr ein Oberbegriff für eine Großfamilie von Betriebssystemen (die so genannten *Unix-Derivate*) als selbst ein Betriebssystem. Unix ging aus einem Ende der 60er Jahre gescheiterten Projekt zur Entwicklung eines neuartigen Betriebssystems hervor. Zwei der Know-how-Träger des Projekts entwickelten die Ideen des gescheiterten Projekts weiter, woraus eine erste Fassung von Unix hervorging. Dabei wurde auch gleich eine neue Programmiersprache mit entwickelt, in der Unix geschrieben war: die Sprache C. Noch heute ist C die Standard-Programmiersprache für Softwareentwicklung unter Unix und Linux. Fast alle bekannten Softwareprodukte, angefangen vom Betriebssystemkern selbst, bis hin zu Produkten wie Apache, PHP, Perl oder MySQL, werden in C entwickelt. Wer C kennt, wird auch schnell feststellen, dass zahlreiche Standardfunktionen in Scriptsprachen wie PHP und Perl auf Standardfunktionen in C zurückgreifen.

Unix-Derivate, darunter auch alle Linux-Distributionen, haben folgende charakteristische Eigenschaften:

- *Multi-User-Fähigkeit:* Es ist möglich, Benutzer und Benutzergruppen anzulegen. Benutzer können bei Verzeichnissen und Dateien festlegen, welche anderen Benutzer und Benutzergruppen welche Rechte haben, z. B. ob sie einen Verzeichnisinhalt sehen können, ob sie darin arbeiten können, ob sie eine Datei schreibend oder nur zum Lesen öffnen können, ob sie ein ausführbares Programm starten können usw. Unter MS Windows gibt es eine ernstzunehmende Benutzerverwaltung erst seit NT/2000/XP.
- *Multitasking-Fähigkeit:* Das Betriebssystem ist in der Lage, die verfügbare CPU-Zeit an mehrere Prozesse zu verteilen, wobei Verteilungsprioritäten beeinflussbar sind. Mehrere Prozesse können dabei auch im Arbeitsspeicher so verwaltet werden, dass keiner den anderen stört und zum Absturz bringt. Auch das beherrscht MS Windows erst seit NT/2000/XP in akzeptabler Form.
- *Netzwerk- und Internetfähigkeit:* Unix-Derivate haben sich in den 70er, 80er und 90er Jahren weitgehend unbemerkt von der breiten Öffentlichkeit parallel und gemeinsam mit dem Internet entwickelt und weiterentwickelt. Das Ergebnis ist eine unerreicht exakte und konforme Unterstützung von Netzstandards und umgekehrt sind nicht wenige Netzstandards so entwickelt worden, wie sie sind, weil ihre Realisierbarkeit sich an Standards aus der Unix-Welt orientierte.
- *Prozessorientierte Arbeitsweise:* Während desktoporientierte Betriebssysteme wie MS Windows oder Apple Macintosh auf eine dialogorientierte Oberfläche hin optimiert sind, liegt die Stärke von Unix-Derivaten eher im Abarbeiten von Prozessen und deren Kombination oder Kommunikation. Zwar gewinnt Linux in den letzten Jahren auch zunehmend Anteile am Markt der grafischen PC-Oberflächen. Doch im Gegensatz zu Windows etwa sind Betriebssystem und grafische Oberfläche bei Unix-Derivaten vollkommen getrennt. Zum notwendigen Arbeitsumfeld eines Unix-Betriebssystems gehört zwar eine so genannte Shell, also eine textorientierte Kommandoschnittstelle. Doch selbst die Shell ist wie der Name schon sagt nur eine Schale, die letztlich aus-

wechselbar ist. Das gilt erst recht für grafische Oberflächen. Die Stärke der Unix-Betriebssysteme liegt daher auch in prozessverarbeitenden Features wie Input-Output, Piping, Forking, Signalisierung usw.

- *Programmierbarkeit:* Wer sich ernsthaft auf Unix-/Linux-Systeme einlässt, kommt früher oder später mit eigener Programmierung in Berührung. Im Gegensatz zu Systemen wie MS Windows, die versuchen, den Anwender vom Betriebssystem abzuschirmen, gehören Programmier- und Scripting-Schnittstellen bei Unix-/Linux-Systemen zur Grundausstattung – egal, ob das der fast überall schon mit distributierte C-Compiler ist oder zumindest eine Shell-Scripting-Schnittstelle oder optimierte Script-Sprachen wie Perl, die ebenfalls aus der Unix-Welt entstanden sind.

Anhand dieser Charakteristika wird schnell deutlich, warum Unix-/Linux-Systeme sich für den Einsatz als Server-Rechner im Internet optimal eignen:

- Für Client-Server-Kommunikation wie zwischen Browser und Webserver oder Mailprogramm und Mailserver ist die prozessorientierte Arbeitsweise optimal geeignet, ebenso für serverseitig dynamische Webseiten, etwa mit PHP.
- Multitasking ist im Server-Betrieb noch viel wichtiger als bei Desktop-PCs, da viele aufrufende Clients viele Prozesse erzeugen, die gleichzeitig abgearbeitet werden müssen.
- Multiuser-Fähigkeit ist ebenfalls sehr wichtig, da so beispielsweise skalierbar ist, welche Prozesse welche anderen Prozesse anstoßen dürfen und welche Projektmitarbeiter in welchen Bereichen arbeiten dürfen.
- Die Unterstützung von Internetstandards ist zuverlässig und in höchstem Maße konform integriert.
- Zahlreiche Basiskonzepte, wie z.B., dass alle Konfigurationsdaten in Klartextdateien stehen (statt in einer monströsen, binär gespeicherten Registry), kommen findigen Programmierern entgegen.

Auch wenn ein System sich wie etwa beim Fernzugang über SSH nur in Form einer spartanischen Shell präsentiert, ist es deswegen nicht »dümmer« als eine pfiffige grafische Oberfläche. Im Gegenteil – der wahre Snob trägt den Goldsaum immer nur im Innenfutter.

Weiterführende Links:

Dokumentation SELFLINUX:
http://www.selflinux.org/selflinux/
Linux-Handbuch:
http://www.linux-ag.de/linux/LHB/index.html
Wikipedia-Artikel (mit diversen ausführlichen Unterartikeln):
http://de.wikipedia.org/wiki/Linux

10.1.1 Geschichte und Bedeutung von Linux

In den 80er Jahren kam es in Folge von Lizenzproblemen zu einer Aufsplittung des »Ur-Unix« in zahlreiche Distributionen verschiedener kommerzieller Anbieter. Darunter das SunOS der Firma Sun, HP-UX von Hewlett-Packard, Sinix von Siemens oder AIX von IBM. Immerhin versuchte AT&T, der ursprüngliche Initiator für Unix, durch das so genannte »lila Buch« einen Standard zu definieren, dem alle Unix-Derivate genügen sollten. Der Standard wurde auch als »System V« (gesprochen »System five«) bekannt. Doch letztlich war dieser Standardisierungsversuch von AT&T auch nur eine Machtdemonstration des Lizenzinhabers der »Marke Unix«. So entstanden konkurrierende Standards wie POSIX oder X/Open.

Am meisten Erfolg hatte indes eine andere Entwicklungslinie, die auf den legendären Ideologen des Open-Source-Gedankens zurückgeht, auf Richard Matthew Stallman. Mit seinem *GNU-Projekt* (GNU steht für »GNU's not Unix«) brachte er ein Betriebssystem auf den Weg, das zahlreiche Basiskonzepte von Unix übernahm, jedoch offen dokumentiert und frei von Lizenzgebühren benutzbar war.

Während die Peripherie von GNU unter dem Namen *Minix* alsbald existierte, fehlte dem GNU-Projekt über lange Zeit noch der eigentliche Betriebssystemkern. Ein programmierbegabter junger Finne namens Linus Torvalds begann 1991 auf eigene Faust, einen solchen Kernel zu programmieren, und nannte ihn seinem Vornamen entsprechend *Linux*. Der Linux-Kernel wurde gemeinsam mit der GNU-Peripherie von der Benutzergemeinde adoptiert und verbreitete sich unter dem Namen GNU/Linux rasch.

Obwohl die GNU-Werkzeuge auch heute noch zu Linux gehören, ist meistens einfach nur noch von Linux die Rede. Außerdem ist auch Linux längst nicht mehr gleich Linux. Im Laufe der Jahre etablierten sich einige konkurrierende Distributionen von Linux. Die den meisten Neuanwendern im deutschsprachigen Raum bekannteste Distribution ist wohl die der Nürnberger Firma *SUSE*. Weil hinter dem Vertrieb eine Firma steckt, die ihre Linux-Distribution auf schicken CD-ROMs verkauft und mit einer narrensicheren Installationsroutine und einer mitgelieferten grafischen Oberfläche vor allem die Zielgruppe der Desktop-Endanwender im Auge hat, gibt es einige Hardliner, welche die SUSE-Distribution schon fast genau so hassen wie ihren Lieblingsfeind Microsoft. Andere, etwas »rohere« Distributionen sind etwa *Debian* Linux, *Redhat* Linux, *Gentoo* Linux, *Mandrake* Linux oder *Knoppix*.

Gemeinsam ist den Distributionen, dass sie den Linux-Kern sowie die GNU-Werkzeuge enthalten. Unterschiede gibt es vor allem bei den Shells, den grafischen Oberflächen, bei zusätzlichen Tools und bei der Art des Vertriebs.

Der einzige andere Familienzweig von Unix, der heute noch annähernd eine Bedeutung wie Linux hat, ist einer, der auf dem ursprünglichen AT&T-Unix basiert: die BSD-Familie, von denen heute die bekannteste Variante *FreeBSD* ist.

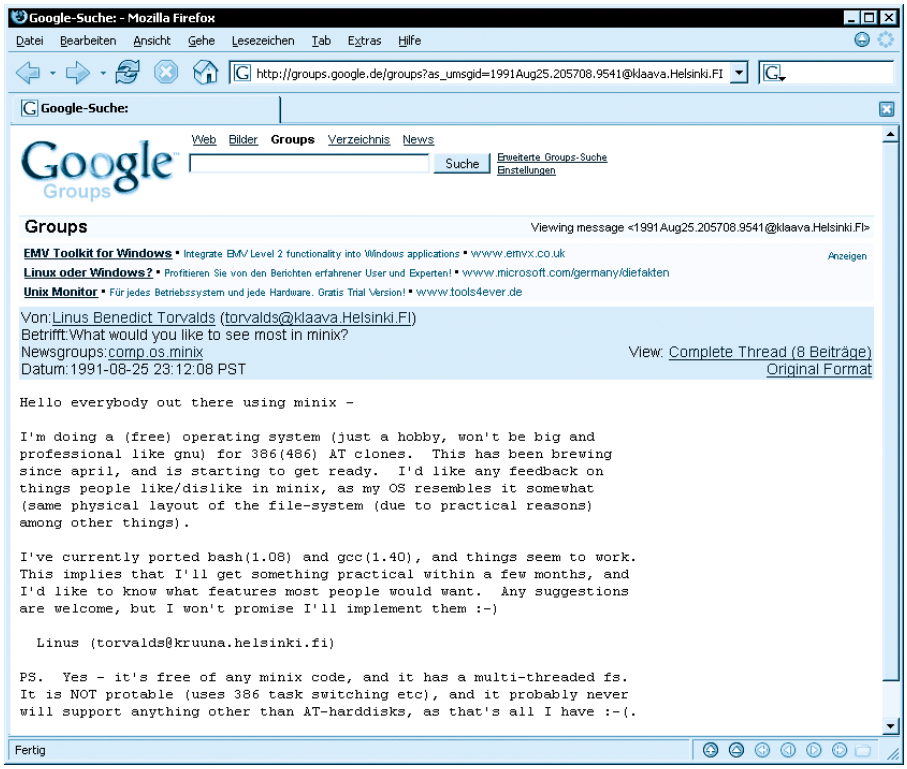


Abbildung 10.1: Legendäres Newsgroup-Posting, mit dem Linus Torvalds Linux ankündigte

Kommerzielle Unix-Distributionen führen heute dagegen ein Nischendasein. »Open Source« hat sich durchgesetzt. Letzteres bedeutet freilich nicht, dass damit nicht auch Geld verdient werden kann: Zahlreiche Spezialisten verdienen Geld durch Linux-Support, Programmierung für Linux-Systeme usw. Doch anders als bei kommerziell vertriebenen Systemen gibt es keine geschützten Quellcodes, keine einmaligen oder regelmäßigen Lizenzgebühren, keine Einschränkungen bei Vervielfältigung oder Weitergabe usw.

10.1.2 Aufbau und Komponenten von Linux

Ein installiertes, benutzbares Linux-System besteht grob gesagt aus:

- einem Betriebssystemkern, dem so genannten Kernel,
- aus einer oder mehreren Shells,
- gegebenenfalls aus einer oder mehreren grafischen Oberflächen.

Der Kernel

Der Kernel hat unter anderem folgende Aufgaben:

- Er stellt Gerätetreiber bereit,
- er verwaltet Prozesse im Arbeitsspeicher,

- er teilt Prozessen CPU-Zeit zu,
- er verwaltet das Dateisystem,
- er ist die Schnittstelle für Shells und grafische Oberflächen.

Der Original-Linux-Kernel ist von *http://www.kernel.org/* beziehbar. Je nach Distribution ist es jedoch oft einfacher, eine an die Distribution angepasste Kernel-Version zu verwenden.

Der Linux-Kernel wird ständig weiterentwickelt. Dabei werden in scheinbar unbedeutend klingenden Zwischenversionen häufig auch wichtige Sicherheitslücken geschlossen oder Schwächen beseitigt. Aus diesem Grund ist es natürlich auch für Mieter eines Root-Servers von Bedeutung, wenn verbesserte Versionen des Linux-Kernels verfügbar sind.

Da das Nachrüsten eines Kernels jedoch eine systemnahe Angelegenheit ist, die meistens auch einen Neustart des Rechners erfordert, ist es nur selten möglich, ein Kernel-Upgrage aus der Ferne, also etwa über einen SSH-Zugang, vorzunehmen. Fragen Sie, wenn Sie einen Root-Server mieten, am besten Ihren Provider, wie er die Möglichkeit von Kernel-Upgrades handhabt. Im Idealfall übernimmt der Provider solche Upgrades ohne Preisaufschlag, jedoch nur auf Anforderung des Kunden hin. Automatische Upgrades ohne Ankündigung können unangenehme Folgen wie Datenverlust haben und im Einzelfall kann es auch vorkommen, dass Programme oder Scripts nach einem Upgrade nicht mehr so funktionieren wie vorher. Als Kunde sollten Sie sich also über die Weiterentwicklung des Linux-Kernels auf dem Laufenden halten, doch in der Regel selbst keine Hand anlegen.

Wenn Sie Linux lokal installieren, installieren Sie den jeweils aktuellen Kernel der gewählten Linux-Distribution mit. Um eine neuere Kernel-Version nachzuinstallieren, laden Sie zunächst die gewünschte Kernel-Version von der Original-Kernel-Site oder von den Seiten der eigenen Linux-Distribution herunter. Die Distributoren bieten auch Handbücher an (als Print-Ausgaben oder online im Web), in denen das genaue Procedere beim Kernel-Upgrade beschrieben wird.

Shells

Wenn Sie sich beispielsweise an einem bei einem Provider gemieteten Server-Rechner mit einem SSH-Client wie PuTTy erfolgreich angemeldet haben, dann befinden Sie sich in einer Shell. Eine Shell ist ein Kommando-Interpreter, also eine Schnittstelle zwischen Benutzer und verfügbaren Kommandos und Programmen.

Es gibt zahlreiche Shells in der Linux-Welt. Die wohl bekannteste ist die so genannte ***bash*** (***Bourne Again Shell***). Diese ist eine namentliche Wiedergeburt der Original ***Bourne-Shell*** (***sh***). Andere für Linux verfügbare Shells sind die ***KornShell*** (***ksh***), die ***Z Shell*** (***zsh***) oder die ***C Shell*** (***csh***).

Manchmal verrät schon der voreingestellte Prompt (der Text vor der Eingabeaufforderung), um welche Shell es sich handelt. Falls Sie z. B. mit SSH an einem Root-Server angemeldet sind und überprüfen wollen, in welcher Shell Sie sich am Server-Rechner befinden, können Sie dies durch Eingabe von

```
echo $SHELL
```

ermitteln. Eine Ausgabe wie `/bin/bash` bedeutet beispielsweise, dass die bash-Shell benutzt wird und dass `bash` im Verzeichnis `/bin` die ausführbare Datei der Shell ist.

Eine Shell hat folgende Aufgaben:

- Sie interpretiert eingegebene Kommandos,
- sie ermöglicht Benutzern eine persönliche, frei konfigurierbare Arbeitsumgebung,
- sie stellt Komforthilfen beim Arbeiten auf der Kommandozeile bereit.

Auf einem typischen Linux-System können Sie am Shell-Prompt mehrere hundert Kommandos eingeben. Die meisten Kommandos erwarten Parameter oder Optionen zur Verarbeitung. Als Ergebnis wird irgendeine Art von Output geliefert. Es gibt Shell-Kommandos für alle erdenklichen Bereiche, egal ob Benutzerverwaltung, Datei- und Verzeichnisverwaltung, Prozessverwaltung, Systemkontrolle, Benutzerkommunikation, dateiübergreifendes Suchen und Ersetzen usw.

Anwender, die aus der Windows-Welt kommen, empfinden einen Kommando-Interpreter meist als Zumutung. Aber fast alle Anwender, die sich die Mühe gemacht haben, sich näher in die Konzepte einer Shell einzuarbeiten, berichten am Ende, dass sie auf Shell-Ebene gewisse Dinge einfach viel effizienter erledigen können als unter einer grafischen Oberfläche.

Grafische Oberflächen

Grafische Oberflächen sind nur dann ein Thema, wenn Sie Linux lokal installieren. Beim Fernzugriff über SSH können Sie immer nur auf Shell-Ebene arbeiten und maximal Dialogprogramme starten, die im Textmodus laufen, da das Terminal-Protokoll nur den Textmodus unterstützt.

Unter Linux sind grafische Oberflächen stärker abgekoppelt vom Betriebssystem als etwa unter MS Windows. Verschiedene Oberflächen können separat hinzu installiert werden. Vorteil dieser Lösung ist, dass Sie als Anwender letztlich frei entscheiden können, welche Oberfläche Sie einsetzen möchten. Nachteil ist, dass mehrere, zueinander nicht kompatible Oberflächen konkurrieren und viele interessante Softwareprodukte nur unter einer oder manchen der möglichen Oberflächen lauffähig sind.

In der Praxis vertreiben allerdings einige Linux-Distributionen bestimmte grafische Oberflächen gleich mit. So wird beispielsweise die SUSE-Distribution mit der KDE-Oberfläche (K-Desktop) vertrieben. Andere bekannte Oberflächen sind GNOME oder X-Windows. Auf welche Oberfläche die Wahl fällt, ist zunächst eine Frage des persönlichen Geschmacks, vor allem aber auch abhängig davon, welche benötigte Software für welche Oberfläche verfügbar ist.

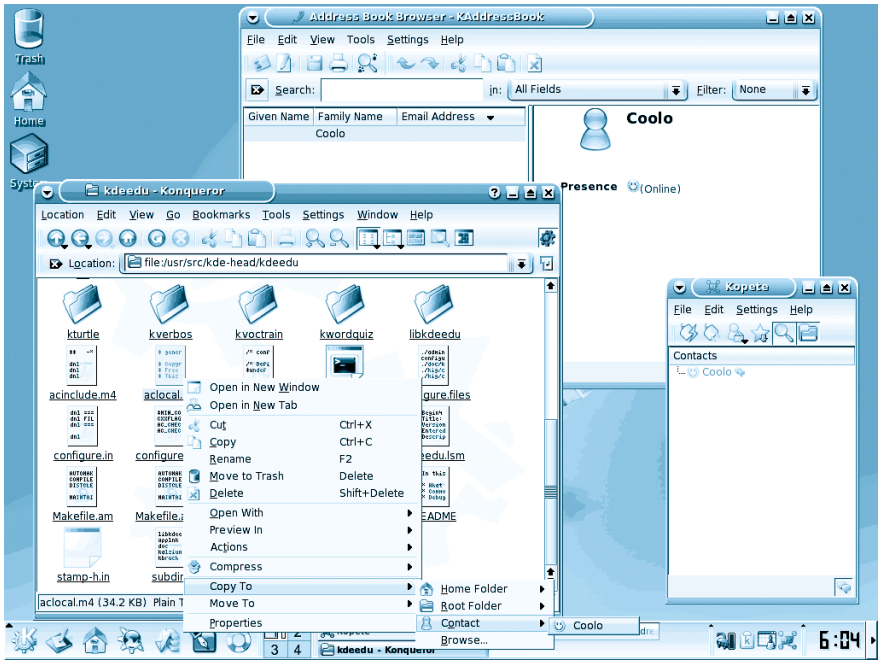


Abbildung 10.2: KDE-Oberfläche

10.1.3 Standard-Verzeichnisbaum von Linux

Ein wichtiger Unterschied zwischen dem Unix-Dateisystem und dem von MS Windows ist, dass es unter Unix/Linux keine Laufwerke und daher auch keine Laufwerksbuchstaben gibt. Geräte wie Festplatten, DVD, CD, Compact Flash Cards oder USB-Dongle erscheinen unter Linux als Verzeichnisse im zentralen Verzeichnisbaum. Man spricht auch von Gerätedateien. Auch direkt ansprechbare LAN-Laufwerke sind über den zentralen Verzeichnisbaum wie lokale Verzeichnisse erreichbar.

Bei Pfadangaben wird im Gegensatz zu Windows stets nur der einfache Schrägstrich / benutzt. Bereits bei diesem Detail ist die größere Nähe von Internetstandards wie dem URI-Schema zu Unix/Linux zu spüren.

Das oberste Verzeichnis des zentralen Verzeichnisbaums ist das *root directory* (Wurzelverzeichnis). Durch folgende Eingabe am Shellprompt können Sie sich den Inhalt des Wurzelverzeichnisses detailliert auflisten lassen:

```
ls -la /
```

Das ls-Kommando listet Verzeichnisinhalte auf. Die Option `l` erzwingt eine detaillierte Ausgabe und a sorgt dafür, das auch die so genannten Punktdateien angezeigt werden, also die »geheimen« Konfigurationsdateien, welche mit einem Punkt beginnen, wie z.B. *.htaccess*. Der zweite Parameter, den das ls-Kommando übergeben bekommt, ist das aufzulistende Verzeichnis. Da / das Wurzelverzeichnis adressiert, wird also dessen Inhalt aufgelistet.

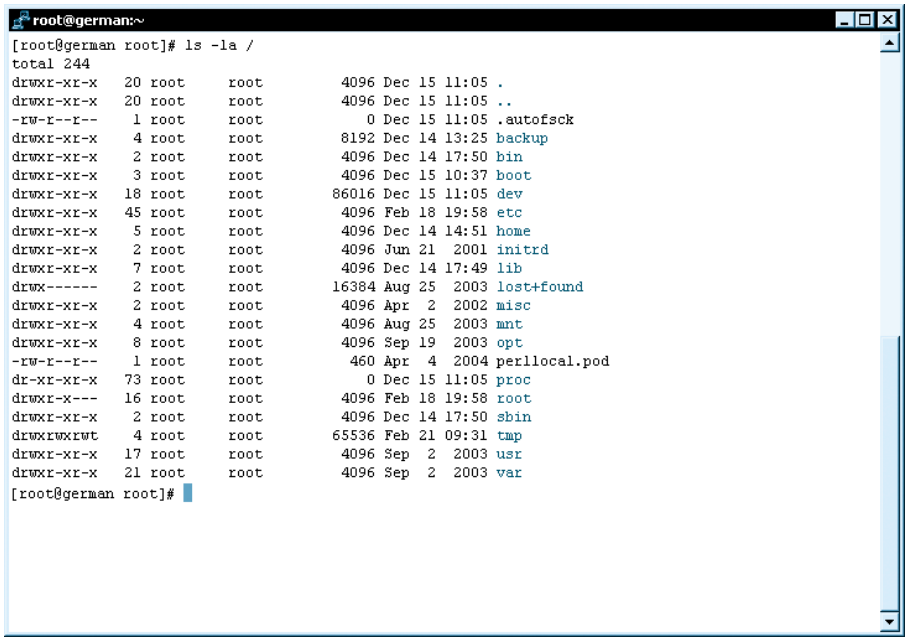


Abbildung 10.3: Kommando ls -la / über SSH-Client PuTTy auf Root-Server

Es gibt eine grobe Verzeichnisstruktur, die in dieser Form von jedem Unix-System angelegt wird, auch von den verschiedenen Linux-Distributionen. Um sich in dem für Windows-Anwender anfänglich etwas ungewohnten Verzeichnisbaum zurechtzufinden, hilft es, die Bedeutung der Standardverzeichnisse zu kennen. Nachfolgende Tabelle listet die Standardverzeichnisse auf:

Verzeichnis	Bedeutung
/bin	Kommandos und Programme, die zum System gehören. In diesem Verzeichnis befinden sich die ausführbaren Programme der meisten Alltagskommandos wie `cat`, `cp`, `mv`, `ln`, `ls`, `mkdir`, `rm`, `rmdir`, `tar` usw.
/boot	Ablageort des Betriebssystem-Kernels.
/dev	Gerätedateien, etwa für Festplatten, CD-Laufwerke, Drucker, Modem. Die Dateien haben keinen eigenen Inhalt, sondern dienen lediglich als nominelle Schnittstelle zu den jeweiligen Gerätetreibern. Besondere Bedeutung hat ***/dev/null***: Dies ist der »Papierkorb« von Linux. Dorthin verschobene Dateien oder Kommandoausgaben gehen verloren.
/etc	Zentrale Konfigurationsdateien, z. B. die Dateien ***passwd***, in welcher alle am System eingerichteten Benutzer gespeichert sind, ***group***, worin die eingerichteten Benutzergruppen gespeichert sind, ***crontab*** für automatisch auszuführende Scriptaufrufe, Programme oder Kommandos, ***hosts*** für eingerichtete Hostnamen-IP-Zuordnungen, und bei einigen Linux-Distributionen auch die zentrale Systemkonfigurationsdatei ***rc.conf***.

Tabelle 10.1: Linux-Standardverzeichnisse

Verzeichnis	Bedeutung
/home	Heimatverzeichnisse der eingerichteten Benutzer. Jeder Benutzer erhält beim Einrichten neben Benutzername und Passwort unterhalb dieses Verzeichnisses ein benutzereigenes Heimatverzeichnis, wobei der Verzeichnisname dem Benutzernamen entspricht. Unterhalb seines Heimatverzeichnisses kann jeder Benutzer seine persönlichen Dateien verwalten – und nur dort sollte er es!
/lib	Von mehreren Kommandos oder Programmen genutzte Code-Bibliotheken. In etwa vergleichbar mit den unter MS Windows bekannten Verzeichnissen *[windows]/system* und *[windows]/system32*.
/lost+found	Systemverzeichnis, in dem bei einem Systemcheck des Dateisystems Dateien gespeichert werden, die nicht korrekt gelöscht werden konnten.
/proc	Virtuelles Dateisystem mit Informationen über gerade laufende Prozesse und verfügbare Hardware. Für jeden gerade laufenden Prozess gibt es ein numerisch benanntes Verzeichnis. Die Nummer entspricht der Prozess-ID (PID).
/sbin	Kommandos und Programme, die beim Systemstart ausgeführt werden oder nur durch den Administrator (Benutzer mit root-Berechtigung) ausführbar sind.
/tmp	Verzeichnis für temporäre Dateien, wie sie von zahlreichen Programmen während der Ausführung angelegt werden. Auf vielen Systemen werden in diesem Verzeichnis beispielsweise auch Session-Dateien gespeichert, die von PHP in Verbindung mit Session-Programmierung erzeugt werden.
/usr	Für Benutzer relevante Programme und Anwendungen sowie für alle oder viele Benutzer relevante Ressourcen. Unter */usr/bin* liegen Programme und unter */usr/lib* von solchen Programmen gemeinsam genutzte Bibliotheken. Manche Provider richten den Apache Webserver so ein, dass unter Verzeichnissen wie */usr/home/httpd* oder */usr/home/web* das Startverzeichnis für Webdokumente zu finden ist. Dies ist allerdings vom Ansatz her falsch, da unterhalb von */usr* eigentlich keine veränderbaren Daten liegen sollten.
/var	Im Gegensatz zu */usr* liegen unterhalb von */var* alle für alle oder viele Benutzer veränderbaren Daten sowie Daten, die von Kommandos, Prozessen oder Servern laufend verändert werden, wie etwa Log-Dateien (unter */var/log*) oder Dateien laufender Prozesse (unter */var/run*). Oftmals werden hier auch eingehende Mails (*/var/mail*) abgelegt und bei Root-Servern ist es der eigentlich optimale Ort, um das Startverzeichnis für Webdokumente zu definieren (z.B. unter */var/www*).

Tabelle 10.1: Linux-Standardverzeichnisse (Fortsetzung)

Der Tabelle ist zu entnehmen, dass die Verzeichnisstruktur also diverse Verzeichnisse enthält, in denen Sie als Benutzer keine Dateien ablegen können oder sollten. Alle persönlichen Dateien sollten Sie stets unterhalb von */home/[IhrBenutzerverzeichnis]* ablegen. Dort können Sie auch nach Belieben weitere Unterverzeichnisstrukturen mit Ihren eigenen Namensvorstellungen anlegen. Wenn Sie ein Virtual-Hosting-Angebot bei einem Provider unterschrieben haben, sehen Sie bei einem Zugang über SSH meist gar nichts anderes als Ihr persönliches Benutzerverzeichnis, d.h., ins Wurzelverzeichnis kommen Sie gar nicht erst. Unterhalb Ihres Benutzerverzeichnisses liegt dann auch das Startverzeichnis für Webdokumente (*document root*).

Bei einem gemieteten Root-Server liegen die Webdaten dagegen meist unterhalb von */var*, weil es allgemeine Daten sind, an denen häufig mehrere Benutzer arbeiten, und die auch durch Scripts oder Programme dynamisch verändert werden können.

Bei Konfigurationsarbeiten müssen Sie hin und wieder Dateien im Verzeichnis */etc* editieren.

10.2 Arbeiten auf Shell-Ebene

Nach den allgemeinen Vorbemerkungen und Übersichten ist es nun Zeit, sich mit der Shell und ihren wichtigsten Kommandos vertraut zu machen. Wir behandeln dabei die am weitesten verbreitete Bourne Again Shell (*bash*). Die meisten Aussagen sind jedoch auf andere Shells übertragbar.

10.2.1 Prompt und Eingabe-Features

Sobald Sie als Benutzer auf einem Linux-System angemeldet sind und sich in einer Shell befinden, egal ob lokal oder über SSH auf einem entfernten Rechner, sehen Sie eine Eingabeaufforderung in Form eines blinkenden Cursors. Davor kann ein Aufforderungstext oder ein Aufforderungszeichen stehen, manchmal auch Informationen wie aktuelles Verzeichnis usw. Diese Eingabeaufforderung wird als Prompt bezeichnet.

Prompt ändern

Den Inhalt vor dem blinkenden Cursor können Sie selbst bestimmen. Geben Sie an der Eingabeaufforderung beispielsweise Nachfolgendes ein und bestätigen Sie mit der Enter-Taste:

```
PS1="Input> "
```

Damit ändert sich der Aufforderungstext vor dem Cursor zu `Input>`.

Das obige »Kommando« ist eigentlich kein Kommando. Stattdessen weisen Sie einer Umgebungsvariablen einen neuen Wert zu. Der benutzereigene Standard-Prompt wird in der Umgebungsvariablen `PS1` gespeichert. Durch die Syntax `Variable=Wert` können Sie Umgebungsvariablen am Prompt einen anderen Wert zuweisen.

Neben statischem Text können Sie in den Eingabeaufforderungstext auch variable Inhalte aufnehmen. Die Syntax der Wertzuweisung an PS1 wird dabei allerdings etwas komplexer. Ein Beispiel:

```
PS1="\u@\h \w > "
```

Anschließend könnte die Eingabeaufforderung dann so aussehen:

Stefan@localhost /home/stefan >

»Könnte« aus dem Grund, weil die Eingabeaufforderung nun Variablen enthält: `\u` steht für den Benutzernamen, `\h` für den Hostnamen und `\w` für das aktuelle Verzeichnis. Auch die Schriftfarbe lässt sich ändern, und zwar mit Hilfe so genannter Escape-Sequenzen. Beispiel:

```
PS1="\\033[34m\]\u@\h \w >\\033[0m\] "
```

Anschließend erscheint der Aufforderungstext in blauer Schriftfarbe. Entscheidend dafür sind die Angaben in eckigen Klammern. Folgende Werte stehen zur Verfügung:

Wert	Bedeutung	Wert	Bedeutung
`[0m\]`	normale Schrift		
`[1m\]`	fett	`[22m\]`	nicht fett
`[4m\]`	unterstrichen	`[24m\]`	nicht unterstrichen
`[5m\]`	blinkend	`[25m\]`	nicht blinkend
`[7m\]`	invers	`[27m\]`	nicht invers
`[39m\]`	Vordergrund normal	`[49m\]`	Hintergrund normal
`[30m\]`	Vordergrund schwarz	`[40m\]`	Hintergrund schwarz
`[31m\]`	Vordergrund rot	`[41m\]`	Hintergrund rot
`[32m\]`	Vordergrund grün	`[42m\]`	Hintergrund grün
`[33m\]`	Vordergrund gelb	`[43m\]`	Hintergrund gelb
`[34m\]`	Vordergrund blau	`[44m\]`	Hintergrund blau
`[35m\]`	Vordergrund magenta	`[45m\]`	Hintergrund magenta
`[36m\]`	Vordergrund zyan	`[46m\]`	Hintergrund zyan
`[37m\]`	Vordergrund weiß	`[47m\]`	Hintergrund weiß

Tabelle 10.2: Werte für Escape-Sequenzen zur Änderung des Prompt-Textes

Um etwa eine Vorder- und Hintergrundfarbe gleichzeitig festzulegen, sind auch Eingaben wie `[37;45m\]` erlaubt.

Die Änderung der Variablen `PS1` gilt, wenn sie am Prompt neu gesetzt wird, nur für die aktuelle Sitzung. Um den Wert dauerhaft zu ändern, müssen Sie in Ihrem persönlichen Benutzerverzeichnis, in das Sie schnell durch das Kommando `cd $HOME` wechseln können, die Datei *.bashrc* editieren. Dies ist die Konfigurationsdatei für Ihre persönliche Shell, die eingelesen wird, wenn Sie sich erfolgreich am System anmelden.

Shell-History und Editierhilfen am Prompt

Die meisten Shells, darunter auch die bash, bieten eine bequeme Möglichkeit an, in der Liste bisher eingegebener Kommandos zu blättern. Benutzen Sie dazu einfach die Pfeiltasten nach oben bzw. unten. Sehr nützlich ist dieses Feature, wenn Sie bereits benutzte, komplexere Kommandos noch einmal eingeben möchten.

Innerhalb eines Kommandos, das Sie gerade eingeben, können Sie sich ähnlich wie in Editoren und Textverarbeitungsprogrammen mit den Pfeiltasten für links und rechts bewegen, ohne zu löschen. An der Stelle des Cursors können Sie Zeichen einfügen. Bereits vorhandene Zeichen dahinter wandern dann automatisch weiter.

Mit der Backspace-Taste können Sie wie üblich Eingaben zeichenweise rückwärts löschen.

Für tippfaule Zeitgenossen interessant ist auch die Verwendung der ⇆-Taste. Damit können Sie längere Namen von Verzeichnissen oder Kommandos ergänzen lassen. Wenn Sie beispielsweise `cd /lo` eingeben und dann die ⇆-Taste drücken, ergänzt die Shell automatisch zu `cd /lost+found`.

Temporäre Variablen

Eine andere Möglichkeit, häufig verwendete Kommandos schnell zu wiederholen, besteht darin, sich eigene Variablen zu definieren. Beispiel:

```
web="ls -la /var/www/docs"
```

Damit wird eine Variable namens `web` definiert. Als Inhalt erhält sie einen Aufruf des ls-Kommandos für die detaillierte Ausgabe des Verzeichnisinhalts von */var/www/docs*. Das in `web` gespeicherte Kommando können Sie nun für die Dauer der aktuellen Sitzung so aufrufen:

```
$web
```

Wichtig ist das führende Dollarzeichen.

Mit `unset web` können Sie eine selbst definierte Variable `web` wieder löschen.

Groß- und Kleinschreibung

Ein wichtiger Unterschied zwischen einem Linux-Shellprompt und der Windows-Eingabeaufforderung ist, dass die Linux-Shell strikt zwischen Groß- und Kleinschreibung unterscheidet. Eine Eingabe wie `ls -a` ist etwas anderes als `ls -A`!

10.2.2 Kommandos, Optionen und Parameter

Es erleichtert das Verständnis der manchmal etwas kryptischen Zeichenfolgen, aus denen ein typisches Linux-Kommando besteht, wenn man den generellen Aufbau und die Möglichkeiten von Kommandos kennt.

Der generelle Aufbau von Kommandos lautet:

```
Kommando [-Optionen] [Parameter]
```

Kommandos ohne Optionen und Parameter werden gleich nach dem Kommandonamen mit der Enter-Taste abgeschickt. Optionen werden durch Leerzeichen getrennt hinter dem Kommandonamen notiert. Parameter folgen, ebenfalls durch Leerzeichen getrennt, dahinter. Mehrere Parameter werden ebenfalls durch Leerzeichen getrennt.

Ein paar typische Ausprägungen:

- `ls` ist ein Kommando ohne Optionen und ohne Parameter. Es listet einfach die Einträge im aktuellen Verzeichnis auf.
- `ls -l` ist ein Kommando mit einer Option, aber ohne Parameter. Es listet die Einträge im aktuellen Verzeichnis mit Detailangaben auf.
- `ls -la` ist ein Kommando mit zwei Optionen und ohne Parameter. Es listet die Einträge detailliert auf (Option `l`) und berücksichtigt außerdem Dateien, die mit einem Punkt anfangen (Option `a`).
- `ls -la /var/www` ist ein Kommando mit zwei Optionen und einem Parameter. Es listet die Einträge des als Parameter übergebenen Verzeichnisses mit dem absoluten Pfadnamen */var/www* auf.
- `cp /var/www/index.php $HOME/save/web` ist ein Kommando ohne Optionen, jedoch mit zwei Parametern. Es kopiert eine Datei *index.php* aus dem Verzeichnis */var/www* in ein Verzeichnis *web*, das wiederum ein Unterverzeichnis von *save* ist, welches sich im persönlichen Benutzerverzeichnis befindet.

Optionen sind in der klassischen Kurzschreibweise immer an dem führenden Minuszeichen erkennbar. Die Option besteht in der Regel aus einem Buchstaben. Achten Sie gerade bei Optionsbuchstaben auf Groß- und Kleinschreibung, denn einige Kommandos haben so viele mögliche Optionen, dass es welche sowohl in Klein- als auch in Großbuchstaben gibt. Die Buchstaben mehrerer Optionen werden ohne Leerzeichen oder weitere Minuszeichen direkt aneinander gehängt. Es entsteht also ein »Optionen-String«.

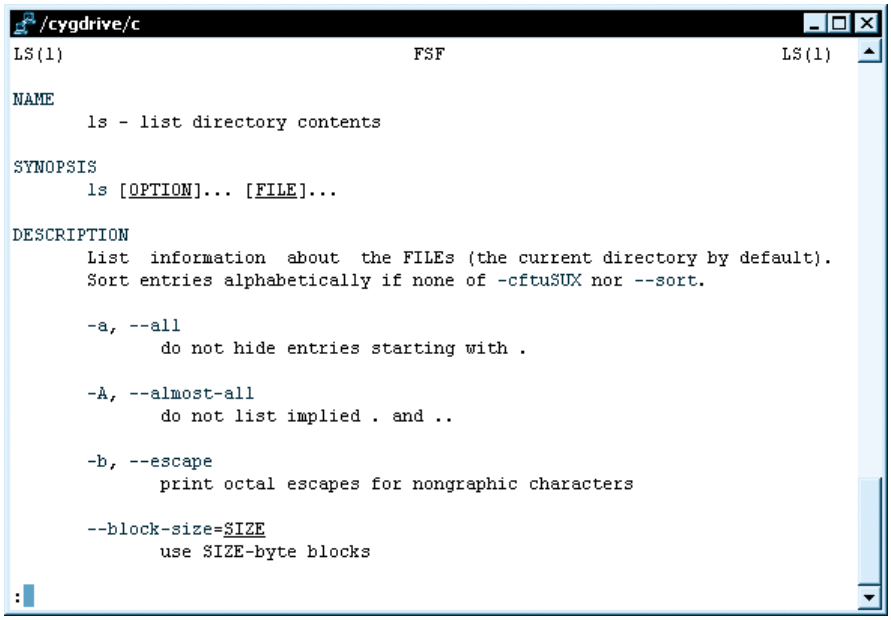


Abbildung 10.4: Optionen in Kurz- und Langschreibweise, aufgelistet mit man

Neben der klassischen Kurzschreibweise hat sich mittlerweile eine Langschreibweise für Optionen etabliert. So hat beispielsweise `ls --all` den gleichen Effekt wie `ls -a`. Diese auch als GNU-Optionen bezeichnete Schreibweise beginnt mit zwei Minuszeichen, gefolgt vom Namen der Option. In der Regel wird diese Schreibweise nur dann benutzt, wenn maximal eine Option an das Kommando übergeben wird.

Hilfe zu Kommandos

Wenn Sie ein Kommando kennen, aber die genaue Syntax oder die möglichen Optionen nicht mehr kennen, können Sie die so genannten Manpages bemühen. Das ist eine Kurzreferenz zu den einzelnen Kommandos. Erwarten Sie allerdings keine benutzerfreundliche Dokumentation davon. Die *Manpages* (*man* steht für *manual*, also Handbuch) enthalten so gut wie keine Beispiele und konzentrieren sich auf die systematische Beschreibung von Optionen und Parametern. Am Shell-Prompt erhalten Sie Hilfe zu einem Kommando wie `ls`, indem Sie eingeben:

```
man ls
```

Mit den Pfeiltasten auf und ab können Sie im Inhalt der Hilfeseiten scrollen und mit ↑ bzw. ↓ können Sie seitenweise blättern. Um eine Hilfeseite wieder zu verlassen, geben Sie einfach `q` ein (für *quit*).

Noch informativer und ausführlicher, aber nicht so universell verfügbar, ist das *info*-System. Probieren Sie einfach aus:

```
info ls
```

Weiterführender Link:

Deutschsprachige Manpages zu den einzelnen Linux-Kommandos:
http://www.infodrom.org/projects/manpages-de/

10.2.3 Umleitungen und Pipes

Manche Kommandos erzeugen längere Ausgaben. In solchen Fällen ist es praktisch, die Ausgabe in einer Datei zu speichern. Beispiel:

```
ls -l /var/run > $HOME/var-run.txt
```

In diesem Beispiel wird die detaillierte Ausgabe des Verzeichnisinhalts von */var/run* nicht einfach ins Terminalfenster geschrieben, wie es normalerweise der Fall wäre, sondern in eine Datei namens *var-run.txt* umgeleitet. Die Datei soll im persönlichen Benutzerverzeichnis gespeichert werden.

Eine einfache Umleitung in eine Datei wird mithilfe des Größer-als-Zeichens > realisiert. Davor steht das Kommando, das eine Ausgabe erzeugt, und dahinter das Ausgabeziel.

Im Beispiel wird die Datei *var-run.txt* automatisch neu erzeugt, falls sie vorher noch nicht vorhanden war. War sie bereits vorhanden, wird ihr Inhalt komplett neu überschrieben. Letzteres geht auch anders. Durch zwei Größer-als-Zeichen (>>), erreichen Sie, dass die Ausgabe des Kommandos ans Ende einer bereits vorhandenen Datei angehängt wird. Ist die Zieldatei dagegen noch nicht vorhanden, wird sie wie bei > automatisch neu angelegt.

Auch der umgekehrte Weg ist möglich. Der Input kann statt von der Eingabezeile am Prompt aus einer Datei eingelesen werden. Beispiel:

```
sort < /etc/hosts
```

Das sort-Kommando ist in der Lage, mehrere Zeilen zu sortieren. Als Input wird jedoch typischerweise eine zeilenweise zu sortierende Textdatei übergeben, in unserem Beispiel die Datei *hosts* im Verzeichnis */etc*.

Noch spannender sind die Möglichkeiten, die Ausgabe eines Kommandos als Input für ein anderes Kommando zu benutzen. Dazu dienen die so genannten *Pipes*. Das wohl bekannteste Beispiel ist Folgendes:

```
ls -l /dev | more
```

Pipes werden durch einen einfachen Senkrechtstrich | notiert. Vor dem Pipe-Zeichen steht ein Kommando, dessen Ausgabe durch das Kommando hinter dem Pipe-Zeichen gefiltert, manipuliert oder variiert wird – je nachdem, was das hintere Kommando tut. Im Beispiel wird ein Detail-Listing des ziemlich umfangreichen Verzeichnisses */dev* durch das Kommando `more` gefiltert. Das Kommando `more` gibt normalerweise den Inhalt einer Datei aus (z. B. `more beispiel.txt`) und erlaubt es dabei, den Inhalt »seitenweise« mit der Leertaste anzuzeigen. Als Kommando hinter einem Pipe-Zeichen eingesetzt, wird es jedoch nicht auf eine Datei angewendet, sondern auf die Bildschirmausgabe des Kommandos vor dem Pipe-Zeichen.

Wir haben hier bewusst einfache Beispiele gewählt, um die Funktionsweise der Umleitungen und Pipes zu erläutern. Im Zusammenspiel mit komplexen und leistungsstarken Einzelkommandos offenbart sich jene Mächtigkeit der Shell, die von Spezialisten so geschätzt wird. Ein einziges zusammengesetztes Shell-Kommando kann extrem präzise Ergebnisse liefern.

10.2.4 Kommandos für Benutzerverwaltung

Bevor Sie sich auf Shell-Ebene auf die Dateiverwaltung stürzen, sollten Sie sich mit der Benutzerverwaltung vertraut machen. Das gilt besonders dann, wenn Sie bei einem Hosting-Provider einen Root-Server angemietet haben.

Benutzer und Gruppen

Linux unterscheidet Benutzer (*user*) und Gruppen (*groups*), die aus Benutzern bestehen. Für real existierende Personen, die auf einem Linux-System arbeiten sollen, können Benutzerkonten eingerichtet werden. Daneben gibt es nach jeder Linux-Installation aber

auch diverse nicht menschliche Benutzer, die intern zum Ausführen von Anwendungen usw. benötigt werden.

Nach einer Neuinstallation eines Linux-Systems hat der erste und oberste Benutzer den Benutzernamen *root*. Dieser Benutzer gehört unter anderem einer gleichnamigen Gruppe *root* an. Der Benutzer *root* kann nun nach Belieben weitere Gruppen anlegen, Benutzer anlegen und jeden Benutzer einer oder mehreren Gruppen zuweisen. Gehört ein Benutzer mehreren Gruppen an, gilt eine Gruppe als Primärgruppe. Benutzer, die der Gruppe *root* zugeordnet werden, haben ebenso wie der Benutzer *root* selbst root-Rechte und gehören damit zu den Systemadministratoren.

Das klingt alles noch recht abstrakt. Interessant wird das Benutzer- und Gruppenkonzept erst in Verbindung mit Rechten auf Verzeichnisse und Dateien. Jedes Verzeichnis und jede Datei gehören nach Erzeugung demjenigen Benutzer, der das Verzeichnis oder die Datei erzeugt hat. Außerdem gehören sie der Gruppe, welche die Primärgruppe des Benutzers ist. Für jedes Verzeichnis und jede Datei lässt sich nun festlegen, welche Rechte der Benutzer selbst daran hat, welche Rechte andere Mitglieder der gleichen Primärgruppe daran haben und welche Rechte der »Rest der Welt« hat, also andere Benutzer, die nicht der gleichen Primärgruppe angehören. Zur Verfügung stehen Leserecht, Schreibrecht und bei ausführbaren Dateien wie Programmen, Scripts oder Kommandos auch Ausführrecht. Auf die Dateirechte gehen wir in Zusammenhang mit den Kommandos zur Dateiverwaltung (***Abschnitt 10.2.5***) noch näher ein.

Wenn Sie bei einem Hosting-Provider einen Root-Server angemietet haben, erhalten Sie häufig nur das Passwort für den Benutzer *root*, also für den Administratorzugang. Benutzername ist dann *root* und das Passwort jenes, das Sie erhalten haben.

In der Unix-Welt gibt es ein paar wichtige Regeln, die zum eigenen Schutz, zum Schutz anderer auf dem Rechner arbeitender Benutzer und zum Schutz von Daten sinnvoll sind:

- Ändere deine Passwörter regelmäßig und so, dass sie niemand kennt und niemand erraten kann.
- Melde dich bei Fernzugang nicht als Benutzer *root* an.
- Arbeite nur als Benutzer *root*, wenn es die Arbeit erfordert.

Beim ersten Anmelden an einem Root-Server, für den Sie nur die Benutzerdaten für *root* erhalten haben, müssen Sie zumindest mal gegen die zweite der genannten Regeln verstoßen. Um sich an die Regeln zu halten, sollten Sie jedoch baldmöglichst das erhaltene Passwort für den Benutzer *root* ändern und außerdem sinnvolle, Ihrem Projektbedarf entsprechende Benutzer anlegen, die keine root-Rechte haben.

Passwort ändern

Das Passwort des Benutzers, als der Sie am System angemeldet sind, können Sie am Shell-Prompt ändern durch Eingabe von:

```
passwd
```

Sie werden aufgefordert, ein neues Passwort einzugeben. Anschließend müssen Sie die Eingabe wiederholen. Beide Eingaben erfolgen aus Sicherheitsgründen verdeckt, also ohne Echo am Bildschirm. Groß- und Kleinschreibung von Buchstaben wird unterschieden. Das Passwort darf neben Buchstaben und Ziffern auch Sonderzeichen enthalten.

Verwenden Sie als Passwort keine Wörter, die in Wörterbüchern vorkommen, und keine Namen oder Zahlen, die mit Ihrer Person oder Ihrem persönlichen Umfeld in Verbindung gebracht werden können (Geburtstag, Hochzeitstag, Hausnummer usw.). Dennoch muss das Passwort keine sinnlose, kaum zu merkende Zeichenfolge sein. Sinnvoll sind beispielsweise Kombinationen von Wörtern, Abkürzungen, Zahlen und Sonderzeichen, z. B. `bein:nasa3` oder `nt§super18`. Auch rein lautmalerische Phantasiewörter wie `bakumilati` oder `klupliklau` sind noch akzeptabel. Wichtig ist, dass Sie sich das Passwort merken können und es nirgendwo notieren müssen.

Als Benutzer ***root*** können Sie nicht nur Ihr eigenes Passwort ändern, sondern auch das anderer Benutzer. Angenommen, es gibt einen eingerichteten Benutzer namens ***stefan***. Dann können Sie dessen Passwort ändern durch Eingabe von:

```
passwd stefan
```

Gruppen und Benutzer anlegen

Das Ändern des root-Passworts und das Ändern von Passwörtern von anderen Benutzern ist eine der Arbeiten, für die root-Kennung erforderlich ist. Das Gleiche gilt für das Anlegen von Gruppen und anderen Benutzern.

Bevor Sie andere Benutzer anlegen, sollten Sie zunächst geeignete Gruppen anlegen. Wenn nicht Dutzende von Benutzern mit ganz verschiedenen Aufgaben auf dem System arbeiten sollen, genügt meist schon eine Gruppe für normale Benutzer.

Zum Anlegen von Gruppen gibt es die Kommandos `groupadd` oder `addgroup`. `groupadd` ist dann verfügbar, wenn auf dem System die Shadow Suite installiert ist (was bei ordentlich vorkonfigurierten Systemen der Fall sein sollte). Probieren Sie im Zweifelsfall beide Kommandos aus, um festzustellen, welches auf Ihrem System zur Verfügung steht. Nachfolgend ein Beispiel für `groupadd`:

```
groupadd webworkers
```

Damit wird eine Gruppe namens ***webworkers*** angelegt.

Neue Benutzer können Sie mit dem Kommando `useradd` oder `adduser` anlegen. `useradd` ist dann verfügbar, wenn auf dem System die Shadow Suite installiert ist. Probieren Sie auch hier im Zweifelsfall beide Kommandos aus, um festzustellen, welches auf Ihrem System zur Verfügung steht. Im Folgenden ein Beispiel für `useradd`:

```
useradd stefan
```

Das ist allerdings nur die spartanischste Form. Sinnvoller ist es, zumindest ein Passwort und die Gruppenzugehörigkeit gleich mit anzugeben. Beispiel:

```
useradd -p fun194pat -g webworkers stefan
```

Damit wird dem neuen Benutzer `stefan` das Passwort `fun194pat` zugewiesen und der Benutzer wird der Gruppe `webworkers` zugewiesen.

Sowohl `groupadd` als auch `useradd` kennen weitere Optionen. So benötigt beispielsweise jede Gruppe intern eine numerische Gruppen-ID (*gid* genannt) und jeder Benutzer eine Benutzer-ID (*uid* genannt). Wenn Sie keine ID-Nummern angeben, werden diese automatisch vergeben. Im Normalfall sollten Sie sich auf diese Automatik verlassen, um Fehler zu vermeiden. Mit der Option `-g` können Sie bei `groupadd` jedoch auch eine explizite Gruppen-ID vergeben und bei `useradd` mit `-u` eine explizite User-ID. Vergeben Sie sicherheitshalber nur Zahlenwerte oberhalb von 1024.

Für Benutzer kann es auch sinnvoll sein, ein abweichendes persönliches Heimatverzeichnis anzulegen. Beispiel:

```
useradd -p fun194pat -g webworkers -d /home/stefan_muenz stefan
```

Dieses Kommando legt für den neuen Benutzer `stefan` zusätzlich das Heimatverzeichnis `/home/stefan_muenz` fest. Normalerweise wäre es `/home/stefan`.

Falls mehrere Shells auf dem System zur Verfügung stehen, können Sie auch eine bestimmte Shell zuweisen. Beispiel:

```
useradd -p fun194pat -g webworkers -d /home/stefan_muenz -s
        /bin/csh stefan
```

Damit wird dem Benutzer zusätzlich die C-Shell zugewiesen. Angegeben werden muss der Pfadname der ausführbaren Datei der Shell.

Gruppen und Benutzer ändern oder löschen

Analog zu den Kommandos `useradd` und `groupadd` gibt es bei installierter Shadow Suite die Kommandos `usermod` zum Ändern eines Benutzers und `userdel` zum Löschen eines Benutzers sowie `groupmod` zum Ändern einer Gruppe und `groupdel` zum Löschen einer Gruppe.

Die Optionen bei `usermod` und `groupmod` sind zu großen Teilen die gleichen wie bei `useradd` und `groupadd`. Wenn Sie also beispielsweise das Heimatverzeichnis eines Benutzers ändern möchten, können Sie das wie bei `useradd` mit der Option `-d` tun.

Um den Benutzernamen eines Benutzers zu ändern, geben Sie Folgendes ein (Beispiel):

```
usermod -l stefan_muenz stefan
```

Ab nun muss sich Benutzer *stefan* mit dem Benutzernamen *stefan_muenz* anmelden.

Auch Gruppennamen können Sie ändern. Ein Beispiel:

```
groupmod -n webber webworkers
```

Damit wird die bestehende Gruppe *webworkers* in *webber* umbenannt.

Zum Löschen genügt die Angabe des Benutzer- bzw. Gruppennamens. Beispiele:

```
userdel stefan_muenz
groupdel webbers
```

Informationen über vorhandene Benutzer und Gruppen

Die einfachste Art, sich »normale« Benutzer, also keine vom System angelegten Benutzer, anzeigen zu lassen, besteht darin, sich den Inhalt des home-Verzeichnisses auflisten zu lassen:

```
ls -l /home
```

Eine Beispielausgabe lautet:

```
drwxrwxrwx  8 andreas    developer   4096 Aug 20  2004 andreas
drwxrwxrwx  4 benjamin   developer   4096 Feb 21 23:21 benjamin
drwx------  5 christian  powerusers  4096 Feb 19 15:41 christian
drwxrwxrwx  2 christoph  developer   4096 Nov 29 01:01 christoph
```

Das `d` am Anfang des ersten Blocks signalisiert, dass es sich um ein Verzeichnis handelt. Da unter /home die persönlichen Verzeichnisse der Benutzer stehen sollten, ist im Idealfall jedes Verzeichnis das eines vorhandenen Benutzers. Auf diese Weise lassen sich zumindest die vorhandenen Benutzernamen ermitteln.

Nähere Informationen bietet ein Blick in die Dateien */etc/passwd*, */etc/shadow* und */etc/group*.

```
cat /etc/passwd
cat /etc/shadow
cat /etc/group
```

Die drei Kommandos listen jeweils den Inhalt der entsprechenden Datei auf. Näheres zum Aufbau der einzelnen Zeilen der Dateien siehe ***Abschnitt 10.4.2***.

Um vorhandene Gruppen aufzulisten, steht auch folgendes Kommando zur Verfügung:

```
groups
```

Wenn Sie nur wissen möchten, welche Gruppen einem bestimmtem Benutzer zugeordnet sind, übergeben Sie an das groups-Kommando den Benutzernamen als Parameter. Beispiel:

```
groups stefan
```

Informationen über aktuell angemeldete Benutzer

Zunächst einmal kann es durchaus sinnvoll sein, sich über sich selbst Klarheit zu verschaffen, beispielsweise, wenn man Sub-Shells mit anderen Benutzerkonten gestartet hat (siehe dazu weiter unten). Um herauszufinden, als wer man selber gerade angemeldet ist, gibt es das folgende Kommando:

```
whoami
```

Dies steht für *who am I?*. Das Kommando gibt nur den Benutzernamen aus. Auskunftsfreudiger ist dagegen folgendes Kommando:

```
id
```

Es bietet zusätzlich zum Benutzernamen auch die Benutzer-ID (uid) sowie die Namen und die IDs (gid) aller Gruppen, denen man zugeordnet ist.

Wenn mehrere Benutzer an einem System arbeiten, z. B. über Fernzugänge, kann es von Bedeutung sein herauszufinden, wer gerade am System angemeldet ist. Das einfache Kommando dazu lautet:

```
users
```

Damit werden die Namen aller aktuell angemeldeten Benutzer aufgelistet. Mehr Informationen liefert:

```
who
```

Das Kommando listet für jeden aktuell angemeldeten Benutzer den Benutzernamen, die Geräteschnittstelle, über die der Benutzer Zugang hat, die Login-Zeit und den Hostnamen oder die IP-Adresse des Hosts, über den der Benutzer Internetzugang hat.

Noch informativer ist:

```
w
```

Dieses Kommando listet neben den Informationen, die auch `who` liefert, zusätzlich die von den einzelnen Benutzern verbrauchte CPU-Zeit sowie deren *idle time* (Leerlaufzeit – also die Zeit, in der Sie das System nicht beschäftigt haben) auf.

Kommunikation mit aktuell angemeldeten Benutzern

Die Shell bietet aktuell angemeldeten Benutzern die Möglichkeit, direkt über die Konsole miteinander zu kommunizieren, in einer Art primitivem Chat. Um einem anderen, aktuell angemeldeten Benutzer eine Nachricht zu senden, geben Sie zunächst ein:

```
write Benutzer
```

Dabei steht `Benutzer` für den gewünschten Benutzernamen. Nach dem Drücken der Enter-Taste können Sie eine oder mehrere Zeilen Text eingeben. Wenn Sie die letzte Zeile abgeschlossen haben (am besten noch mal Enter drücken), halten Sie die Strg-Taste gedrückt und drücken gleichzeitig C. Mit dieser Tastenkombination brechen Sie eigentlich Programme ab, doch bei `write` wird damit einfach ein »Dateiende« erzeugt und die Nachricht wird versendet. Sie erscheint beim Empfänger direkt am Shell-Prompt.

Sub-Shell mit anderem Benutzer starten

Wenn Sie auf einem neu installierten System mit root-Kennung andere Benutzer anlegen, sollten Sie auch für sich selber einen Benutzer einrichten. Mit diesem Benutzer sollten Sie sich fortan am System anmelden (das gilt auch und vor allem für Fernanmeldungen via SSH) – und nicht mehr mit der root-Kennung.

Sollten Sie während der Arbeit doch einmal die root-Kennung benötigen, können Sie einfach eine Sub-Shell für den Benutzer *root* starten:

```
su root
```

Das Kommando `su` steht für *substitute user*. Als Parameter wird ihm der Benutzername übergeben, zu dem Sie »werden« möchten. Anschließend müssen Sie das Passwort für den Benutzer *root* eingeben. Ist die Anmeldung erfolgreich, wird eine Shell gestartet, in der Sie alle root-Rechte haben.

Um eine so gestartete Sub-Shell wieder zu verlassen, geben Sie ein:

```
exit
```

Anschließend befinden Sie sich wieder am Prompt der »Eltern-Shell«, von wo aus Sie die Sub-Shell aufgerufen haben.

Wenn Sie umgekehrt als Benutzer *root* arbeiten und Sub-Shells mit anderen Benutzernamen öffnen, brauchen Sie übrigens kein Passwort einzugeben – der Benutzer *root* darf jederzeit zu jedem anderen Benutzer werden, ohne sich auszuweisen.

10.2.5 Kommandos für Dateiverwaltung

Alle Arten von persönlichen Dateien sollten Sie wie schon erwähnt unterhalb Ihres persönlichen Heimatverzeichnisses ablegen. In Systemverzeichnissen haben solche Dateien nichts zu suchen. Auch der Apache Webserver sollte so eingerichtet sein, dass die *document root*, also das Startverzeichnis für Dateien des Webprojekts, auf ein dafür übliches Verzeichnis wie */var/www* zeigt.

Das Datei- und Verzeichniskonzept von Linux

Jede Datei unter Linux stellt einen so genannten *Inode* (Informationsknoten) dar. Darin sind alle Informationen zur Datei gespeichert, wie z.B.:

- Benutzername des Eigentümers,
- Name der Eigentümergruppe,

- Zugriffsrechte, was der Eigentümer, die Eigentümergruppe und der »Rest der Welt« mit der Datei tun dürfen,
- Informationen über letzte Änderung und letzten Zugriff auf die Datei,
- Typ der Datei,
- Zeiger auf Datenblöcke, welche die Dateninhalte der Datei speichern.

Ein Verzeichnis ist auch nichts anderes als eine Datei, nämlich eine Datei des Typs »Verzeichnis«. Der Inhalt einer solchen Datei besteht in einer Liste von Dateien, die in diesem Verzeichnis gespeichert sind. Sogar Geräte wie CD-ROM, Festplatten, Netzwerkschnittstellen oder Drucker sind nichts anderes als Dateien, eben des Typs »Gerät«.

Sogar temporäre Daten wie laufende Prozesse oder Sessiondaten werden vom System im Dateisystem gespeichert, solange sie aktiv sind, und gelöscht, wenn sie nicht mehr aktiv sind. Dies hat große Vorteile, beispielsweise um eigene Tools zur Systemüberwachung zu programmieren oder um nach einem Systemabsturz den eingefrorenen Zustand analysieren zu können.

Es gibt also nur einen zentralen »Baum« im Linux-Dateisystem, wobei sich die Hierarchie durch Dateien des Typs »Verzeichnis« ergibt. Alle erreichbaren Laufwerke, Geräte und Datenträger erscheinen als Dateien in diesem einen Verzeichnisbaum.

Beim Durchstöbern der Standardverzeichnisse eines Linux-Systems wird Ihnen schnell auffallen, dass viele Dateien keine Endung haben. In der Tat gibt es in der Unix-Welt traditionell keine Verknüpfungen zwischen bestimmten Dateinamenerweiterungen und dem Typ der Datei, wie etwa in der Windows-Welt. Der Punkt war seit jeher ein ganz normales Zeichen, das in Dateinamen vorkommen durfte, aber kein Trenner zwischen eigentlichem Namen und Namenserweiterung. Erst in neuerer Zeit, etwa durch das System der Mime-Typen, halten Dateien mit Endung auch in Linux-Systemen Einzug. Dennoch gibt es weiterhin keine systemeigene Verknüpfungstabelle. Stattdessen wird der Typ der Datei in den Inode-Daten jeder Datei gespeichert. Über ein Shell-Kommando ist der Typ einer jeden Datei ermittelbar. Beispiel:

```
file index.old
```

Als Ergebnis wird der Dateityp der Datei ***index.old*** im aktuellen Verzeichnis ausgegeben.

Im Verzeichnisbaum bewegen

In andere Verzeichnisse wechseln Sie mithilfe des cd-Kommandos (***change directory***). Falls das aktuelle Verzeichnis, in dem Sie sich befinden, nicht am Shell-Prompt angezeigt wird, können Sie jederzeit ermitteln, wo Sie sich befinden. Geben Sie dazu ein:

```
pwd
```

Das Kommando steht für ***print working directory*** und gibt den absoluten Pfadnamen des aktuellen Verzeichnisses aus.

Beim cd-Kommando können Sie wahlweise absolute oder mit relative Pfadnamen angeben. Die Regeln sind die gleichen wie etwa bei lokalen Hyperlinks in HTML. Beispiele:

```
cd /
```

Mit dieser absoluten Pfadangabe wechseln Sie ins Wurzelverzeichnis des Systems.

```
cd /home/stefan
```

Das ist ebenfalls eine absolute Pfadangabe, erkennbar an dem führenden / bei der Pfadangabe. Das Kommando bewirkt einen Wechsel ins Verzeichnis */home/stefan*. Sollte das Verzeichnis nicht existieren, erhalten Sie eine Fehlermeldung.

```
cd ..
```

In dieser Form eingegeben, wechseln Sie ins Verzeichnis oberhalb des aktuellen Verzeichnisses. Wie unter Windows und DOS steht die Zeichenfolge .. für das Verzeichnis oberhalb, und . für das aktuelle Verzeichnis.

```
cd ../../stefan
```

Mit diesem Kommando wechseln Sie zwei Verzeichnisse höher und dort wiederum ins Unterverzeichnis *stefan*.

Verzeichnispfade können in Umgebungsvariablen gespeichert sein. So können Sie von jedem Punkt aus jederzeit einfach in Ihr persönliches Heimatverzeichnis wechseln durch:

```
cd $HOME
```

Verzeichnisse anlegen und löschen

Im aktuellen Verzeichnis können Sie wie folgt ein Unterverzeichnis anlegen:

```
mkdir docs
```

`mkdir` steht für *make directory*. Damit wird ein Verzeichnis namens *docs* angelegt. Ebenso können Sie aber auch absolute oder relative Pfadangaben benutzen, um Verzeichnisse an anderen Orten im Verzeichnisbaum anzulegen. Beispiele:

```
mkdir ../../stefan/fotos
mkdir /var/www/scripts
```

Leere Verzeichnisse löschen Sie mit dem rmdir-Kommando:

```
rmdir docs
```

`rmdir` steht für *remove directory*.

Falls das Verzeichnis Dateien enthält, können Sie diese im Beispiel zuvor löschen mit:

```
rm docs/*
```

Dateien suchen und filtern

Zum Auffinden von Dateien können Sie das find-Kommando benutzen. Ein Beispiel:

```
find /etc -name *.conf
```

Hinter dem Kommandonamen übergeben Sie zunächst den Pfadnamen des gewünschten Startverzeichnisses für die Suche. Es werden rekursiv alle Verzeichnisse unterhalb des angegebenen Startverzeichnisses durchsucht. Um nach Dateinamen zu suchen, geben Sie hinter dem Pfadnamen des Suchstartverzeichnisses die Option `-name` an. Dahinter folgt der gesuchte Dateiname.

Zur Spezifizierung des Dateinamens können Sie beim find-Kommando und auch bei anderen Kommandos, etwa beim ls-Kommando, mit Wildcards arbeiten. Das Sternzeichen (*) steht für keine oder beliebig viele Zeichen und das Fragezeichen (?) für ein einzelnes Zeichen. Das obige Beispiel findet alle Dateien im Verzeichnis */etc*, die auf *.conf* enden. Weitere Beispiele:

```
find / -name httpd.conf
```

Dieses Kommando durchsucht den gesamten Verzeichnisbaum beginnend vom Wurzelverzeichnis nach einer bestimmten Datei, nämlich nach *httpd.conf*.

```
find / -name httpd.conf
```

Dieses Kommando durchsucht den gesamten Verzeichnisbaum beginnend vom Wurzelverzeichnis:

```
find / -name ?.???
```

Dieses Kommando findet alle Dateien im System, die eine Dateiendung mit drei Zeichen haben und einen Namen davor, der nur ein Zeichen hat, also etwa *x.gif*.

Neben den beiden Wildcard-Zeichen gibt es auch noch die Möglichkeit, mithilfe eckiger Klammern einen Zeichenbereich anzugeben. Ein Beispiel:

```
find / -name [abc]?.g*
```

Dieses Kommando findet alle Dateien im System, die mit a, b oder c beginnen, dann noch ein weiteres Zeichen haben, dann einen Punkt und dahinter eine Dateiendung, beginnend mit g.

```
find / -name [A-Za-z].*
```

Dieses Kommando findet alle Dateien, deren Name nur aus einem Groß- oder Kleinbuchstaben besteht und aus einer beliebigen Dateiendung.

Das find-Kommando ist noch wesentlich mächtiger, als Dateien nur nach Namen oder Namensteilen zu suchen. Zum einen kann `find` auch nach ganz anderen Kriterien wie

Zeitstempel der Datei, Dateigröße oder Zugriffsrechten suchen und zum anderen kann ihm gleich noch ein anderes Dateikommando mit übergeben werden, welches dann auf alle gefundenen Dateien angewendet wird.

Zunächst einmal zur Suche nach anderen Dateikriterien. Dazu ein paar Beispiele:

```
find /var/www -mtime -7
```

Dieses Kommando findet unterhalb von */var/www* alle Dateien, deren Inhalt innerhalb der letzten 7 Tage geändert wurde. Es gibt noch mehr von diesen Zeitoptionen. Nachfolgende Tabelle listet die Unterschiede auf:

Option	Bedeutung
`-mtime -n`	Dateien, die innerhalb der letzten `n` * 24 Stunden geändert wurden.
`-mtime +n`	Dateien, die vor mehr als `n` * 24 Stunden geändert wurden.
`-mtime n`	Dateien, die vor genau `n` * 24 Stunden geändert wurden.
`-mmin -n`	Dateien, die innerhalb der letzten `n` Minuten geändert wurden.
`-mmin +n`	Dateien, die vor mehr als `n` Minuten geändert wurden.
`-mmin n`	Dateien, die vor genau `n` Minuten geändert wurden.
`-atime -n`	Dateien, auf die innerhalb der letzten `n` * 24 Stunden das letzte Mal zugegriffen wurde.
`-atime +n`	Dateien, auf die vor mehr als `n` * 24 Stunden das letzte Mal zugegriffen wurde.
`-atime n`	Dateien, auf die vor genau `n` * 24 Stunden das letzte Mal zugegriffen wurde.
`-amin -n`	Dateien, auf die innerhalb der letzten `n` Minuten das letzte Mal zugegriffen wurde.
`-amin +n`	Dateien, auf die vor mehr als `n` Minuten das letzte Mal zugegriffen wurde.
`-amin n`	Dateien, auf die vor genau `n` Minuten das letzte Mal zugegriffen wurde.
`-ctime -n`	Dateien, deren Status sich innerhalb der letzten `n` * 24 Stunden das letzte Mal geändert hat.
`-ctime +n`	Dateien, deren Status sich vor mehr als `n` * 24 Stunden das letzte Mal geändert hat.
`-ctime n`	Dateien, deren Status sich vor genau `n` * 24 Stunden das letzte Mal geändert hat.
`-cmin -n`	Dateien, deren Status sich innerhalb der letzten `n` Minuten das letzte Mal geändert hat.
`-cmin +n`	Dateien, deren Status sich vor mehr als `n` Minuten das letzte Mal geändert hat.
`-cmin n`	Dateien, deren Status sich vor genau `n` Minuten das letzte Mal geändert hat.

Tabelle 10.3: find-Zeitoptionen

Andere Optionen funktionieren ähnlich – beispielsweise das Suchen von Dateien, abhängig von ihrer Größe. Ein Beispiel:

```
find /var/www -size +200k
```

Das Beispielkommando findet alle Dateien unterhalb von */var/www*, die größer als 200 Kbyte sind. Ein anderes Beispiel:

```
find /var/www -size -100c
```

Dieses Kommando findet alle Dateien unterhalb von */var/www*, die weniger als 100 Byte haben.

```
find /var/www -size 1080c
```

Dieses Kommando findet alle Dateien unterhalb von */var/www*, die genau 1080 Byte haben. Auch bei `-size` können Sie also explizit positive Werte (für »Dateien größer als«), negative Werte (für »Dateien kleiner als«) und vorzeichenlose Werte (für »Dateien mit genau der angegebenen Größe«) angeben. Hinter dem Wert kann ein kleines k (für »Wert wird als Anzahl Kilobyte« interpretiert) oder ein kleines c (für »Wert wird als Anzahl Byte« interpretiert) folgen.

Auch nach dem Eigentümer oder der Gruppenzugehörigkeit von Dateien können Sie suchen. Beispiele:

```
find /var/www -user stefan
find /var/www -group webworkers
```

Das erste Kommando findet alle Dateien, für die Benutzer *stefan* der Eigentümer ist und das zweite Kommando alle Dateien, die der Gruppe *webworkers* gehören.

Und noch ein letztes Beispiel für Suchoptionen:

```
find /var/www -perm 751
```

Dieses Kommando findet unterhalb von */var/www* alle Dateien, deren Zugriffsrechte auf oktal 751 gesetzt sind (siehe dazu auch ***Abschnitt 10.2.6***).

In den bisherigen Beispielen gibt das find-Kommando einfach nur die Namen der gefundenen Dateien aus. In vielen Fällen sind jedoch weitere Details sinnvoll. Dazu können Sie das find-Kommando mit dem ls-Kommando über die exec-Option verbinden. Ein Beispiel:

```
find /var/www -size +200k -exec ls -l {} \;
```

Dieses Beispiel findet wie schon weiter oben beschrieben unterhalb von */var/www* alle Dateien mit einer Größe von über 200 Kbyte. Auf die gefundenen Dateien wird das Kommando `ls -l` angewendet, welches detailliert über die Dateien Auskunft gibt. Um ein solches Kommando mit dem find-Kommando zu verknüpfen, notieren Sie hinter der Suchspezifikation `-exec`, gefolgt vom gewünschten Kommando. Die leeren geschweiften Klammern `{}` sind nötig, um anzugeben, dass das Kommando auf jede einzelne gefundene Datei angewendet werden soll. Am Ende eines so verknüpften Kommandos muss dann noch ein Leerzeichen, ein Backslash `\` und ein Semikolon notiert werden.

Hüten Sie sich als unerfahrener Benutzer jedoch davor, Kommandos wie `rm` (Dateien löschen) mit `find` zu verknüpfen. Dann genügt schon ein Tippfehler oder eine kleine Unachtsamkeit bei der Kommandoeingabe, um eine Katastrophe auszulösen.

Dateien kopieren

Zum Kopieren von Dateien steht das cp-Kommando zur Verfügung:

```
cp index.html index_old.html
```

Dieses Kommando legt von der Datei *index.html* eine Kopie unter dem Namen *index_old.html* an. Beide Dateien befinden sich dabei im gleichen, nämlich dem aktuellen Verzeichnis.

Wenn Sie mehrere Dateien auf einmal kopieren, muss das Kopierziel ein Verzeichnis sein. Mehrere Dateien können Sie entweder explizit hintereinander eingeben oder mithilfe von Wildcards auswählen. Beispiele:

```
cp index.html impressum.html /var/www/save
cp *.html /var/www/save
cp neu/*.html /var/www
```

Im ersten Fall werden die beiden genannten HTML-Dateien ins Verzeichnis mit dem absoluten Pfadnamen */var/www/save* kopiert und im zweiten und dritten Fall alle Dateien mit Endung *.html*. Im dritten Fall wird die Quelle relativ adressiert, nämlich alle HTML-Dateien im Unterverzeichnis *neu* aus Sicht des aktuellen Verzeichnisses.

Sie können auch ganze Verzeichnisbäume auf einmal kopieren. Beispiel:

```
cp -R /var/www /var/backup
```

Die Option `-R` bewirkt, dass der gesamte Verzeichnisbaum unterhalb von */var/www* unter */var/backup* als Kopie angelegt wird.

Um eine Datei aus einem anderen Verzeichnis ins aktuelle Verzeichnis zu kopieren, benutzen Sie beim Kopierziel einfach die Punktsyntax zur Adressierung des aktuellen Verzeichnisses:

```
cp save/index.html .
```

Dateien umbenennen und verschieben

Ganz ähnlich funktioniert das Kommando `mv` (für *move*), das sowohl zum Umbenennen als auch zum Verschieben dient:

```
mv index.html index_old.html
```

Damit wird die Datei *index.html* nach *index_old.html* umbenannt, da sie innerhalb des gleichen Verzeichnisses verschoben wird. Auch diese Variante bewirkt de facto eine Umbenennung:

```
mv /var/www/index.html /var/www/index_old.html
```

Ist der zweite Parameter ein Verzeichnis, wird die Datei unter ihrem bisherigen Namen in das andere Verzeichnis verschoben. Beispiel:

```
mv index.html save
```

Damit wird die Datei *index.html* ins Unterverzeichnis *save* verschoben. Oder das Gleiche mit absoluten Pfadnamen:

```
mv /var/www/index.html /var/www/save
```

Auch beim mv-Kommando kann mit mehreren Quelldateien oder Wildcards gearbeitet werden. Das Ziel muss in diesem Fall ein Verzeichnis sein, es darf sich also nur um eine Verschiebeaktion handeln.

Dateien löschen

Das allgemeine Löschkommando lautet `rm`:

```
rm index_old.html
```

Damit wird die Datei *index_old.html* im aktuellen Verzeichnis gelöscht. Absolute und relative Pfadangaben sind selbstverständlich erlaubt. Auf Verzeichnisse ist das Kommando jedoch nicht anwendbar – dafür gibt es das bereits früher beschriebene rmdir-Kommando.

Auch die Angabe mehrerer expliziter Dateien und Wildcards sind bei `rm` erlaubt. Sogar die Option `-R` wie beim Kopieren ist möglich. Allerdings sollten Sie sich beim rekursiven Löschen vorher genau überlegen, was Sie tun. Nur allzu leicht gehen Daten durch solche Aktionen verloren.

Wenn Sie mehrere Dateien auf einmal löschen, kann es sein (das ist in Ihrer Benutzerkonfiguration einstellbar), dass Sie jede Datei einzeln bestätigen müssen. Drücken Sie dabei auf [y], um die jeweils abgefragte Datei zu löschen, und auf [n], um die Datei nicht zu löschen. Um die Abfrage zu verhindern, können Sie die Option `-f` verwenden:

```
rm -f temp/*
```

Damit löschen Sie alle Dateien im Unterverzeichnis *temp* und zwar ohne Rückfrage bzw. ohne Einzelbestätigung.

Inhalte von Dateien anzeigen

Über einen textorientierten Zugang wie am Shell-Prompt können natürlich nur Dateien mit Textinhalt sinnvoll angezeigt werden. Dazu kursieren mittlerweile mehrere Kommandos. Das älteste lautet `cat` und listet den Inhalt einer Datei in einem Rutsch auf. Anschließend erscheint automatisch wieder der Shell-Prompt. Beispiel:

```
cat save/index.html
```

Das Kommando gibt den Inhalt der Datei *index.html* aus dem Unterverzeichnis *save* aus.

Ideal geeignet ist das cat-Kommando, wenn seine Ausgabe als Input für andere Kommandos dienen soll, z. B.:

```
cat namensliste.txt | sort
```

Damit wird der Inhalt der Datei *namensliste.txt* zeilenweise sortiert ausgegeben.

Zum Betrachten von Inhalten längerer Dateien ist das cat-Kommando jedoch ungeeignet. Besser eignen sich hierzu die Kommandos `more` und `less`, wobei `less` am komfortabelsten ist. Probieren Sie einfach aus, welche Kommandos bei Ihnen verfügbar sind:

```
more datei.txt
less datei.txt
```

Bei `more` können Sie seitenweise mit der Leertaste durch den Inhalt blättern, jedoch nur einmal und vorwärts. Bei `less` können Sie mit den Pfeiltasten zeilenweise scrollen und mit den Tasten für ↑ und ↓ seitenweise vor- und zurückblättern. Beide Betrachterprogramme können Sie jederzeit durch Eingabe von `q` wieder beenden.

Archivieren/Dearchivieren und Komprimieren/Dekomprimieren

In der Windows-Welt ist es verbreitet, auch dann komprimierte Dateien (etwa im ZIP-Format) zu erzeugen, wenn es einem gar nicht so sehr aufs Komprimieren von Daten ankommt, sondern einfach darauf, etliche Dateien in einer Archivdatei unterzubringen. Zum Anhängen an eine Mail ist ein solches Archiv beispielsweise wesentlich praktischer als lauter Einzeldateien.

In der Welt von Unix und Linux sind Archivieren und Komprimieren getrennte Vorgänge und ebenso Dearchivieren und Dekomprimieren. Verwöhnten Windows-Anwendern, die ganze Verzeichnisbäume mit Drag and Drop in eine Anwendung wie WinZip packen, kommt das Prozedere unter Unix/Linux verständlicherweise etwas antiquiert vor. Ein Hauptgrund für die Aufgabenteilung von Archivieren und Komprimieren ist nämlich, dass das gzip-Kommando, welches für die Komprimierung zuständig ist, immer nur auf eine einzelne Datei anwendbar ist, also wirklich nur komprimiert, ohne zusätzlich Archive zu erzeugen. Glücklicherweise gibt es außerdem das tar-Kommando, dessen Aufgabe es ist, mehrere Dateien in eine Archivdatei zu packen (ohne sie zu komprimieren). Führt man also zuerst ein tar-Kommando aus, um etwa aus »allen Dateien der Installation« eine Archivdatei zu machen, und anschließend ein gzip-Kommando auf diese Archivdatei, dann hat man ein komprimiertes Archiv mehrerer Dateien, das sich beispielsweise als Download-Datei im Internet eignet. Das Auspacken einer solchen Datei geschieht dann ebenfalls in zwei Schritten. Zunächst wird die Datei dekomprimiert, so dass die unkomprimierte Archivdatei wieder zum Vorschein kommt, und anschließend wird das Archiv ausgepackt.

Beginnen wir mit dem Archivieren. Folgendes Kommando erzeugt aus einzeln angegebenen Dateien eine Archivdatei:

```
tar -cf handbuch.tar handbuch.fm handbuch.mif handbuch.xml
```

Dieses Kommando packt die Dateien *handbuch.fm*, *handbuch.mif* und *handbuch.xml* in eine Archivdatei namens *handbuch.tar*. Die Aufrufoption `-cf` teilt dem tar-Kommando mit, dass ein neues Archiv erzeugt werden soll, und zwar in der angegebenen Datei.

`tar` steht übrigens für *tape archiver* und verrät damit deutlich, aus welcher Welt es stammt.

Um ein ganzes Verzeichnis mitsamt Unterverzeichnisstruktur in eine Archivdatei zu packen, wechseln Sie zunächst am besten in das betreffende Verzeichnis. Das Packen kann dann durch folgendes Kommando erfolgen:

```
tar -cf projekt.tar *
```

Alle Dateien und Verzeichnisse werden in die Archivdatei *projekt.tar* gepackt. Unterverzeichnisstrukturen bleiben erhalten.

Den Inhalt eines erzeugten tar-Archivs können Sie sich selbstverständlich auflisten lassen. Geben Sie dazu ein Kommando wie das folgende ein:

```
tar -tvf projekt.tar
```

Durch die Optionen `-tfv` erhalten Sie ein ausführliches Listing aller enthaltenen Dateien einschließlich Pfadname.

Eine tar-Archivdatei kann je nach Anzahl und Umfang der darin gespeicherten Dateien sehr groß werden. Um Archive etwa für Mailversand oder Download zu optimieren oder um einfach nicht unnötig Plattenspeicher zu verbrauchen, besteht der nächste Schritt meist darin, die erzeugte Archivdatei zu komprimieren. Das Standardkommando dazu ist `gzip`. Ein Beispiel:

```
gzip projekt.tar
```

Nach erfolgreichem Ablauf existiert anschließend keine tar-Datei mehr, dafür aber eine gleichnamige tar.gz-Datei. In unserem Beispiel wird aus *projekt.tar* also *projekt.tar.gz*.

Um den Inhalt der archivierten Datei zu überprüfen, können Sie die Option `-l` benutzen:

```
gzip -l projekt.tar.gz
```

Ausgegeben werden Informationen zur Kompressionsdichte sowie Name und Größe der archivierten Datei.

Die meisten Downloads mit Linux-Software liegen in diesem Format vor, also in einer Datei mit der Endung *tar.gz*. Um etwa nach einem Download an die in einer solchen Datei enthaltenen Daten zu gelangen, müssen Sie die Datei zuerst dekomprimieren. Legen Sie

die Datei dazu am besten in dem Verzeichnis ab, in dem Sie später die dekomprimierten und entpackten Dateien ablegen möchten. Das Dekomprimieren ist einfach:

```
gunzip projekt.tar.gz
```

Daraus wird dann, unserem Beispiel folgend, wieder die Datei *projekt.tar*. Die Datei *projekt.tar.gz* existiert anschließend nicht mehr.

Um an die Originaldaten zu gelangen, muss nun noch die Archivdatei ausgepackt werden:

```
tar -xf projekt.tar
```

Damit werden im aktuellen Verzeichnis alle in *projekt.tar* gespeicherten Inhalte ausgepackt.

Harte und symbolische Links

In manchen Fällen ist es sinnvoll, Dateien an mehreren Stellen im Verzeichnisbaum erreichen zu können, wobei aber nur eine davon die »echte« Datei ist. Die übrigen Erscheinungen sind Links auf die Originaldatei. Links sind selbst natürlich ebenfalls Dateien. Handelt es sich beim Original beispielsweise um eine ausführbare Datei, so bewirkt das Ausführen der Linkdatei das Ausführen der Originaldatei, da die Linkdatei auf die Originaldatei verweist.

Linux unterscheidet jedoch zwischen harten und weichen (»symbolischen«) Links. Harte Links verweisen direkt auf den Inode der Originaldatei. Im Verzeichnisbaum sind sie nicht vom Original zu unterscheiden, bei der Linkdatei werden z.B. die Dateigröße und der Zeitstempel der Originaldatei angezeigt. Anders dagegen symbolische Links: Sie stellen unterscheidbare Dateien dar und bei einer Auflistung mit `ls -l` werden sie deutlich als Links gekennzeichnet.

Harte Links haben den Vorteil, dass sie nicht ins Leere zeigen. Sie werden praktisch mit dem Original, auf das sie zeigen, mitverwaltet. Symbolische Links dagegen nicht. Sie zeigen ins Leere, falls ihr Verweisziel gelöscht oder verschoben wird. Dafür können symbolische Links im Gegensatz zu harten Links auch auf andere Partitionen verweisen.

Beide Arten von Links werden mit dem ln-Kommando gesetzt. Ein harter Link:

```
ln produkte/index.html produkte.html
```

Damit wird ein harter Link unter dem Dateinamen *produkte.html* erzeugt, der auf die Datei *index.html* im Unterverzeichnis *produkte* verweist.

Als symbolischer Link sieht das gleiche Kommando so aus:

```
ln -s produkte/index.html produkte.html
```

Der Unterschied besteht also nur in der Option `-s` (für *symbolic*).

Jede der beiden Dateien kann in beiden Fällen beispielsweise im Browser aufgerufen oder mit einem Editor zum Bearbeiten geöffnet werden. Der Link leitet dabei einfach zum Original weiter.

10.2.6 Kommandos für Zugriffsrechte

Zugriffsrechte auf Dateien sind in einem Linux-System der Schlüssel für die Rechte, die ein eingerichteter Benutzer am System hat.

Benutzertypen und Zugriffsarten

Jede Datei gehört nach dem Erzeugen dem Benutzer, von dem sie erzeugt wird, und der Gruppe, welche die Primärgruppe dieses Benutzers ist. Insgesamt werden für die Zugriffsrechte auf eine Datei (das können nach dem Dateikonzept von Linux also auch Verzeichnisse, Geräte oder laufende Prozesse sein) drei Benutzertypen unterschieden:

1. der Benutzer, dem die Datei gehört,
2. die Primärgruppe des Benutzers, dem die Datei gehört,
3. der »Rest der Welt«, worunter alle Benutzer mit Zugang zum System verstanden werden, die nicht zur Primärgruppe des Dateieigentümers gehören.

Für jeden dieser drei Benutzertypen wird bei jeder Datei festgelegt, was der Benutzertyp mit der Datei tun darf. Unterschieden werden:

1. Lesen erlaubt ja/nein (also z. B. Ansehen oder Kopieren),
2. Schreiben erlaubt ja/nein (also z. B. Bearbeiten, Löschen oder Verschieben),
3. Ausführen erlaubt ja/nein (nur bei Scripts, Programmen usw. von Bedeutung).

Daraus ergeben sich logische Kombinationen wie die folgenden Beispiele:

- Der Benutzer darf die Datei lesen und schreiben, die Mitglieder seiner Primärgruppe dürfen sie lesen und der Rest der Welt darf sie weder lesen noch schreiben.
- Der Benutzer darf die Datei lesen, schreiben und ausführen. Andere Mitglieder seiner Primärgruppe dürfen sie lesen und ausführen. Sonst darf niemand irgendetwas mit der Datei tun.
- Sowohl der Benutzer als auch die Mitglieder seiner Primärgruppe als auch alle übrigen Benutzer dürfen die Datei lesen und schreiben.

Darstellung von Zugriffsrechten

Zugriffsrechte auf Dateien werden an verschiedenen Stellen angezeigt. Ein Beispiel sind ausführliche Verzeichnis-Listings mit `ls -l`.

Wenn Sie Verzeichnisinhalte mit `ls -l` auflisten, werden in der ersten Spalte die Dateirechte und der Typ der Datei angezeigt. Das allererste Zeichen signalisiert dabei den Typ. Wenn es sich um eine normale Datei handelt, wird ein Minuszeichen (`-`) angezeigt. Ist die

Datei ein Verzeichnis, wird `d` angezeigt. Andere Typen sind `b` für Blockgerät (z.B. Festplatte), `c` für zeichenbasiertes Gerät (z.B. Terminal) oder `p` für Pufferdateien, wie sie von Pipe-Kommandos erzeugt werden.

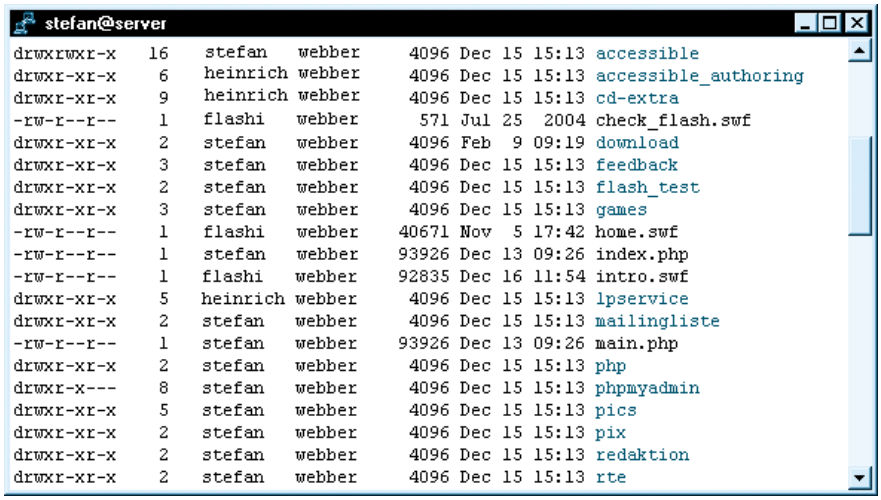


Abbildung 10.5: Ausgabe von ls -l inklusive Anzeige von Dateirechten und Eigentümern

Dahinter folgen im Normalfall neun Zeichen mit Buchstaben wie r, w und x oder Minuszeichen.

Nachfolgende Tabelle listet auf, wie die gesamte Zeichenfolge zu interpretieren ist:

Position	Bedeutung
1	Typ der Datei (`-`, `d`, `b`, `c`, `p`).
2	Zeigt an, ob der Eigentümer die Datei lesen darf (`r`) oder nicht (`-`).
3	Zeigt an, ob der Eigentümer die Datei schreiben darf (`s`) oder nicht (`-`).
4	Zeigt an, ob der Eigentümer die Datei ausführen darf (`x`) oder nicht (`-`).
5	Zeigt an, ob Mitglieder der Primärgruppe des Eigentümers die Datei lesen dürfen (`r`) oder nicht (`-`).
6	Zeigt an, ob Mitglieder der Primärgruppe des Eigentümers die Datei schreiben dürfen (`s`) oder nicht (`-`).
7	Zeigt an, ob Mitglieder der Primärgruppe des Eigentümers die Datei ausführen dürfen (`x`) oder nicht (`-`).
8	Zeigt an, ob sonstige Benutzer die Datei lesen dürfen (`r`) oder nicht (`-`).
9	Zeigt an, ob sonstige Benutzer die Datei schreiben dürfen (`s`) oder nicht (`-`).
10	Zeigt an, ob sonstige Benutzer die Datei ausführen dürfen (`x`) oder nicht (`-`).

Tabelle 10.4: Rechte-String in Linux (rwxrwxrwx)

Eine Zeichenkette wie `-rw-r--r--` bedeutet also: Es handelt sich um eine »normale« Datei (`-` am Anfang). Der Eigentümer der Datei darf die Datei lesen (`r`) und schreiben (`w`). Mitglieder seiner Primärgruppe dürfen sie nur lesen und der Rest der Welt darf sie ebenfalls lesen.

Eine Zeichenkette wie `drwxrwxr-x` bedeutet: Es handelt sich um ein Verzeichnis (`d`). Der Eigentümer darf das Verzeichnis, also Dateien darin, lesen, schreiben und ausführen. Auch die Mitglieder seiner Primärgruppe dürfen Dateien darin lesen, schreiben und ausführen. Der Rest der Welt darf lesen und ausführen, aber nicht schreiben. Beachten Sie, dass etwa eine Scriptdatei innerhalb eines Verzeichnisses nur dann ausführbar ist, wenn sowohl die Scriptdatei selbst als auch das Verzeichnis, in dem sie sich befindet, auf ausführbar gesetzt ist.

Neben der Schreibweise als Rechte-String gibt es auch eine numerische Schreibweise mit Hilfe von Oktalzahlen, also Zahlen zwischen 0 und 7. Nachfolgende Tabelle listet die Zusammenhänge zwischen Rechte-String und Oktalschreibweise auf:

Stringschreibweise	Oktalschreibweise
`rwx`	7
`rw-`	6
`r-x`	5
`r--`	4
`-wx`	3
`-w-`	2
`--x`	1
`---`	0

Tabelle 10.5: Zuordnung zwischen Stringschreibweise und Oktalschreibweise

Ein Rechtestring wie `rw-rw-r--` lautet in Oktalschreibweise demnach `664` und ein Rechtestring wie `rwxrwxr-x` lautet `775`.

Während Kommandos wie `ls -l` die Stringschreibweise der Rechte ausgeben, wird bei anderen Kommandos eher die Oktalschreibweise verwendet. Auch in vielen anderen Zusammenhängen, z. B. bei den Voreinstellungen in Programmen wie WinSCP, kommen Sie mal mit Rechte-Strings, mal mit der Oktalschreibweise in Berührung. Das »Umrechnen« zwischen beiden Schreibweisen sollten Sie daher beherrschen.

Zugriffsrechte ändern

Das Kommando `ls -l` listet in Spalte 3 auf, welcher Benutzer der Eigentümer der Datei ist, und in Spalte 4, welches die besitzende Gruppe ist. Per Default ist dies die Primärgruppe des Eigentümers.

Um die Rechteverhältnisse einer Datei zu ändern, gibt es zwei Möglichkeiten: Entweder man ändert nur die Zugriffsrechte für die drei Benutzertypen (Eigentümer, Gruppe, Rest

der Welt) und lässt die Besitzverhältnisse wie sie sind; oder man ändert den Eigentümer und/oder die besitzende Gruppe und lässt die Zugriffsrechte wie sie sind.

Mit dem chmod-Kommando (`chmod` = ***change mode***) können Sie die Zugriffsrechte ändern. In unseren Beispielen verwenden wir die oktale Schreibweise:

```
chmod 664 index.html
```

Damit werden für den Eigentümer und seine Gruppe Schreib- und Leserecht gewährt und für den Rest der Welt nur Leserecht.

Es können auch mehrere Dateien angegeben werden oder mit Wildcards ausgewählt werden. Sogar ganze Unterverzeichnisstrukturen lassen sich rekursiv ändern. Beispiele:

```
chmod 664 index.html produkte.html impressum.html
chmod 664 *
chmod 664 -R *
```

Im ersten Fall werden die Zugriffsrechte der drei angegebenen Dateien geändert. Im zweiten Fall werden alle Dateien im aktuellen Verzeichnis geändert und im dritten Fall zusätzlich alle Dateien in Unterverzeichnissen und deren Unterverzeichnissen usw.

Um den Eigentümer und wahlweise auch die Eigentümergruppe einer Datei zu ändern, steht das chown-Kommando (`chown` = ***change owner***) zur Verfügung:

```
chown andreas demo.swf
```

Damit wird ***andreas*** zum Eigentümer der Datei ***demo.swf***.

In der Praxis ist es oft sinnvoll, sowohl Eigentümer als auch Eigentümergruppe zu wechseln:

```
chown andreas:webworkers demo.swf
```

Durch die Syntax `Benutzername:Gruppenname` können Sie beides auf einmal zuweisen.

Um nur die Eigentümergruppe zu ändern, können Sie das chgrp-Kommando verwenden (`chgrp` = ***change group***):

```
chgrp webworkers screensaver.zip
```

In diesem Beispiel wird der Datei ***screensaver.zip*** die Eigentümergruppe ***webworkers*** zugewiesen.

Die beiden Kommandos `chown` und `chgrp` erlauben ebenso wie `chmod` die Angabe mehrerer Dateien, einer Dateiauswahl durch Wildcards und die Option `-R` für rekursive Abarbeitung ganzer Unterverzeichnisbäume.

Das Ändern von Zugriffsrechten oder Eigentumsverhältnissen mithilfe der drei beschriebenen Kommandos ist allerdings nur möglich, wenn Sie selbst über hinreichende Rechte verfügen. Das Kommando `chown` ausführen darf in der Regel nur, wer root-Rechte besitzt,

in den meisten Fällen also nur der Benutzer *root*. Beim Ändern der Eigentümergruppe mit `chgrp` muss der Benutzer, der das Kommando ausführt, selbst Mitglied der neu zugewiesenen Gruppe sein. Bei `chmod` müssen Sie entweder root-Rechte besitzen oder der Eigentümer der Datei sein.

Voreinstellungen bei Dateirechten

Beim Erzeugen neuer Dateien werden Default-Werte für Zugriffsrechte vergeben. Verbreitet sind beispielsweise `644` (`rw-r--r--`) oder `660` (`rw-rw----`). Welche Rechte automatisch vergeben werden, wird beim Systemstart durch Ausführen eines Kommandos namens `umask` bestimmt. Sie können `umask` jedoch auch selbst aufrufen, um für neu zu erzeugende Dateien gewünschte Default-Rechte zu vergeben.

Das umask-Kommando arbeitet mit Bitoperatoren. Nachfolgende Tabelle listet auf, welche sinnvollen umask-Definitionen welche Wirkung haben:

umask-Kommando	Wirkung
`umask 022`	Neue Dateien erhalten die Rechte `755`.
`umask 133`	Neue Dateien erhalten die Rechte `644`.
`umask 117`	Neue Dateien erhalten die Rechte `660`.
`umask 006`	Neue Dateien erhalten die Rechte `771`.

Tabelle 10.6: umask und Default-Rechte für neue Dateien

Die Oktalschreibweise zeigt, dass der Wert von `umask`, ziffernweise abgezogen von den Maximalrechten `777`, die gewünschte Rechtekombination ergibt.

10.2.7 Kommandos zur Systemüberwachung

Wenn Sie als Mieter eines Root-Servers Performanceeinbußen bei Ihren dort gehosteten Webseiten feststellen oder gemeldet bekommen, die zunächst nicht oder nicht immer existieren, können Probleme auf Betriebssystemebene eine mögliche Ursache sein. Möglicherweise ist der Arbeitsspeicher zu knapp, die Festplatten könnten fast voll belegt sein oder die CPU ist zu langsam. Aber auch dafür kann es verschiedene Ursachen geben. Der Arbeitsspeicher kann beispielsweise durch so genannte Zombie-Prozesse belastet sein.

Ohne ein Systemspezialist sein zu müssen, können Sie mithilfe einiger Kommandos durchaus Hinweise auf Probleme im System finden, die Sie beispielsweise dem Provider mitteilen und in manchen Fällen sogar selbst beseitigen können.

Plattenspeicher kontrollieren

Das klassische Kommando hierzu ist das df-Kommando (`df` = *disk free*).

Das df-Kommando listet für alle in den Verzeichnisbaum eingehängten physikalischen Datenträger auf, wie viel Speicherplatz darauf belegt bzw. noch frei ist. Sinnvoll sind folgende Aufrufoptionen:

```
df -h
```

Damit werden die Angaben zu belegtem und freiem Speicher in Mbyte (M) oder Gbyte (G) angegeben. Fehlt die Option, werden stattdessen die intern benutzten Blockzahlen aufgelistet.

```
df -i
```

Damit werden die Anzahl vorhandener und noch freier Inode-Knoten ausgegeben. Dies entspricht der Anzahl vorhandener bzw. noch möglicher Dateien im System.

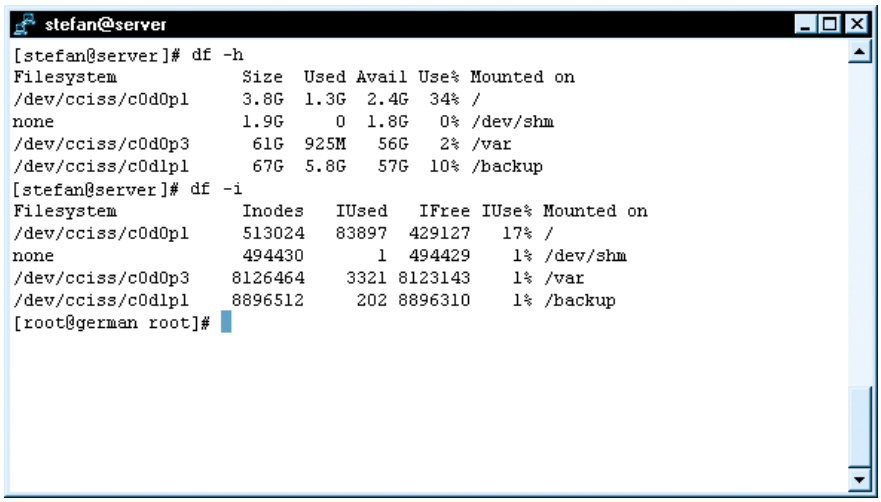


Abbildung 10.6: Typische Ausgaben bei df -h und df -i

Die erste Spalte gibt die Gerätedatei des Datenträgers mit Pfadnamen an. Die zweite, dritte und vierte Spalte gibt Auskunft über belegten und freien Speicher, abhängig von der gewählten Aufrufoption. Die vierte Spalte teilt die Belegung in Prozent mit, ebenfalls abhängig von der Aufrufoption. In der letzten Spalte steht, unter welchem Verzeichnispfad im Dateisystem die Daten des jeweiligen Datenträgers zu finden sind. Das df-Kommando eignet sich also auch, um Informationen über Datenträger und deren Zuordnung zum Dateisystem zu ermitteln.

Arbeitsspeicher kontrollieren

Das Kommando, um sich über die aktuelle Arbeitsspeicherbelegung zu informieren, ist das free-Kommando:

`free -m` listet die Werte in Mbyte auf.

`free -k` listet die Werte in Kbyte auf.

`free -b` listet die Werte in Byte auf.

Nun entsteht gerade beim Arbeitsspeicher häufig der Wunsch, nicht nur ein einmaliges Ergebnis zu erhalten, sondern die Ergebnisse fortlaufend aktualisiert zu bekommen, um

die Auslastung über einen gewissen Zeitraum am Bildschirm mitverfolgen zu können. Hierzu bietet das free-Kommando die Option `-s` an. Beispiel:

```
free -s 2
```

Damit wird das Ergebnis alle zwei Sekunden neu ermittelt und ausgegeben. Die Ausgaben erfolgen untereinander, d.h., die bisherigen Ausgaben bleiben erhalten und können so zu Vergleichen herangezogen werden. Das Kommando befindet sich nun in einer Endlosschleife. Durch das Halten der gedrückten Strg-Taste und Drücken von C können Sie es beenden.

Laufende Prozesse und Systemauslastung kontrollieren

Um sich über aktuelle Prozesse und die allgemeine Systemauslastung zu informieren, verwenden Sie am besten das top-Kommando.

```
top
```

Damit starten Sie den interaktiven Modus des Kommandos. Die Anzeige wird automatisch regelmäßig aktualisiert. Um die Anzeige zu beenden, drücken Sie einfach Q.

Um die Aktualisierungsrate zu beeinflussen, können Sie `top` wie folgt aufrufen:

```
top -d1
```

Damit wird die Anzeige jede Sekunde aktualisiert. Hinter dem Schalter `d` geben Sie die gewünschte Anzahl Sekunden bis zur nächsten Aktualisierung an.

Wenn Sie nur eine einmalige Ausgabe der Ergebnisse wünschen und gleich wieder zum Shell-Prompt zurückkehren möchten, können Sie Folgendes eingeben:

```
top -bn1
```

Damit wird angegeben, dass das Ergebnis nur einmal angezeigt werden soll und dass das Kommando nicht im interaktiven Modus laufen soll. Diese Art des Aufrufs eignet sich beispielsweise auch gut, um die Ausgabe in einer Datei zu speichern:

```
top -bn1 > top.txt
```

Damit wird eine einmalige Erzeugung des Ergebnisses in einer Datei *top.txt* gespeichert.

Das top-Kommando zeigt eine Menge wichtiger Daten zur Systemüberwachung an.

Im Kopfbereich werden diverse Informationen wie die seit dem letzten Systemstart vergangene Zeit, die Anzahl aktueller Benutzer und – ganz wichtig – der so genannte *load average* angezeigt. Hinter *load average* erscheinen drei Zahlen: Diese geben die Anzahl zur Verarbeitung wartender Prozesse in der letzten Minute, den letzten 5 Minuten und den letzten 15 Minuten an (mit zwei Nachkommastellen). Je höher die Werte, desto stärker ist das System ausgelastet. Wenn der load-Wert etwa bei einem als Webserver eingesetzten

Linux-Rechner über einen längeren Zeitraum deutlich mehr als etwa 3 oder 4 beträgt, dann ist das System überlastet, d.h., die CPU kann die Menge an anstehenden Arbeiten kaum erledigen. Das ist ein Zeichen dafür, dass der Rechner zu schwach ist und mit einer schnelleren CPU oder mit mehreren CPUs ausgestattet werden sollte.

Des Weiteren werden im Kopfbereich die Anzahl gestarteter Prozesse ausgegeben, wobei auch zwischen solchen unterschieden wird, die seit der letzten Ergebnisauffrischung die CPU beschäftigen (*running*) und solchen, die geladen sind, aber seit der letzten Ergebnisauffrischung keine CPU-Zeit belegt haben (*sleeping*). Wichtig ist auch der Wert für *zombies*.

Zombies können durch das unter Unix/Linux übliche Prozess-Forking entstehen. Wenn zahlreiche Zombies angezeigt werden, ist das ein Hinweis darauf, dass ein Programm oder Script, welches Prozess-Forking verwendet, nicht ordentlich funktioniert oder z.B. mehrfach gewaltsam abgebrochen wurde. Auch ein Hacker, der ins System eingedrungen ist, oder Angriffe von außen in Form von Scripts, die den Rechner mit Anfragen belasten, können durch bewusste Erzeugung von immer mehr Zombies dafür sorgen, dass der Rechner langsam, aber sicher in die Knie geht.

Der CPU-Status ist eine wichtige ergänzende Information. Wird die CPU ständig zu 100% von *user* oder *system* ausgelastet, dann ist sie wahrscheinlich überlastet, was sich in den *load*-Werten bemerkbar machen sollte.

Auch die Belegung des physikalischen und virtuellen Arbeitsspeichers wird mit ausgegeben und aktualisiert. Dabei sollten Sie sich nicht davon verwirren lassen, wenn ständig fast der gesamte verfügbare Arbeitsspeicher (*av*) belegt (*used*) ist. Das System reserviert sich bis auf einen kleinen Rest den verfügbaren Arbeitsspeicher und verwaltet dann darin seine Prozesse und Daten.

In einer Liste, die so lang ist, wie das Anzeigefenster hoch ist, werden aktuelle Prozesse in tabellarischer Form angezeigt. Jede Zeile stellt einen Prozess dar. Die einzelnen Spalten haben folgende Bedeutung:

Spaltenüberschrift	Bedeutung
PID	Eindeutige interne Identifikationsnummer des Prozesses.
USER	Benutzername des Benutzers, der den Prozess gestartet hat.
PRI	Priorität des Prozesses. Der Wert schwankt üblicherweise zwischen -19 (niedrigste Priorität) und 19 (höchste Priorität). Der Wert 0 bedeutet mittlere Priorität.
NI	Nice-Wert. Damit lässt sich die Priorität beeinflussen. Werte kleiner als 0 erhöhen die Priorität, Werte größer 0 verringern sie.
SIZE	Größe, die der Prozess auf dem Stack belegt.
RSS	Größe, die der Prozess im Arbeitsspeicher belegt.
SHARE	Größe von gemeinsam mit anderen Prozessen genutzten Speicherbereichen, die der Prozess benutzt.

Tabelle 10.7: Spalten der tabellarischen Anzeige für Prozesse im top-Kommando

Spaltenüberschrift	Bedeutung
STAT	Status des Prozesses: *S* bedeutet schlafend, *W* wartend, *R* laufend, *T* beendet und *Z* bezeichnet einen Zombie-Prozess.
%CPU	Prozentwert, wie viel von der CPU-Rechenzeit der Prozess seit der letzten Ergebnisauffrischung verbraucht hat.
%MEM	Prozentwert, wie viel vom verfügbaren Arbeitsspeicher der Prozess seit der letzten Ergebnisauffrischung belegt hat.
TIME	CPU-Zeit, die der Prozess seit seinem Start verbraucht hat.
COMMAND	Programm oder Kommando, welches dem Prozess an der Oberfläche entspricht.

Tabelle 10.7: Spalten der tabellarischen Anzeige für Prozesse im top-Kommando (Fortsetzung)

Die letzte Spalte gibt Auskunft darüber, welches Programm oder Kommando eigentlich hinter dem jeweiligen Prozess steckt. Beim Betrieb eines typischen Apache Webservers mit Webanwendungen, die MySQL-Datenbankanbindung nutzen, tauchen bei laufendem Betrieb zahlreiche Prozesse mit Kommandonamen *httpd* und *mysqld* auf. An der Anzahl der httpd-Prozesse ist erkennbar, wie viele offene Verbindungen der Apache Webserver zu aufrufenden Clients aktuell unterhält. An den mysqld-Aufrufen können Sie erkennen, wie viele Datenbankverbindungen derzeit bestehen.

Das top-Kommando kann allerdings noch mehr, als nur Daten über aktuelle Prozesse anzuzeigen. Wenn Sie im interaktiven Modus h drücken, erscheint eine kommentierte Liste weiterer Buchstaben, die Sie im interaktiven Modus drücken können. Beenden können Sie die Hilfeanzeige mit einer beliebigen Taste. Wenn bei einem solchen Kommando ein anschließender Dialog erforderlich ist, wie z.B. beim Killen eines Prozesses, dann erscheint die entsprechende Eingabeaufforderung in der Zeile oberhalb der Spaltenüberschriften der Prozesstabelle.

Prozesse killen

Zombie-Prozesse oder abgestürzte Prozesse lassen sich gewaltsam beenden. Als normaler Benutzer können Sie jedoch nur Prozesse killen, die von Ihnen selbst gestartet wurden. Als Benutzer *root* können Sie dagegen Prozesse aller Benutzer killen.

Zum Killen eines Prozesses benötigen Sie dessen Prozessnummer (PID). Das top-Kommando listet auch die PIDs auf. Ein einfacheres Kommando lautet:

```
ps -Af
```

Damit listen Sie laufende Prozesse aller Benutzer auf und lassen sich alle Details mit ausgeben. Um nur selbst gestartete Prozesse anzuzeigen, können Sie verwenden:

```
ps -f
```

In der zweiten Spalte wird die PID eines Prozesses angezeigt. Um einen Prozess mit der PID 16901 zu killen, geben Sie ein:

```
kill 16901
```

Datum und Uhrzeit kontrollieren

Viele professionelle Webanwendungen speichern Zeitstempel, z. B. um Anwendern einen zeitlich begrenzten und von ihnen bezahlten Zugang zu bestimmten Inhalten zu gewähren. Grundlage aller Zeitstempelfunktionen in Scriptsprachen ist jedoch die Rechnerzeit. Nach Rechnerausfällen empfiehlt es sich daher stets, die Einstellungen zu Datum und Uhrzeit zu kontrollieren und gegebenenfalls zu aktualisieren.

```
date
```

Damit geben Sie Datum und Uhrzeit nach der Uhr des Rechners aus.

Um den Wert zu ändern, gibt es zahlreiche Möglichkeiten. Hier eine davon:

```
date -s='Wed, 22 Jun 2005 16:09:00 +0100'
```

Damit wird die Rechneruhr auf Mittwoch, den 22. Juni 2005, 16:09 Uhr und 0 Sekunden, bei 1 Stunde Zeitverschiebung plus gegenüber Greenwich Meantime oder UTC gesetzt.

System neu starten (Reboot)

Einen Server-Rechner mit Linux-System sollten Sie niemals ohne besonderen Grund neu starten. Sollte jedoch mal ein Fall eintreten, der einen Neustart rechtfertigt, dann können Sie Folgendes eingeben:

```
shutdown -r now
```

Damit wird der Rechner ordentlich heruntergefahren und anschließend wieder hochgefahren. Wenn Sie dieses Kommando jedoch aus der Ferne via SSH-Zugang starten, sollten Sie vorher überprüfen, ob beim Hochfahren auf jeden Fall der SSH-Server neu gestartet wird. Andernfalls können Sie sich anschließend nicht mehr anmelden.

10.2.8 Kommandos für Softwareverwaltung

In der Regel ist es nötig, mit der root-Kennung zu arbeiten, wenn neue Software installiert werden soll. Wechseln Sie also mit `su root` zum Super-User *root*, bevor Sie eine heruntergeladene Software auf einem Linux-System installieren möchten. Es gibt zwei typische Arten, wie Linux-Software im Internet zum Download angeboten wird: entweder als so genanntes RPM-Paket oder in Form der bereits behandelten tar.gz-Archive. Daneben werden immer häufiger auch bz2- und tgz-Dateien zum Download angeboten.

RPM steht für *Red Hat Paket Manager*. Das Konzept wurde von der Red Hat Distribution eingeführt. RPM-Pakete haben den Vorteil, dass sie bereits vorkompiliert sind. Da vorkompilierte Software jedoch nur unter bestimmten Systemvoraussetzungen lauffähig ist, werden solche Pakete nur für bestimmte Linux-Distributionen angeboten. Achten Sie bei Downloads genau darauf, ob bei einem RPM-Paket auch die auf Ihrem Rechner laufende Linux-Distribution genannt wird.

tar.gz-Archive, bz2-Archive und tgz-Archive enthalten dagegen unkompilierte, im Quellcode vorliegende Software. Mithilfe eines mitgelieferten Kompilierungsscripts und des auf dem Rechner hoffentlich installierten C-Compilers ist es dann möglich, die Software zu kompilieren und zu installieren.

Beispiel einer Download-Entscheidung

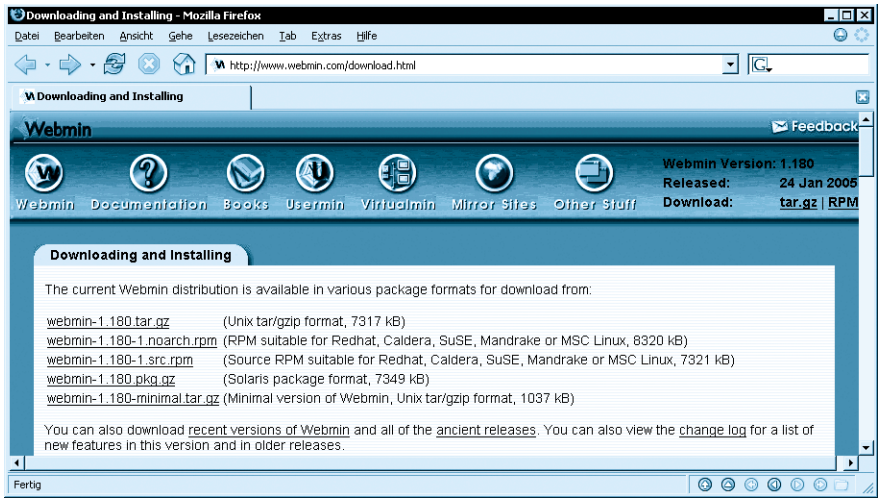


Abbildung 10.7: Typische Download-Site eines Unix/Linux-Software-Produkts (hier: Webmin)

Angenommen, Sie möchten das abgebildete Webmin auf Ihrem Rechner installieren (eine Software, die eine webbasierte Verwaltungsoberfläche für ein Unix/Linux-System realisiert).

Wenn auf Ihrem Rechner eine Linux-Distribution von Redhat, Caldera, SUSE, Mandrake oder MSC läuft, dann können Sie beispielsweise die zweite angebotene Download-Datei (*webmin-1.180-1.noarch.rpm*) anklicken.

Wenn aber eine andere Linux-Distribution läuft, z.B. Gentoo Linux, dann können Sie die rpm-Datei im abgebildeten Beispiel nicht herunterladen, weil Gentoo nicht bei den Distributionen genannt ist, unter denen das Kompilat lauffähig ist. In diesem Fall laden Sie die allgemeine tar.gz-Datei (*webmin-1.180.tar.gz*) herunter.

RPM-Pakete installieren

1. Legen Sie die RPM-Datei in einem temporären Verzeichnis ab, z.B. unter */tmp/install*.
2. Wenn Sie das Programm zum ersten Mal installieren, geben Sie ein:

```
rpm -ivh dateiname.rpm
```

Wenn das Programm bereits installiert ist und Sie eine neuere Version installieren möchten, geben Sie ein:

```
rpm -U dateiname.rpm
```

Archive mit Quellcode-Software installieren

1. Legen Sie die RPM-Datei in einem temporären Verzeichnis ab, z.B. unter */tmp/install*.
2. Dekomprimieren Sie die Download-Datei.

   ```
   gzip -d dateiname.tar.gz
   ```

   Wenn die Download-Datei auf *tgz* endet, geben Sie ein:

   ```
   gunzip dateiname.tgz
   ```

   Wenn die Download-Datei auf *bz2* endet, geben Sie ein:

   ```
   bzip2 -d dateiname.tar.bz2
   ```

3. Geben Sie nun ein:

   ```
   tar xvf dateiname.tar
   ```

   In den meisten Fällen wird der Inhalt des tar-Archivs in ein Unterverzeichnis entpackt.
4. Lassen Sie sich mit `ls -l` den Inhalt des Download-Verzeichnisses (z.B. */tmp/install*) auflisten und prüfen Sie, ob der Archivinhalt in ein Unterverzeichnis entpackt wurde. Wenn ja, wechseln Sie mit `cd verzeichnisname` in das entsprechende Unterverzeichnis.
5. Geben Sie ein:

   ```
   less README
   ```

   oder:

   ```
   more README
   ```

   Sollte keine Datei namens README existieren, dann lassen Sie sich mit `ls -l` den Verzeichnisinhalt auflisten und suchen Sie nach einer vergleichbaren Datei, z.B. *readme.txt*.

   Lesen Sie in der readme-Datei vor allem nach, welche Aufrufmöglichkeiten das Script namens *configure* kennt und welche Optionen sich dadurch einstellen lassen. Überlegen Sie, welche der möglichen Optionen Sie setzen möchten.
6. Geben Sie ein:

   ```
   ./configure [ggfs. mit Optionen]
   ```

In den meisten Installationen sollte ein Script mit diesem Namen existieren. Es handelt sich um ein Shellscript, das Installationsoptionen wie das Festlegen des endgültigen Programmverzeichnisses erlaubt.

7. Geben Sie, nachdem Sie alle gewünschten Konfigurationsoptionen festgelegt haben, ein:

   ```
   make
   ```

   Damit wird das im Quellcode vorliegende Programm kompiliert. Das make-Kommando sucht selbständig nach einem geeigneten Compiler. Falls kein Compiler gefunden wird, oder falls während des Kompilierens Probleme auftreten, kann es während des Vorgangs zur Ausgabe von Fehlermeldungen kommen. Nicht jede Fehlermeldung oder Warnung ist ein Grund zur Panik. Manchmal können Hunderte von Warnungen ausgegeben werden, die aber keine negativen Auswirkungen haben. Fatale Probleme führen jedoch zum Abbruch.

   Die make-Prozedur kann je nach Umfang der Software einige Zeit in Anspruch nehmen.

8. Geben Sie, wenn das make-Kommando vollständig durchgelaufen ist, ein:

   ```
   make install
   ```

   Damit wird die zuvor kompilierte Software in ihr endgültiges Verzeichnis kopiert.

Beachten Sie, dass die Installation manchmal auch von diesem üblichen Schema abweichen kann. Einige Archive enthalten auch andere Installationsscripts, z.B. *setup.sh* oder *install.sh*.

An dem ganzen umständlichen Prozedere, das aus EDV-Urzeiten stammt und für Anwender, die aus der Windows-Welt kommen, nicht gerade sehr entgegenkommend wirkt, können Sie schon sehen, dass es sich lohnt, nach RPM-Versionen von Software Ausschau zu halten oder nach anderen vorkompilierten Installationsversionen.

10.3 Dateibearbeitung mit dem vi-Editor

Solange Sie an den Quelltexten einer Website arbeiten, sollten Sie dies in einer lokalen Entwicklungsumgebung tun. Fertige Dateien können Sie dann auf den Server-Rechner hochladen. Es gibt jedoch auch genügend Fälle, in denen es sinnvoller ist, eine Datei direkt auf dem Server-Rechner zu bearbeiten. Das betrifft beispielsweise Bearbeitungen der *httpd.conf*, also der zentralen Konfigurationsdatei des Apache Webservers, oder auch die verzeichnis-spezifischen *.htaccess*-Dateien. Auch Konfigurationsdateien des Systems müssen hin und wieder editiert werden.

Wenn Sie aus der Ferne am Rechner arbeiten, also etwa mit SSH-Zugang, können Sie nur Editoren verwenden, die sich auf Shell-Ebene starten lassen und im Textmodus arbeiten. In der Linux-Welt gibt es zahlreiche Editoren, die das leisten. Es existieren auch moder-

nere Editoren als den hier vorgestellten vi-Editor. Es gibt aber keinen anderen Editor, der so verbreitet und universell auf allen Systemen verfügbar ist wie vi. Die Mühe, sich in die für Windows-Anwender sehr verquere Bearbeitungs- und Kommandophilosophie dieses Editors einzuarbeiten, zahlt sich am Ende dadurch aus, dass Sie auf praktisch allen Unix-/Linux-Systemen in der Lage sind, ohne weitere Kenntnisse Textdateien zu bearbeiten.

Auf neueren Systemen ist übrigens meistens bereits die weiterentwickelte Variante des vi-Editors installiert: *vim*. Dennoch können Sie diesen Editor ebenfalls mit dem klassischen vi-Kommando aufrufen.

10.3.1 Starten und Beenden von vi

Sie können vi am Shell-Prompt ohne Datei starten:

```
vi
```

In diesem Fall gibt der Editor seine »Credits« aus und informiert über Hilfemöglichkeiten.

Im Normalfall wird vi jedoch stets mit einer Datei aufgerufen. Existiert die angegebene Datei, wird sie geladen und angezeigt. Existiert sie noch nicht, wird ein leerer Bearbeitungsbereich angezeigt. Wenn Sie Text eingeben und speichern, wird er unter dem Dateinamen gespeichert, den Sie beim Aufruf angegeben haben. Beispiele:

```
vi hinweise.txt
vi /usr/bin/apache-2.0.53/conf/httpd.conf
```

Sie können also Dateien im aktuellen Verzeichnis übergeben, aber auch Pfadnamen zu Dateien in anderen Verzeichnissen.

Darüber hinaus dürfen Sie sich, wenn Sie möchten, noch ein paar Extras merken:

```
vi + hinweise.txt
```

Durch diesen Aufruf öffnen Sie die Datei *hinweise.txt* im vi-Editor. Der Cursor wird jedoch nicht wie üblich am Beginn positioniert, sondern steht gleich in der letzten Textzeile der Datei.

```
vi +28 hinweise.txt
```

Damit wird der Cursor nach dem Laden der Datei sofort in Zeile 28 positioniert. Diese Form des Aufrufs ist sehr hilfreich, wenn Sie z.B. ein Script entwickeln und bei dessen Ausführung eine Fehlermeldung des Script-Interpreters erhalten. Solche Fehlermeldungen enthalten in aller Regel die Zeilennummer, in der ein Syntaxfehler festgestellt wurde.

```
vi +/Stefan hinweise.txt
```

Damit wird der Cursor nach dem Laden der Datei in der Zeile positioniert, in der zum ersten Mal die Zeichenkette `Stefan` vorkommt.

Sie können vi auch mit mehreren Dateien starten. Beispiel:

```
vi hinweise.txt /usr/bin/apache-2.0.53/conf/httpd.conf
```

Angezeigt wird dann zunächst nur die erste angegebene Datei.

Der vi-Editor präsentiert sich in einer denkbar spartanischen Bedienoberfläche. Es sind keine Menüs oder Ähnliches zu sehen, nur nackter Text. Beim Eingeben von Text passiert auch nichts. Verwöhnte Windows-Anwender geraten an dieser Stelle meist in Panik. Ohne Vorkenntnisse ist man in der Tat nicht einmal in der Lage, den Editor wieder ordentlich zu beenden, geschweige denn, sonst etwas Sinnvolles damit zu tun.

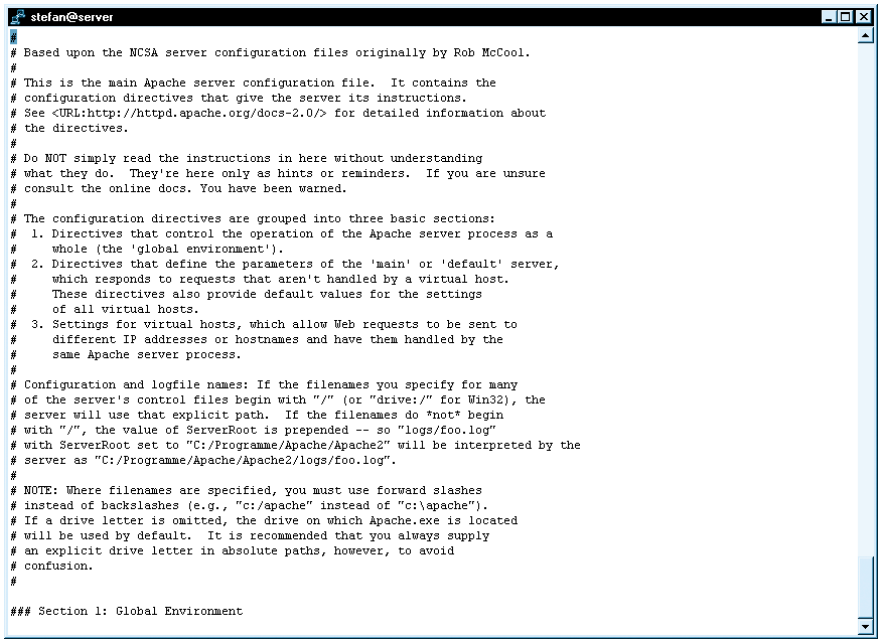


Abbildung 10.8: Nackter Text, keine Bedienoberfläche: die httpd.conf im vi-Editor

Alle Interaktionen zum Bedienen des Editors finden im so genannten Kommandomodus statt, den wir im nächsten Abschnitt behandeln. Dort finden Sie auch die vi-Kommandos, um den Editor zu beenden und zum Shell-Prompt zurückzukehren.

10.3.2 Kommandomodus, Kommandozeile und Eingabemodus

Nach dem Öffnen oder Anlegen einer Datei mit dem vi-Editor befinden Sie sich zunächst im Kommandomodus des Editors. Deshalb können Sie auch noch keinen Text eingeben, sondern nur Kommandos. Dazu müssen Sie allerdings die Zeichenfolge von möglichen Kommandos kennen. Probieren wir ein erstes Kommando aus. Geben Sie folgende Zeichenfolge ein:

```
:help
```

Notieren Sie ein führendes Doppelpunktzeichen und dahinter das Wort *help*. Sobald Sie den Doppelpunkt eingeben, können Sie sehen, dass der Cursor nun in der untersten Zeile steht. Das liegt daran, dass der Doppelpunkt eines der Zeichen ist, die vi als Start eines Kommandos erkennt.

Die unterste Zeile ist die Kommandozeile des vi-Editors.

Drücken Sie, nachdem Sie in der Kommandozeile `:help` eingegeben haben, die Enter-Taste. Am Bildschirm wird nun Hilfe zum Editor angezeigt. Beenden Sie die Hilfeanzeige durch folgendes Kommando:

```
:exit
```

Drücken Sie am Ende der Kommandoeingabe stets die Enter-Taste, um das Kommando auszuführen. Durch das Kommando `:exit` kehren Sie wieder zur bearbeiteten Datei zurück. Sie befinden sich weiterhin im Kommandomodus.

Um vom Kommandomodus in den Eingabemodus zu wechseln, also um Text bearbeiten zu können, müssen Sie ein entsprechendes Kommando eingeben, ebenso wie Sie ein Kommando eingeben müssen, um den Editor zu beenden oder um Text zu suchen oder um andere Aktionen durchzuführen.

vi-Kommandos zum Speichern und Beenden des vi-Editors

Kommando	Wirkung
`:w`	Speichert die angezeigte Datei, ohne zu beenden. Geeignet zum Zwischenspeichern.
`:wq`	Speichert die angezeigte Datei und beendet den Editor.
`:q`	Beendet den Editor, jedoch nur, wenn nichts geändert wurde oder wenn zuvor bereits mit `:w` gespeichert wurde.
`:q!`	Beendet den Editor, ohne zu speichern. Änderungen seit dem Öffnen oder letzten Zwischenspeichern gehen verloren.
`ZZ`	(ohne Doppelpunkt): gleiche Wirkung wie `:wq`.

Tabelle 10.8: vi-Kommandos zum Speichern und Beenden des vi-Editors

vi-Kommandos zum Bewegen im Text

Probieren Sie, wenn Sie vi an einer PC-Tastatur bedienen, zum Bewegen im Text die üblichen Pfeiltasten und die Tasten für ↑ und ↓. Sollte das nicht funktionieren, können Sie die nachfolgenden Buchstaben drücken.

Kommando	Wirkung
`l` (kleines L)	Cursor ein Zeichen weiter nach rechts.
`h`	Cursor ein Zeichen weiter nach links.

Tabelle 10.9: vi-Kommandos zum Bewegen im Text

Kommando	Wirkung
k	Cursor eine Zeile weiter nach oben.
j	Cursor eine Zeile weiter nach unten.
w	Cursor ein Wort nach rechts.
b	Cursor ein Wort nach links.
G	Cursor an den Anfang der letzten Zeile.
Strg+f	Wie »Bild ab«.
Strg+b	Wie »Bild auf«.
^	Cursor an den Anfang der aktuellen Zeile.
$	Cursor ans Ende der aktuellen Zeile.

Tabelle 10.9: vi-Kommandos zum Bewegen im Text (Fortsetzung)

vi-Kommandos zum Wechseln in den Eingabemodus

Bewegen Sie den Cursor gegebenenfalls zunächst an die Stelle im Text, an der Sie etwas einfügen möchten.

Nachdem Sie eines der nachfolgenden Kommandos eingegeben und mit Enter bestätigt haben, können Sie Text entsprechend der Charakteristik des Kommandos eingeben. Um wieder in den Kommandomodus zurückzukehren, drücken Sie die Esc-Taste.

Kommando	Wirkung
i	Text vor dem Cursor einfügen. Dies ist der normale Einfügemodus für neuen Text. In diesem Modus können Sie auch mit der Enter-Taste neue Zeilen beginnen. Innerhalb der aktuellen Zeile können Sie mit der ←-Taste entweder zeichenweise zurückgehen, ohne zu löschen, oder zeichenweise rückwärts löschen. Das freie Bewegen im Text mit Pfeiltasten kann möglich sein oder auch nicht. Probieren Sie es aus! Wenn Sie bei der Eingabe mit Backspace nicht rückwärts löschen können, probieren Sie Strg+h oder wechseln Sie zurück in den Kommandomodus und verwenden Sie anschließend ein Kommando zum Löschen von Text.
I (großes i)	Wie i, jedoch wird der Text am Anfang der aktuellen Zeile eingefügt.
a	Wie i, jedoch wird der Text hinter dem Cursor eingefügt, nicht vor dem Cursor.
A	Wie i, jedoch wird der Text am Ende der aktuellen Zeile eingefügt.
o	Wie i, jedoch wird eine neue Zeile hinter der aktuellen Zeile eingefügt und der Cursor wird an den Anfang der neuen Zeile gesetzt.
O	Wie o, jedoch wird der Cursor ans Ende der aktuellen Zeile gesetzt.
R	Überschreibmodus einschalten. Sie können ab Cursorposition Zeichen für Zeichen mit neuen Zeichen überschreiben. Ansonsten gelten die gleichen Hinweise wie bei i.
r	Einzelnes Zeichen ersetzen: r drücken, dann neues Zeichen eingeben, dann mit Enter bestätigen. Sinnvoll, um einzelne Tippfehler zu korrigieren, ohne direkt in den Eingabemodus zu wechseln.

Tabelle 10.10: Kommandos zum Wechseln in den Eingabemodus

vi-Kommandos zum Löschen von Text

Kommando	Wirkung
`x`	Löscht das Zeichen unter dem Cursor. Die Zeichen rechts davon wandern eins nach links.
`X`	Löscht das Zeichen links vom Cursor. Hat eine Wirkung wie »rückwärts Löschen mit der Backspace-Taste«.
`D`	Löscht den Rest der Zeile ab Cursorposition.
`dd`	Löscht die gesamte aktuelle Zeile. Zeilen dahinter rücken vor.

Tabelle 10.11: vi-Kommandos zum Löschen von Text

vi-Kommandos zum Suchen und Ersetzen von Text

Kommando	Wirkung
`/Suchtext`	Sucht ab Cursorposition nach `Suchtext` in Richtung Dateiende und positioniert bei Auffinden auf die nächste Fundstelle. `Suchtext` wird als regulärer Ausdruck interpretiert. Näheres zu regulären Ausdrücken in ***Kapitel 12*** im Zusammenhang mit PHP.
`?Suchtext`	Sucht ab Cursorposition nach `Suchtext` in Richtung Dateianfang und positioniert bei Auffinden auf die nächste Fundstelle.
`n`	Weitersuchen, wenn Cursor auf Fundstelle steht.
`N`	Weitersuchen, wenn Cursor auf Fundstelle steht, jedoch in die umgekehrte Richtung.
`:1,$s/alt/neu/`	Sucht in der gesamten Datei nach `alt` und ersetzt es durch `neu`. Die Syntax `1,$` bedeutet: Suche ab Zeile 1 bis zur letzten Zeile. `3,6` würde nur in den Zeilen 3 bis 6 suchen und ersetzen. `alt` wird als regulärer Ausdruck interpretiert. Bei `neu` können in alt geklammerte Teilausdrücke mit `\1`, `\2` usw. referenziert werden.

Tabelle 10.12: vi-Kommandos zum Suchen und Ersetzen von Text

Diverse vi-Kommandos

Kommando	Wirkung
`:n`	Zur nächsten Datei wechseln, falls vi mit mehreren Dateinamen aufgerufen wurde.
`:n!`	Wie `:n`, jedoch ohne die aktuelle Datei zu sichern.
`:e#`	Zur vorherigen Datei wechseln.
`:e dateiname`	Eine andere Datei öffnen. `dateiname` kann auch ein vollständiger Pfadname sein. Die aktuelle Datei wird beendet.
`:e! dateiname`	Wie `:e`, jedoch ohne die aktuelle Datei zu sichern.
`:r dateiname`	Fügt den Inhalt von `dateiname` hinter der aktuellen Zeile ein. `dateiname` kann auch ein vollständiger Pfadname sein.
`:! kommando`	Führt `kommando` auf Shell-Ebene aus.

Tabelle 10.13: Diverse vi-Kommandos

10.4 Wichtige Konfigurationsdateien

Wichtige Konfigurationsdateien liegen normalerweise stets als Klartextdateien vor, d.h., Sie können den Inhalt dieser Dateien mit Kommandos wie `cat`, `more` und `less` ausgeben lassen oder mit einem Editor wie vi bearbeiten.

Alle wichtigen anwendungsunabhängigen Konfigurationsdateien liegen unterhalb des Verzeichnisses */etc*, teilweise direkt in diesem Verzeichnis, teilweise auch in weiteren Unterverzeichnissen.

Leider verwenden die unterschiedlichen Linux-Distributionen recht unterschiedliche Lösungen für einzelne Konfigurationsdateien, weshalb distributionsübergreifende Aussagen an manchen Stellen etwas vage bleiben müssen.

10.4.1 Allgemeine Systemkonfigurationsdateien

Ändern Sie die in diesem Abschnitt vorgestellten Dateien auf keinen Fall, wenn Sie nicht über weitreichende Systemkenntnisse verfügen. Die Dateien werden hier nur vorgestellt, damit Sie ein Grundverständnis für die Systemkonfiguration entwickeln.

Boot-Konfiguration in lilo.conf

Die meisten heutigen Linux-Rechner sind vom Typ her PCs, also mit Hardware ausgerüstet, auf der verschiedene Betriebssysteme laufen können. Dazu bedarf es eines so genannten Boot-Managers, der in der Lage ist, verschiedene installierte Betriebssysteme zu starten oder von anderen Boot-Managern gestartet zu werden. Ein solcher Boot-Manager ist das Programm *lilo* (steht für *Linux Loader*). Der Boot-Manager *lilo* bezieht seine Konfiguration aus der Datei */etc/lilo.conf*.

In *lilo.conf* wird festgelegt, welches Betriebssystem per Default geladen werden soll und welche Betriebssysteme welche Festplattenpartitionen laden sollen. Einträge darin, welche mit `image =` beginnen, zeigen auf eine Linux-Partition. Die Zuweisung an `image =` ist der Pfadname des Linux-Kernels. Unterhalb davon stehen diverse Optionen. Bei `label=` steht die Bezeichnung, unter der das Betriebssystem startet.

Runlevel-Definitionen in inittab

Das erste Programm, das beim Hochfahren eines Linux-Systems ausgeführt wird, ist das Programm *init*. Diesem wird als Parameter mitgeteilt, in welchem so genannten Runlevel das System hochfahren soll, z.B. *init 3*.

Die Definitionen der möglichen Runlevel stehen in der Datei */etc/inittab*. Die Definition der Runlevel variiert zwischen den Linux-Distributionen stark.

Relativ typisch ist diejenige von SUSE. Dabei steht Runlevel 0 für »Halt« (System heruntergefahren), Runlevel 1 für »Single User Mode«, Runlevel 2 und 3 für »Multi User Mode«, wobei 2 ohne und 3 mit Netzwerkanschluss bedeutet, und Runlevel 6 steht für »System neu starten«. Während der normalen Betriebsphase befindet sich ein über das Internet

administrierbarer Linux-Rechner nach dieser Unterscheidung in Runlevel 3. Der »Single User Mode« ist nur für Wartungsarbeiten gedacht.

Alle Zeilen von */etc/inittab*, die keine Leerzeilen oder Kommentarzeilen sind (# am Zeilenanfang), besteht aus vier Feldern, die durch Doppelpunkte getrennt sind. Das Feldschema lautet:

```
ID:Runlevel:Art:Prozess
```

Das Programm *init* arbeitet alle Zeilen aus */etc/inittab* ab, bei denen `Runlevel`, also Feld 2, dem übergebenen Runlevel entspricht.

`ID` ist eine eindeutige Bezeichnung des auszuführenden Prozesses, während `Prozess` der Pfadname der zugehörigen ausführbaren Datei ist, gegebenenfalls auch mit Parameterübergaben. Runlevel zeigt an, bei welchem Runlevel der entsprechende Prozess ausgeführt wird. Das Feld `Art` hat verschiedene Funktionen. Wenn sein Wert `wait` lautet, bedeutet dies: Führe den Prozess aus und warte, bis er fertig ausgeführt ist. Lautet der Wert `once`, wird der Prozess ausgeführt und es wird gleichzeitig weiter fortgefahren.

Die zentralen Aufrufe betreffen je nach Runlevel ein bestimmtes Shellscript mit dem Namen *rc*. Typische Einträge lauten etwa so:

```
l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l6:6:wait:/etc/rc.d/rc 6
```

Zentrale Shell-Vorgaben in profile und bashrc

Das Script */etc/profile* ist selbst ein Shellscript und das Startscript für die bash-Shell. Es führt unter anderem weitere Scripts in einem Unterverzeichnis */etc/profile.d* aus. Die dort befindlichen Scripts definieren zentrale Einstellungen für alle Benutzer, beispielsweise Default-Optionen für bestimmte Kommandos.

Eine ähnliche Aufgabe hat */etc/bashrc*, ebenfalls ein Shellscript. In diesem Script werden beispielsweise die zentralen umask-Einstellungen definiert, die festlegen, welche Rechte neu erzeugte Dateien automatisch erhalten. Weitere Definitionen in diesem Script betreffen das Default-Aussehen des Shell-Prompts oder so genannte Alias-Definitionen. Typische Alias-Definition lauten beispielsweise:

```
alias rm='rm -i'
alias ls='ls -l'
```

Damit wird festgelegt, dass bei Eingabe von `rm` in Wirklichkeit `rm -i` ausgeführt wird und bei Eingabe von `ls` in Wirklichkeit `ls -l`. Ersteres bewirkt, dass das Löschen von Dateien stets noch einmal bestätigt werden muss, bevor es ausgeführt wird, und Letzteres bewirkt, dass Verzeichnisauflistungen immer in der ausführlichen Darstellungsform erscheinen.

Neben diesen zentralen Varianten gibt es auch benutzerspezifische Varianten dieser Dateien in Form von Punktdateien, also *.bashrc* und *.profile*. Diese befinden sich in den persönlichen Startverzeichnissen der jeweiligen Benutzer und können vom Benutzer selbst geändert werden. Je nach System kann es sein, dass nur eines der beiden Scripts Verwendung findet.

Logging-Einstellungen in syslog.conf

Eines der wichtigsten Mittel, um Systemaktionen oder Aktionen von Benutzern zurückzuverfolgen, sind die Log-Dateien, die normalerweise unter */var/log* gespeichert werden. Die Log-Dateien werden laufend aktualisiert, indem jedes Ereignis, das in einer Log-Datei verzeichnet werden soll, ans Ende der betreffenden Log-Datei als einzeiliger Eintrag hinzugefügt wird. Das Mitloggen von Ereignissen übernimmt ein Daemon namens *syslogd*, der beim Systemstart seine Arbeit aufnimmt.

Welche Aktionen von *syslogd* mitgeloggt und in welcher Log-Datei die Ereignisse verzeichnet werden sollen, wird in der Datei */etc/syslog.conf* festgelegt. Ein Beispieleintrag aus dieser Datei: Der Datei können Sie die vollständigen Pfadnamen der Log-Dateien entnehmen. Werfen Sie ruhig auch einen Blick in die entsprechenden Log-Dateien. So bekommen Sie einen Eindruck davon, wie und was am System alles mitgeloggt wird und was Sie zur Not zurückverfolgen können.

10.4.2 Konfigurationsdateien für Benutzer und Gruppen

Auf den meisten heutigen Linux-Systemen speichern drei Konfigurationsdateien im Verzeichnis */etc* die wesentlichen Daten über eingerichtete Benutzer und Gruppen:

- Die Datei */etc/group* speichert Basiseigenschaften von Gruppen.
- Die Datei */etc/passwd* speichert Basiseigenschaften von Benutzern.
- Die Datei */etc/shadow* ist eine Erweiterung zur Datei *passwd*, die für mehr Passwortsicherheit sorgt.

Bearbeiten Sie diese Dateien jedoch nicht direkt, sondern verwenden Sie zur Benutzer- und Gruppenverwaltung die dazu vorgesehenen Kommandos (siehe ***Abschnitt 10.2.4***). Ein Blick in diese Dateien ist jedoch sinnvoll, um sich über vorhandene Gruppen, Benutzer und Abhängigkeiten zu informieren.

Aufbau der Datei group

Ein paar typische Einträge aus dieser Datei sehen folgendermaßen aus:

```
root:x:0:root
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
sys:x:3:root,bin,adm
adm:x:4:root,adm,daemon
tty:x:5:
disk:x:6:root,adm
```

```
lp:x:7:lp
mem:x:8:
kmem:x:9:
wheel:x:10:root
webworkers:x:500:
```

Jeder Eintrag besteht aus vier Feldern, die durch Doppelpunkte voneinander getrennt werden.

Feld 1 bezeichnet den Gruppennamen. Feld 2 ist für Gruppenpasswörter vorgesehen, wird jedoch nicht verwendet. Feld 3 enthält die Gruppen-ID (gid) und Feld 4 Gruppenmitglieder (falls mehrere, werden diese durch Kommata getrennt).

Viele Gruppen wie *root*, *bin*, *daemon*, *sys*, *adm* usw. wurden vom System selbst angelegt, um Berechtigungen für wichtige Systemprozesse festzulegen.

Aufbau der Datei passwd

Typische Einträge aus dieser Datei sehen folgendermaßen aus:

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
stefan:x:500:500::/home/stefan:/bin/bash
andreas:x:501:500::/home/andreas:/bin/bash
nina:x:502:500::/home/nina:/bin/bash
```

Jeder Eintrag besteht aus sieben Feldern, die durch Doppelpunkte voneinander getrennt werden.

Feld 1 bezeichnet den Benutzernamen. Feld 2 enthält auf klassischen Systemen das Benutzerpasswort in verschlüsselter Form. Da die passwd-Datei jedoch für diverse andere Programme lesbar sein muss, ist sie auch ein beliebtes Ziel für Angreifer von außen, die versuchen, über Internetkommunikation die Passwörter zu ermitteln. Zwar haben solche Angreifer dann immer noch das Problem, die Passwörter entschlüsseln zu müssen, doch sicherer ist es, diese gleich in eine andere Datei auszulagern, die nur vom Benutzer *root* lesbar ist, also auch nicht von Scripts oder Ähnlichem, die nicht unter root-Kennung laufen. Deshalb enthält dieses Feld auf neueren Systemen nur ein Platzhalterzeichen und die verschlüsselten Passwörter stehen in einer anderen Datei, z.B. in */etc/shadow*. Die Zahlen in den Feldern 3 und 4 geben die Benutzer-ID (uid) und die Gruppen-ID (gid) seiner Primärgruppe an. Durch Vergleich des Werts aus Feld 4 mit den Werten aus Feld 3 von */etc/group* kann der Gruppenname der Primärgruppe des Benutzers ermittelt werden. Feld 5 enthält in der Praxis meist keine Angabe. Hier können freie Daten zum Benutzer stehen, etwa Mailadressen, Telefonnummern usw. Feld 6 gibt den absoluten Pfadnamen des Heimatverzeichnisses des Benutzers an und Feld 7 den Pfadnamen der ausführbaren Datei, die dem Benutzer bei Anmeldung eine Shell zur Verfügung stellt. Am Dateinamen ist erkennbar, welche Shell der Benutzer erhält. Viele Benutzer wie *root*, *bin*, *daemon*, *adm*

usw. wurden vom System selbst angelegt, um Berechtigungen für wichtige Systemprozesse festzulegen.

Aufbau der Datei shadow

Die Datei */etc/shadow* existiert nur, wenn die Shadow Suite auf dem System installiert wurde.

Die Datei kann nur mit root-Kennung gelesen werden. Typische Einträge darin sehen wie folgt aus:

```
root:$lsdjkhdfadlökgjaavfdtli09443refad:12847:0:30:7:::
stefan:34pzi8gfrkogahfasdföligfgk43p594:12839:0:60:7:3::
```

Die Datei enthält neun Felder, wieder durch Doppelpunkte getrennt.

Feld 1 enthält den Benutzernamen. Feld 2 enthält das Benutzerpasswort in verschlüsselter Form. Feld 3 gibt an, wann das Passwort zuletzt geändert wurde. Der Wert bedeutet die Anzahl Tage seit dem 1.1.1970. Feld 4 gibt an, nach wie viel Tagen das Passwort frühestens geändert werden kann. Ein Wert von 0 bedeutet, dass das Passwort jederzeit geändert werden kann. In Feld 5 steht, nach wie viel Tagen das Passwort geändert werden muss, bevor es verfällt. In Feld 6 steht, ab wie viel Tagen vor dem Verfall des Passworts der Benutzer beim Einloggen gewarnt wird, sein Passwort rechtzeitig zu ändern. In Feld 8 schließlich steht, wie viel Tage nach Verfall eines Passworts das Benutzerkonto gesperrt wird. Feld 9 wird derzeit nicht benutzt.

Der Datei lässt sich ein wichtiges Feature der Shadow-Suite entnehmen: die Möglichkeit, beschränkte Gültigkeit für Passwörter zu definieren. Auf Mehrbenutzersystemen, bei denen die Sicherheit von hoher Bedeutung ist, ist dies ein wichtiger Faktor, um die Benutzer zu mehr Sicherheitsbewusstsein anzuhalten.

Falls Sie für vorhandene Benutzer beispielsweise Einstellungen zum Konto- oder Passwortverfall ändern möchten, benutzen Sie das Shell-Kommando `usermod`. Beispiele:

```
usermod -e 2005-12-31 stefan
```

Damit wird angewiesen, dass das Passwort des Benutzers `stefan` am 31.12.2005 verfällt.

```
usermod -f 7 stefan
```

Damit wird festgelegt, dass das Benutzerkonto von `stefan` 7 Tage nach Passwortverfall dauerhaft gesperrt wird.

10.4.3 Zeitgesteuerte Programmausführung (crontab)

Bei vielen professionellen Websites, die mithilfe von PHP oder vergleichbaren Scriptsprachen von Seitenbesuchern initiierte Daten auf dem Server speichern, fallen auch Arbeiten an, die besser nicht von den Scripts selber erledigt werden, sondern von unabhängigen Scripts. Das betrifft beispielsweise Scripts, die bestimmte Daten sichern oder aus gespei-

cherten Daten regelmäßige Reports erstellen, die sie dann an den Site-Betreiber per Mail versenden. Solche Scripts können in frei definierbaren regelmäßigen Abständen vom System automatisch ausgeführt werden, z. B. einmal täglich oder alle 20 Minuten.

Zuständig dafür ist ein Daemon namens *crond*. Er wird beim Systemstart geladen und sorgt dafür, dass Kommandos, Programme und Scripts entsprechend der Konfiguration regelmäßig gestartet werden. Die Konfiguration, also was wie oft oder wann gestartet werden soll, wird in crontab-Dateien gespeichert. Es gibt eine zentrale Datei */etc/crontab*, in der vom System angelegte Terminausführungen gespeichert sind. In dieser Datei sollten auch sonst nur solche Terminausführungen gespeichert werden, bei denen root-Kennung erforderlich ist.

Daneben können auch einzelne Benutzer eigene crontab-Dateien einrichten. Ob sie das dürfen, bzw. welche Benutzer es dürfen und welche nicht, ist in zwei Dateien namens */etc/cron.deny* und */etc/cron.allow* festlegbar. Sollten diese Dateien nicht existieren, haben alle Benutzer die Möglichkeit, eigene crontab-Dateien anzulegen und damit Programme, Scripts usw. terminiert auszuführen.

Geben Sie dazu als normaler Benutzer am Shell-Prompt ein:

```
crontab -e
```

Damit wird der Standardeditor des Systems (meistens vi) gestartet und eine leere oder bereits mit Daten gefüllte Datei angezeigt. Benutzerspezifische crontab-Dateien werden im Verzeichnis */var/spool/cron* abgelegt, und zwar mit dem Benutzernamen als Dateinamen. Das System ist jedoch in der Regel so eingerichtet, dass Sie die Datei nur über obiges Kommando bearbeiten können.



Abbildung 10.9: Typischer Inhalt einer zentralen crontab-Datei im vi-Editor

Aufbau der zentralen crontab-Datei

Im oberen Teil der crontab-Datei werden folgende wichtige Umgebungsvariablen für den cron-Daemon definiert:

- `SHELL=` der Pfadname der ausführbaren Datei der Shell, unter der die Terminkommandos gestartet werden sollen.
- `PATH=` Pfadnamen, in denen ausführbare Programme gesucht werden, die im unteren Teil der Datei nicht mit Pfadnamen angegeben sind. Mehrere Pfade werden durch Doppelpunkte getrennt.
- `MAILTO=` Benutzername des Benutzers, der Statusreports oder aufgetretene Fehler bei Terminausführungen mitgeteilt bekommt.
- `HOME=` das Startverzeichnis für den Benutzer *cron*.

Der untere Teil der Datei enthält die eigentlichen Angaben zu Terminausführungen. Jede Zeile stellt eine Aussage der folgenden Art dar: »starte Programm / Kommando / Script x dann und dann oder so und so oft«. Zeilen, die mit # beginnen, sind Kommentarzeilen.

Das genaue Schema lautet:

```
Minute Stunde Monatstag Monat Wochentag Benutzer Kommando
```

Die Teile werden durch Leerraum voneinander getrennt. Die ersten fünf Felder bestimmen, wann oder wie oft das Kommando ausgeführt werden soll.

Ein paar Beispiele:

```
*  *  *  *  *  root echo "ich: jede Minute"
0  *  *  *  *  root echo "ich: jede vollen Stunde"
20 *  *  *  *  root echo "ich: immer um 20 nach"
10 3  *  *  *  root echo "ich: täglich um 03:10 Uhr"
1  23 1  *  *  root echo "ich: am Monatsersten um 23:01 Uhr"
11 11 11 11 *  root echo "ich: am 11.11. um 11:11 Uhr"
5  0  *  *  1  root echo "ich: montags um 00:05 Uhr"
```

Aus den Beispielen geht die Wirkungsweise hervor. Ein Sternchen bedeutet: keine Angabe zu diesem Zeitfaktor. Ansonsten müssen sinnvolle Werte übergeben werden, d.h. bei Minuten 0 bis 59, bei Stunden 0 bis 23, bei Monatstagen 1 bis 28, oder wenn der Monat angegeben wird, von 1 bis Anzahl Tage im betreffenden Monat. Bei Wochentagen steht 0 für Sonntag, 1 für Montag usw. bis 6 für Samstag.

Der erste Eintrag in den Beispielen bleibt allerdings erklärungsbedürftig. Wenn keinerlei zeitliche Einschränkungen definiert werden, wird dies als »andauernd« interpretiert. Der Daemon *crond* überprüft jedoch nur jede Minute alle crontab-Dateien neu und führt gegebenenfalls Programme aus.

Es gibt noch speziellere Angaben. Ein paar Beispiele:

```
0 0   *  *  1-5  root echo "ich: Mo-Fr um Mitternacht"
10 */2 *  *  *   root echo "ich: alle 2 Stunden um 10 nach voll"
1 0 1  4,8,12 *  root echo "ich: 1.4., 1.8., 1.12. 00:01 Uhr"
```

Mit der Syntax `x-y` können Sie einen Bereich definieren, mit `*/x` einen Bruch, der so viel bedeutet wie »alle x Einheiten« und mit `x,y,z` eine Aufzählung der Art »dann, dann und dann«.

Bei den meisten Linux-Installationen gibt es bereits voreingestellte Scripts, die stündlich, täglich, wöchentlich und monatlich ausgeführt werden. Diese sind in der Regel unter den Dateinamen *cron.hourly*, *cron.daily*, *cron.weekly* und *cron.monthly* im Verzeichnis */etc* oder */etc/cron.d* gespeichert.

Einrichten von cron.deny und cron.allow

Um nur bestimmten Benutzern das Anlegen eigener crontab-Dateien zu ermöglichen, verbieten Sie am besten allen Benutzern das Anlegen und geben dann explizit Benutzer an, denen es erlaubt sein soll.

Rufen Sie dazu mit dem vi-Editor eine Datei */etc/cron.deny* auf. Wechseln Sie in den Einfügemodus und geben Sie nur das Wort `ALL` ein. Anschließend können Sie die Datei mit dem vi-Kommando `:w` speichern. Geben Sie dann `:e /etc/cron.allow` ein. Tragen Sie in dieser neuen Datei in jeder Zeile den Benutzernamen eines Benutzers ein, der crontab-Dateien anlegen können soll. Achten Sie auf korrekte Groß- und Kleinschreibung bei den Benutzernamen. Speichern Sie am Ende die Datei und beenden Sie vi.

Aufbau benutzerspezifischer crontab-Dateien

Der Aufbau benutzerspezifischer crontab-Dateien entspricht dem der zentralen Datei */etc/crontab* mit folgenden Unterschieden:

- Das Kopf-Feld `MAILTO` kann weggelassen werden.
- Bei den Termineinträgen kann das Feld mit dem Benutzernamen weggelassen werden.

10.5 Einfache Shellscripts

An verschiedenen Stellen dieses Kapitels war bereits von Shellscripts die Rede. Mithilfe von Shellscripts können Sie zahlreiche Aufgaben automatisieren, z.B. auch durch Einbinden als cron-Job. Beliebt sind Shellscripts auch, um sich eigene »Luxus-Kommandos« zu basteln oder um Informationen sammeln und anzeigen zu lassen.

Wir werden uns hier auf die bash-Shell beschränken und nur in Ansätzen die Möglichkeiten der Shell-Programmierung beschreiben.

10.5.1 Allgemeines zu Shellscripts

Da es so viele leistungsfähige Shell-Kommandos gibt, können Shellscripts ungemein viele Aufgaben wahrnehmen. Es gibt allerdings auch Grenzen der Shellscriptsprache. Richtige Programmiersprachen erlauben noch mehr Möglichkeiten, wie beispielsweise das Arbeiten mit Dateizeigern, die zeichengenaues Auslesen und Beschreiben von Dateien erlauben, oder mit komplexen Datenstrukturen. Deshalb sind programmiersprachenorientierte Scriptsprachen wie Perl entstanden. Während jedoch Perl und andere Scriptsprachen erst installiert werden müssen, ist die Shellscriptsprache immer verfügbar. Der Grund dafür ist, dass die Shellscriptsprache ein Teil der Shell selbst ist.

Die typische Dateiendung für Shellscripts ist ***.sh***. Allerdings kennt Linux wie bereits früher erwähnt keine strenge Abhängigkeit zwischen Dateiendung und Dateityp. Was ein Shellscript unter Linux zum Shellscript macht, ist nicht die Dateiendung, sondern wird durch zwei Faktoren bestimmt:

- Die Datei muss ausführbar sein für Benutzer, die sie ausführen können sollen.
- Die Datei muss in der ersten Zeile den Script-Interpreter angeben.

Während die erste Bedingung leicht durch Anwendung des Kommandos `chmod` erfüllbar ist, ist die zweite Bedingung für nicht eingefleischte Unix-Menschen erklärungsbedürftig. Wenn Sie ein Shellscript für die bash-Shell schreiben, lautet die erste Zeile typischerweise:

```
#!/bin/bash
```

Dabei werden die ersten beiden Bytes, also die Zeichen `#!`, als »shebang« bezeichnet (englisch auszusprechen – das Wort kann im Deutschen Verschiedenes bedeuten, z. B. »Bude«). Hinter der shebang-Zeichenfolge, die übrigens für alle Scriptsprachen unter Unix/Linux gleich ist, egal ob Perl, Python oder eben Shellscripts, folgt der absolute Pfadname zum Script-Interpreter. Im Fall von Shellscripts ist dies die ausführbare Datei der Shell selbst, bei der bash-Shell also typischerweise ***/bin/bash***. Bei jeder Datei, die versucht wird auszuführen, untersucht der Kommando-Interpreter, falls es sich nicht um ein kompiliertes Programm handelt, ob die shebang-Bedingung erfüllt ist. Wenn ja, kann der Dateiinhalt einfach an den in der ***Shebang-Zeile*** genannten Interpreter übergeben werden.

Shellscripts können mit jedem Texteditor erstellt werden. Der vi-Editor ist bestens geeignet.

Weiterführender Link:

Shell-Programmierung:
http://www.netzmafia.de/skripten/unix/unix8.html

Script-Beispiel, Ablageort und Ausführung

Das nachfolgende Script gibt ein paar wichtige Informationen für den aktuellen Benutzer aus:

```
#!/bin/bash
# Infos für den Benutzer
echo "Ihr Benutzername lautet: `whoami`"
echo "Ihr aktuelles Verzeichnis ist: `pwd`"
echo "Die aktuelle Systemzeit ist: `date`"
exit 0
```

Angenommen, Sie speichern das Script unter dem Dateinamen *myinfo* ab und erteilen ihm mit `chmod 744 myinfo` ausreichende Rechte. Im Verzeichnis, in dem es liegt, können Sie es nun mit `./myinfo` ausführen. Die Syntax `./` zur Bezeichnung des aktuellen Verzeichnisses ist in diesem Fall wichtig, sofern es ein Verzeichnis ist, das nicht im »Pfad« liegt. Welche Verzeichnisse im Pfad liegen, können Sie ermitteln, indem Sie am Shell-Prompt eingeben:

```
echo $PATH
```

Durch Doppelpunkte getrennt werden die Pfade angegeben, die im »Pfad« liegen, also beim Versuch, Programme, Kommandos oder Scripts auszuführen, automatisch durchsucht werden.

In der Praxis empfiehlt es sich, für persönliche Verwaltungs-Shellscripts unterhalb des persönlichen Benutzerverzeichnisses ein Verzeichnis namens *bin* anzulegen und die Rechte in geeigneter Form festlegen. Also so:

```
cd $HOME
mkdir bin
chmod 744 bin
```

Als Nächstes sollten Sie das neue Verzeichnis in Ihre PATH-Variable aufnehmen. Bearbeiten Sie dazu die Datei *.bashrc* in Ihrem Benutzerverzeichnis mit `vi .bashrc`. Notieren Sie z. B. am Ende der Datei eine neue Zeile wie diese:

```
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/home/Benutzeverzeichnis/bin
```

Dabei müssen Sie natürlich `Benutzerverzeichnis` durch den Verzeichnisnamen Ihres Benutzerverzeichnisses ersetzen. Beim nächsten Starten der Shell für Ihren Benutzernamen befindet sich das Verzeichnis, in dem Ihre eigenen Shellscripts liegen, dann im Pfad und Sie können sie von überall aus ohne Angabe von Pfadnamen wie normale Kommandos ausführen, also beispielsweise eingeben:

```
myinfo
```

Am Ende des obigen Beispielscripts ist eine Zeile mit dem Inhalt `exit 0` notiert. Dies sollten Sie in einem sauber programmierten Shellscript am Ende immer notieren. Das Kom-

mando `exit` teilt dem Interpreter mit, das Shellscript zu beenden. Der Wert 0 ist das »Ende mit gutem Ausgang«.

Im Rahmen der allgemeinen Syntax von Shellscripts sind noch folgende Punkte wichtig:

- Jedes Kommando kommt entweder in eine eigene Zeile oder es muss am Ende ein Semikolon (;) erhalten.
- Alles, was hinter einem Gatterzeichen (#) bis zum Ende der Zeile folgt, wird ignoriert, also als Kommentar behandelt.
- Scripts dürfen auch Leerzeilen enthalten sowie Einrückungen, um die Lesbarkeit zu erhöhen.

10.5.2 Parameter, Variablen und Funktionen

Ein Shellscript erfüllt wichtige Bedingungen moderner Programmiersprachen. Dazu gehört, Werte in Variablen zwischenspeichern zu können, Code in Subroutinen zu notieren, der nur bei Aufruf der Routine ausgeführt wird, Werte an Subroutinen übergeben zu können und Werte von ihnen zurückzuerhalten sowie Werte von außen, also beim Scriptaufruf, übergeben zu bekommen.

Parameter

Einem Script können beim Aufruf Parameter übergeben werden. Deren Werte stehen dem Script in Form von speziellen Variablen zur Verfügung. Ein kleines Beispiel:

```
#!/bin/bash
# Zählt Dateien eines Typs im Verzeichnis
ls -l *.$1 | wc -l
exit 0
```

Angenommen, Sie speichern dieses Script in einem Verzeichnis, das im Pfad liegt, unter dem Namen ***countfiles*** ab. Dann können Sie in einem beliebigen Verzeichnis beispielsweise eingeben:

```
countfiles php
```

Das Script ermittelt die Anzahl der Dateien des Typs ****.php*** im aktuellen Verzeichnis.

Dazu bekommt es die gewünschte Dateiendung als Parameter übergeben. Innerhalb des Scripts ist der Zugriff auf den ersten Parameterwert möglich durch `$1`, der auf einen zweiten Parameter durch `$2`, auf einen dritten durch `$3` usw. Auch `$0` gibt es – darin ist der Name des Scripts gespeichert, im Beispiel also `countfiles`.

Im Beispielscript wird `$1` in ein ls-Kommando eingebaut: `ls -l *.$1` wird, falls in `$1` der Wert `php` gespeichert ist, zu `ls -l *.php` umgesetzt oder, falls etwa `xml` gespeichert ist, zu `ls -l *.xml`. Damit werden jedoch nur alle Dateien des entsprechenden Typs aufgelistet, und zwar ausführlich, mit einer Datei pro Zeile. Was wir indes ermitteln möchten, ist nur deren Anzahl. Dazu reichen wir den Output des ls-Kommandos durch eine Röhre (Pipe),

also mithilfe des Zeichens |, an das Kommando wc durch. Dieses zählt normalerweise Wörter in einer Datei (wc = *word count*), kann aber auch, mit der Option -l aufgerufen, Zeilen zählen. So wie im Script notiert zählt es die Zeilen des Outputs vom ls-Kommando. Das Ergebnis wird ausgegeben.

Variablen

Ebenso, wie Sie am Shell-Prompt eigene Variablen definieren und mit Werten belegen können, ist das auch in einem Shellscript möglich. Eine Variable wird folgendermaßen definiert:

```
Name=Wert
```

oder:

```
Name="Wert mit Leerzeichen"
```

Dabei darf zwischen Name=Wert kein Leerzeichen stehen.

Die Wertzuweisung kann auch selber Variablen enthalten. Beispiel:

```
homedir="Dein Heimatverzeichnis ist: $HOME"
```

In diesem Fall enthält die Wertzuweisung eigenen Text. Innerhalb davon kommt die Variable HOME vor. Um Variableninhalte anzusprechen, muss dem Variablennamen ein Dollarzeichen vorangestellt werden. Mit $HOME wird also der Wert der vordefinierten Variablen HOME angesprochen.

Ferner kann eine Wertzuweisung Kommandos enthalten. Beispiel:

```
phps="Im Verzeichnis sind `ls -l *.php | wc -l` PHP-Dateien"
```

Die Wertzuweisung enthält eigenen Text sowie ein Kommando, das Sie schon aus dem Beispiel weiter oben kennen und das Dateien im Verzeichnis zählt. Kommandos innerhalb von Wertzuweisungen an Variablen müssen Sie in so genannten *Backticks* notieren. Auf der PC-Tastatur ist dies das Zeichen für den Accent grave. Um die Backticks zu erzeugen, halten Sie auf den meisten Tastaturen die ⇧-Taste gedrückt, tippen einmal auf die Taste mit den französischen Accent-Zeichen und anschließend die Leertaste.

Mit dem echo-Kommando können Variableninhalte ausgegeben werden. Noch mal das Beispiel zuvor, aber diesmal mit anschließender Ausgabe:

```
phps="Im Verzeichnis sind `ls -l *.php | wc -l` PHP-Dateien"
echo $phps
```

Damit gibt das Script so etwas aus wie: Im Verzeichnis sind 14 PHP-Dateien.

Variablen kann auch jederzeit ein neuer Wert zugewiesen werden. Beispiel:

```
x=abc
y=def
x=ghi
echo $x
```

Ausgegeben wird am Ende `ghi`. Durch die zweite Zuweisung an `x` wurde die erste überschrieben.

Sie können jedoch auch Variableninhalte verknüpfen, sogar den bisherigen Inhalt einer Variablen. Beispiel:

```
x=abc
y=def
x=$x$y
echo $x
```

Damit wird am Ende `abcdef` ausgegeben, denn die zweite Zuweisung an `x` besteht aus dem alten Wert von `x` und dem von `y`. Das gilt übrigens auch für Zahlen – diese werden ebenfalls als Zeichenketten behandelt. Wenn `x` den Wert 10 hätte und `y` den Wert 20, dann würde durch `$x+$y` als Wert `10+20` herauskommen, nicht aber das Ergebnis dieser Rechenoperation. Um einfache Rechenoperationen mit Variableninhalten durchzuführen, können Sie das Kommando `expr` benutzen. Um bei unserem Beispiel zu bleiben:

```
x=10
y=20
x=`expr $x + $y`
echo $x
```

Damit wird wie erhofft 30 ausgegeben. Das expr-Kommando verarbeitet Variablen, deren Inhalt sich numerisch interpretieren lässt, und Zahlen. Bei Kommazahlen ist der Punkt das Dezimalzeichen. Neben den Grundrechenarten + – * / beherrscht das expr-Kommando auch noch Vergleiche und die logische Wahrheitsbewertung. Informationen zu den Möglichkeiten bietet das Shell-Kommando `man expr`.

Vordefinierte Variablen

Innerhalb eines Shellscripts stehen auch vordefinierte Variablen zur Verfügung, die an Stelle sprechender Namen aus Zahlen, Satzzeichen oder Ähnlichem bestehen. Nachfolgende Tabelle listet die vordefinierten Variablen auf:

Kommando	Wirkung
`!`	Wahr, wenn eine Bedingung nicht zutrifft.
`$s1 = $s2`	Wahr, wenn die Variablen `$s1` und `$s2` die gleiche Zeichenkette enthalten.
`$s1 != $s2`	Wahr, wenn die Variablen `$s1` und `$s2` nicht die gleiche Zeichenkette enthalten.

Tabelle 10.14: Vordefinierte Variablen für Shellscripts

Kommando	Wirkung
`-n $s`	Wahr, wenn die in `$s` gespeicherte Zeichenkette mehr als 0 Zeichen enthält.
`-z $s`	Wahr, wenn die in `$s` gespeicherte Zeichenkette 0 Zeichen enthält, also leer ist.
`$n1 -eq $n2`	Wahr, wenn die in `$n1` gespeicherte Zahl gleich der in `$n2` gespeicherten Zahl ist.
`$n1 -eq $n2`	Wahr, wenn die Zahl in `$n1` größer oder gleich der von `$n2` ist.
`$n1 -ge $n2`	Wahr, wenn die Zahl in `$n1` größer oder gleich der von `$n2` ist.
`$n1 -gt $n2`	Wahr, wenn die Zahl in `$n1` größer als die von `$n2` ist.
`$n1 -le $n2`	Wahr, wenn die Zahl in `$n1` kleiner oder gleich der von `$n2` ist.
`$*`	Alle ans Script übergebenen Parameter als eine Zeichenkette.
`$@`	Alle ans Script übergebenen Parameter als Liste (geeignet etwa zur Abarbeitung in for-Schleifen).
`$#`	Anzahl der übergebenen Parameter.
`$?`	Exit-Status des zuletzt ausgeführten Kommandos.
`$$`	Prozessnummer (PID) dieses Shellscripts.
`$!`	Prozessnummer (PID) des zuletzt ausgeführten Hintergrundkommandos.
`$0`	Aufgerufener Dateiname dieses Shellscripts.
`$1 bis $n`	Die einzelnen ans Script übergebenen Parameter.

Tabelle 10.14: Vordefinierte Variablen für Shellscripts (Fortsetzung)

Funktionen

Scriptcode, der an mehreren Stellen im Script benötigt wird, kann in Funktionen untergebracht werden. Ein einfaches Beispiel:

```
#!/bin/bash
make_html_from_title() {
   echo "<html><title>$2</title>" >> $1
   echo "<body><h1>$2</h1></body></html>" >> $1
}
make_html_from_title "test.htm" "Ein kleiner Test"
```

Der Beispielauszug definiert eine Funktion namens `make_html_from_title`. Hinter dem Funktionsnamen muss ein rundes Klammernpaar `()` notiert werden. Parameter, die von der Funktion erwartet werden, müssen nicht in den Klammern benannt und deklariert werden. Sie können einfach innerhalb der Funktion mit den Spezialvariablen `$1` für den ersten übergebenen Parameter, `$2` für den zweiten, `$3` für den dritten usw. angesprochen werden.

Im Beispiel wird mithilfe des echo-Kommandos HTML-Code erzeugt. Dabei wird der Wert des zweiten Parameters, der an die Funktion übergeben wird, an zwei Stellen mit `$2` in den HTML-Code eingebaut.

Normalerweise würde `echo` seine Ausgabe einfach auf die Standardausgabe schreiben. Im Beispiel wird die Ausgabe jedoch ans Ende einer Datei geschrieben, deren Name in `$1` gespeichert ist.

Beim Aufruf muss die Funktion natürlich so bedient werden, dass sie etwas Sinnvolles tut. Wichtig beim Funktionsaufruf ist, dass diesmal keine runden Klammern notiert werden. Es wird einfach der Funktionsname notiert und dahinter, jeweils durch Leerzeichen getrennt, die gewünschten Parameter. Parameter mit Leerzeichen müssen in hohe Anführungszeichen gesetzt werden, wie im Beispiel zu sehen.

10.5.3 Bedingungen, Fallunterscheidungen und Schleifen

Die Shellprogrammierung kennt alle programmiersprachentypischen Kontrollstrukturen.

Verzweigungen

Sie können Scriptanweisungen abhängig ausführen, also nur dann, wenn eine formulierbare Bedingung zutrifft. Für den Fall, dass diese nicht zutrifft, können Sie auch einen else-Zweig definieren.

Es gibt jedoch bei der bash-Shell (im Gegensatz zu manch anderen Shells) keine andere Möglichkeit, Bedingungen zu formulieren, als Kommandos zu notieren. Ob eine Bedingung erfüllt ist oder nicht, entscheidet dann der exit-Status des Kommandos. Im »Gutfall« liefern Kommandos unter Linux stets den exit-Status 0 zurück. Die Bedingung gilt also als erfüllt, wenn das Kommando, welches als Bedingung dient, erfolgreich ausgeführt wurde. Ein Beispiel:

```
#!/bin/bash
if pwd | grep ^$HOME$
then
   echo "Sie befinden sich im Heimatverzeichnis"
else
   echo "Sie befinden sich nicht im Heimatverzeichnis"
fi
exit 0
```

Bedingungen werden durch `if` eingeleitet. Dahinter können ein oder mehrere Kommandos folgen. Pipes wie im Beispiel zählen als ein Kommando. Wenn der exit-Status insgesamt 0 ist, gilt die Bedingung als erfüllt. Im Beispiel ist dies der Fall, wenn das grep-Kommando, welches Text nach regulären Ausdrücken untersucht, in dem Output, den das pwd-Kommando liefert, den Inhalt der Variablen `$HOME` findet. Das pwd-Kommando liefert das aktuelle Verzeichnis. Wenn in `$HOME` der gleiche Wert gespeichert ist, befindet sich der Benutzer in seinem Heimatverzeichnis, ansonsten nicht.

Glücklicherweise gibt es ein spezielles Kommando namens `test`, das unter fast allen Linux-Installationen verfügbar ist. Dieses Kommando erlaubt es, Ausdrücke zu bewerten, und liefert einen entsprechenden exit-Status. Als besondere Raffinesse ist dieses Kommando häufig auch als Link unter dem Dateinamen *[* gespeichert, also nur eine öffnende eckige Klammer. Da es außerdem als letzten Parameter eine geschlossene eckige Klammer *]* akzeptiert, findet man in der Shellprogrammierung häufig »scheinbare« if-Bedingungen wie diese:

```
if [ ! -e $dir ]; then
   mkdir $dir;
fi
```

Das ist das Gleiche wie:

```
if test ! -e $dir ; then
   mkdir $dir;
fi
```

Wichtig ist bei der Notationsform mit den eckigen Klammern, dass nach der öffnenden und vor der schließenden Klammer ein Leerzeichen stehen muss.

Mithilfe des test-Kommandos lassen sich Bedingungen für Verzweigungen und Schleifen formulieren. Nachfolgende Tabelle listet Möglichkeiten des test-Kommandos auf:

Kommando	Wirkung
`!`	Wahr, wenn eine Bedingung nicht zutrifft.
`$s1 = $s2`	Wahr, wenn die Variablen `$s1` und `$s2` die gleiche Zeichenkette enthalten.
`$s1 != $s2`	Wahr, wenn die Variablen `$s1` und `$s2` nicht die gleiche Zeichenkette enthalten.
`-n $s`	Wahr, wenn die in `$s` gespeicherte Zeichenkette mehr als 0 Zeichen enthält.
`-z $s`	Wahr, wenn die in `$s` gespeicherte Zeichenkette 0 Zeichen enthält, also leer ist.
`$n1 -eq $n2`	Wahr, wenn die in `$n1` gespeicherte Zahl gleich der in `$n2` gespeicherten Zahl ist.
`$n1 -eq $n2`	Wahr, wenn die Zahl in `$n1` größer oder gleich der von `$n2` ist.
`$n1 -ge $n2`	Wahr, wenn die Zahl in `$n1` größer oder gleich der von `$n2` ist.
`$n1 -gt $n2`	Wahr, wenn die Zahl in `$n1` größer als die von `$n2` ist.
`$n1 -le $n2`	Wahr, wenn die Zahl in `$n1` kleiner oder gleich der von `$n2` ist.
`$n1 -lt $n2`	Wahr, wenn die Zahl in `$n1` kleiner als die von `$n2` ist.
`$n1 -ne $n2`	Wahr, wenn die Zahlen in `$n1` und `$n2` ungleich sind.
`-b $f`	Wahr, wenn der in `$f` gespeicherte Dateiname existiert und die Datei ein block-orientiertes Gerät ist.
`-c $f`	Wahr, wenn `$f` existiert und ein zeichenorientiertes Gerät ist.
`-d $f`	Wahr, wenn `$f` existiert und ein Verzeichnis ist.
`-f $f`	Wahr, wenn `$f` existiert und eine normale Datei ist.
`-g $f`	Wahr, wenn `$f` existiert und das Gruppen-ID-Bit gesetzt ist.
`-h $f`	Wahr, wenn `$f` existiert und ein symbolischer Link ist.
`-b $f`	Wahr, wenn `$f` existiert und das Sticky-Bit gesetzt ist.
`-p $f`	Wahr, wenn `$f` existiert und eine benannte Pipe ist.
`-u $f`	Wahr, wenn `$f` existiert und das Setuid-Bit gesetzt ist.
`-w $f`	Wahr, wenn `$f` existiert und geändert oder gelöscht werden kann.
`-x $f`	Wahr, wenn `$f` existiert und eine ausführbare Datei ist.

Tabelle 10.15: Bedingungen mit dem test-Kommando

Kommando	Wirkung
`B1 -a B2`	Wahr, wenn Bedingung `B1` und Bedingung `B2` wahr sind.
`B1 -o B2`	Wahr, wenn wenigstens eine der beiden Bedingungen `B1` oder `B2` zutrifft.

Tabelle 10.15: Bedingungen mit dem test-Kommando (Fortsetzung)

Bedingte Ausführungen mit `if` müssen am Ende, wie in den Beispielen zu sehen, mit `fi` abgeschlossen werden. Der if-Zweig selbst muss stets mit `then` eingeleitet werden. Ein else-Zweig kann mit `else` festgelegt werden. Für die Formulierung mehrerer Bedingung steht außerdem `elif` (*else if*) zur Verfügung. Ein typisches Konstrukt sieht folgendermaßen aus:

```
if [ -d backup ]
then
   # tu was
elif [ -h backup ]
then
   # tu was anderes
else
   # tu noch was anderes
fi
```

Fallunterscheidungen

Wenn mehrere Werte, die eine Variable haben kann, unterschieden werden sollen, um abhängig davon unterschiedliche Anweisungen auszuführen, bietet sich die Syntax der Fallunterscheidung an. Ein Beispiel:

```
#!/bin/bash
echo "Zahl zwischen 1 und 3 eingeben: "
read line
case "$line" in
   1) echo "Sie haben 1 Wunsch frei!";;
   2) echo "Sie haben 2 Wünsche frei!";;
   3) echo "Sie haben 3 Wünsche frei!";;
esac
exit 0
```

Das Script fordert den Benutzer zu einer Eingabe auf. Mit dem read-Kommando kann der Benutzer etwas eingeben und mit Enter abschicken. In der Variable hinter `read` (im Beispiel in der Variablen `line`) wird dann die vom Benutzer eingegebene Zeichenkette gespeichert.

Auf den Wert der Variablen `$line` wird im Beispiel eine Fallunterscheidung angewendet. Solche Fallunterscheidungen beginnen mit der Konstruktion `case "$Variablenname" in` und enden mit `esac`. Die unterschiedenen Werte erhalten am Ende eine schließende runde Klammer. Abhängig von jedem unterschiedenen Wert können ein oder mehrere Kommandos notiert werden. Diese müssen jedoch alle mit doppeltem Semikolon abgeschlossen werden.

for-Schleifen

Solche Schleifen eignen sich zur Abarbeitung einer Liste von Werten. Für jeden Listenwert werden ein oder mehrere Kommandos wiederholt. Ein einfaches Beispiel:

```
#!/bin/bash
for i in `ls`
do
   if [ -f $i ]
   then
      if [ -w $i ]
      then
         echo "$i können Sie bearbeiten oder löschen"
      else
         echo "$i können Sie nicht bearbeiten oder löschen"
      fi
   fi
done
exit 0
```

Das Beispielscript gibt für reguläre Dateien im aktuellen Verzeichnis aus, ob der Benutzer diese bearbeiten und löschen kann oder nicht. Bearbeiten oder löschen kann er sie dann, wenn er Schreibberechtigung für die Datei hat.

Dazu wird mit dem Kommando `ls` eine Liste der Dateien im Verzeichnis erzeugt. Auf eine solche Liste ist eine for-Schleife anwendbar. Mit `for i in `ls`` wird formuliert: »für jeden Eintrag der Liste, die das ls-Kommando liefert, wobei der aktuelle Eintrag jeweils in der Variablen `i` gespeichert ist«. Was hinter dem Schlüsselwort `in` steht, muss also eine Liste sein. Viele Kommandos wie etwa `ls`, `ps`, `who` und andere erzeugen geeignete Listen. Aber auch die vordefinierte Variable `$@` lässt sich an dieser Stelle prima verwenden, da sie eine Liste der ans Script übergebenen Parameter speichert.

Der Schleifenkörper muss innerhalb der Schlüsselwörter `do` und `done` stehen.

Im Beispiel wird zunächst abgefragt, ob der aktuelle Verzeichniseintrag eine reguläre Datei ist. Wenn nicht, wird gleich der nächste Schleifendurchgang gestartet. Wenn ja, wird in einer inneren if-Verzweigung abgefragt, ob die Datei für den Benutzer beschreibbar ist. Abhängig vom Ergebnis wird eine entsprechende Meldung ausgegeben.

while-Schleifen

Schleifen mit `while` eignen sich dann, wenn die Anzahl der Schleifendurchläufe nicht wie bei `for` durch die Länge einer Liste festgelegt ist. Stattdessen wird bei while-Schleifen ähnlich wie bei if-Verzweigungen eine Bedingung formuliert. Solange die Bedingung wahr ist, wird der Schleifenkörper wiederholt. Innerhalb des Schleifenkörpers muss dafür gesorgt werden, dass die Schleifenbedingung irgendwann unwahr wird, ansonsten entsteht eine Endlosschleife. Ein Beispiel:

```
#!/bin/bash
echo "Raten Sie eine Zahl zwischen 1 und 9: "
```

```
read n
while [ $n -ne 5 ]
do
  echo "falsch: Raten Sie erneut: "
  read n
done
echo "RICHTIG!"
exit 0
```

Im Beispielscript wird der Benutzer aufgefordert, eine Zahl zwischen 1 und 9 zu raten und einzugeben. Eine davon, nämlich 5, ist die »richtige« Zahl. Mit `read` wird die Benutzereingabe zunächst in die Variable `n` eingelesen. Dann wird in einer while-Bedingung mithilfe des test-Kommandos geprüft, ob die Zahl ungleich 5 ist. Solange dies der Fall ist, ist die Schleifenbedingung wahr und der Schleifenkörper wird ausgeführt. Innerhalb des Schleifenkörpers werden die Eingabeaufforderung und das Einlesen einer neuen Zahl wiederholt. Auf diese Weise verändert sich der Wert von `$n` durch die Benutzereingabe und irgendwann gibt dieser hoffentlich 5 ein. Dann ist die Schleifenbedingung unwahr, die Schleife wird verlassen und das echo-Kommando am Ende des Scripts wird ausgeführt.

Der Schleifenkörper wird wie bei for-Schleifen in `do` bzw. `done` eingeschlossen.

10.5.4 Komplettbeispiel: ein Errrorlog-Analyse-Script

Das Thema Shellscript-Programmierung soll abgeschlossen werden durch ein praxisnahes Script für Mieter eines Root-Servers. Das Script wertet die Error-Log-Datei des Apache Webservers aus und gibt in einer Liste absteigend sortiert die Dateien aus, die am häufigsten nicht gefunden wurden, also Requests von Browsern oder anderen Webclients, die nicht existierende Seiten oder andere Ressourcen angefordert haben. Dabei werden wir auch noch einige neue Kommandos kennen lernen und sehen, wie mächtig komplexere Pipes sein können.

Hier zunächst das vollständige Listing des Shellscripts – nennen wir es ***top404report*** (da 404 der HTTP-Statuscode für nicht gefundene Ressourcen ist):

Listing 10.1: Shellscript »top404report«

```
#!/bin/bash
errorfile=/cygdrive/c/Programme/Apache/Apache2/logs/error_log
if [ ! -f $errorfile ]
then
   exit -1
fi
if [ ! $1 ]
then
   exit -2
fi
for x in `grep "File does not exist" $errorfile | awk '{print $13}' | sort  |
uniq`
do
  grep $x $errorfile | wc -l | tr -d '\n'
```

```
  echo " mal angefragt und nicht gefunden: $x"
done |  sort -rn | head -$1
exit 0
```

Zu Beginn des Scripts wird der volle Pfadname der Errorlog-Datei in der Variablen `errorfile` gespeichert. Wenn die Datei nicht existiert oder wenn dem Script beim Aufruf kein Parameter übergeben wird, wird die weitere Ausführung mit negativem exit-Status beendet.

Als Parameter erwartet das Script eine Zahl. Diese teilt ihm mit, wie viele Zeilen es ausgeben soll, also wie lang die Liste der Top-404-Ressourcen sein soll. Ein typischer Aufruf wäre also `top404report 10`.

Das aus mehreren Pipes zusammengesetzte Kommando der for-Schleife ist wie folgt aufzulösen. Das Kommando `grep "File does not exist" $errorfile` durchsucht die Errorlog-Datei zunächst nach Zeilen, in denen die Zeichenkette `File does not exist` vorkommt. Dies ist die Standardmeldung, die der Apache Webserver bei Auftreten eines 404-Statuscodes in die Errorlog-Datei einträgt. Als Ergebnis dieses Kommandos würden alle Zeilen ausgegeben, die diese Zeichenkette enthalten.

Das Ergebnis wird jedoch nicht ausgegeben, sondern über eine Pipe an das Kommando `awk` weitergereicht. Hinter `awk` verbirgt sich eigentlich ein kompletter Scriptinterpreter mit eigener Scriptsprache. Es handelt sich um ein mächtiges Tool zur Textmanipulation. Suchen und Ersetzen in Dateien ist nur eine von vielen Möglichkeiten. Auch via Kommandozeile kann `awk` aufgerufen werden und nützliche Dienste leisten. Eine Übersicht über die Möglichkeiten bieten die Hilfeseiten (Kommandos `man awk` oder `info awk`).

Beim Aufruf über die Kommandozeile muss der Scriptinhalt in geschweifte Klammern eingeschlossen werden. In unserem Beispiel wird mit `awk '{print $13}'` eine Anweisung ausgeführt, die den Inhalt des 13. »Feldes« der von `grep` via Pipe übergebenen Zeilen ausgibt. Hintergrund ist, dass `awk` jede Textzeile als »Datensatz« behandeln kann. Dabei wird per Default jedes Leerraumzeichen als Feldtrenner behandelt. Mit `awk '{print $13}'` wird also das 13. Wort oder Element einer Zeile ausgegeben. Betrachten wir zum Verständnis eine typische Zeile aus der Errorlog-Datei des Apache:

```
[Mon Feb 28 08:27:11 2005] [error] [client 217.232.154.250] File does not exist:
/var/www/selfhtml.org/de/dokumente/favicon.ico
```

Zählt man in diesem Eintrag die durch Leerzeichen getrennten Blöcke, so stellt sich heraus, dass der 13. Block die Pfadangabe am Ende der Zeile ist. Durch die Pipe an `awk` bleibt also eine Liste von Pfadnamen wie */var/www/selfhtml.org/de/dokumente/favicon.ico* übrig.

Damit jedoch noch nicht genug. Diese Liste wird nun durch eine weitere Pipe an das Kommando `sort` durchgereicht. Dieses sortiert die Einträge alphabetisch. Dieser Sortiervorgang ist für unser Ziel jedoch eigentlich gar nicht von Belang. Er wird nur deshalb angestoßen, weil es ein Kommando namens `uniq` gibt, das aus einer sortierten Liste alle doppelten Einträge entfernt. Eine weitere Pipe, die das Ergebnis von `sort` an `uniq` durch-

reicht, bewirkt also, dass am Ende für jeden Pfadnamen, der ein oder mehrmals nicht gefunden wurde, ein Listeneintrag existiert. Damit hat die Liste genau die Charakteristik, die sie für die for-Schleife braucht.

Innerhalb der for-Schleife wird nun für jeden Pfadnamen der Liste ermittelt, wie oft dieser Pfadname in der Errorlog-Datei vorkommt. In der Variablen `x` steht dabei jeweils der aktuelle Pfadname. Die in `errorfile` gespeicherte Datei wird mit `grep` nach `x` durchsucht. Da uns dabei nur interessiert, in wie vielen Zeilen `x` vorkommt, wird das Ergebnis von `grep` mittels einer Pipe an das Kommando `wc -l` durchgereicht, das die Zeilen zählt. Das Pipe-Ergebnis enthält nun neben der gewünschten Zahl jedoch jeweils noch ein abschließendes Zeilenumbruchzeichen. Um dies zu eliminieren, wird das Ergebnis in einer weiteren Pipe an das Kommando `tr` übergeben. Dieses übersetzt oder löscht Zeichen in einer Zeichenkette. Das Kommando `tr -d \n` löscht Zeilenumbruchzeichen. Die Option `-d` steht für Löschen und `\n` ist die Schreibweise, um ein Zeilenumbruchzeichen sichtbar zu notieren. Damit ist die nackte Anzahl von Vorkommnissen in der Liste ermittelt.

Im nächsten Kommando innerhalb des Schleifenkörpers der for-Schleife wird das Ergebnis ausgegeben. Anschließend wird die for-Schleife mit `done` beendet. Genauer gesagt, wird sie das nicht. Hinter dem abschließenden `done` folgen noch zwei Pipes und zwei entsprechende Kommandos.

Der Grund ist, dass wir bislang noch gar nicht den ans Script übergebenen Parameter berücksichtigt haben. Es soll ja nicht für jede nicht gefundene Ressource ausgegeben werden, wie oft diese nicht gefunden wurde, sondern nur für die soundso viel am häufigsten nicht gefundenen Ressourcen. Also muss das Ergebnis erstens noch numerisch absteigend sortiert werden nach den am häufigsten nicht gefundenen Ressourcen und zweitens sollen vom Sortierergebnis nur so viele Zeilen ausgegeben werden, wie im ans Script übergebenen Wert angefordert wurden.

Die numerisch absteigende Sortierung wird von `sort -rn` besorgt. Die Option `-r` steht für absteigende Sortierung und `-n` für numerische Sortierung. Das Ergebnis der Sortierung wird dann noch an das Kommando `head` durchgereicht. Dieses gibt von einer Datei oder einem Input nur die ersten Zeilen aus. Wie viele Zeilen, kann als Parameter übergeben werden. In unserem Script übergeben wir `$1`, also den Parameter, den das Script selbst erhalten hat.

Etwas schwer nachvollziehbar ist möglicherweise, warum innerhalb der for-Schleife mit `echo` Inhalte ausgegeben werden, die aber dann doch nicht am Bildschirm erscheinen, sondern erst noch durch die beiden Pipes gefiltert werden, die hinter der for-Schleife notiert sind. Dazu ist es einfach wichtig, sich zu merken, dass bei einer Pipe grundsätzlich alles, was vor ihr auf die Standardausgabe geschrieben wird, nicht geschrieben wird, sondern erst durch die Pipe muss. Und Pipes sind auf alle erdenklichen Kommandos anwendbar, auch auf das for-Kommando. Denn aus Shell-Sicht ist eine for-Schleife auch nichts anderes als ein Kommando.

11 Webseiten in HTTP-Umgebung

In diesem Kapitel wird dargestellt, wie sich ein HTML-Dokument in einer HTTP-Umgebung im Unterschied zu einer lokalen Umgebung verhält und welche Möglichkeiten es gibt, serverseitig dynamische Inhalte zu erzeugen. Letzteres werden wir zwar später noch ausführlich in Zusammenhang mit PHP behandeln. Doch dieses Kapitel möchte zeigen, welche anderen Möglichkeiten es noch gibt und in welchen Fällen diese PHP sogar vorzuziehen sind.

11.1 Webseiten serverseitig

In ***Abschnitt 9.3.3*** behandelten wir das HTTP-Protokoll und die Kommunikation zwischen einem Browser und einem Webserver. Beide – sowohl der Browser als auch der Webserver, sind in der Lage, HTTP-Kopfdaten zu lesen und zu senden. Aus der HTTP-Kommunikation ergeben sich einige Besonderheiten, die Sie kennen müssen, wenn Sie Webseiten auf Servern bereitstellen.

11.1.1 URIs und Pfade

Jede Datei, die sich innerhalb der ***Document Root*** eines Webservers befindet und von einem Webclient wie etwa einem Browser oder dem Robot einer Suchmaschine via HTTP aufgerufen werden kann, hat einen eigenen URI. In ***Abschnitt 4.8.1*** lernten Sie in Zusammenhang mit Hyperlinks in HTML den Aufbau eines URIs bereits kennen. URIs für Webseiten beginnen mit ***http://*** oder, wenn Secure Socket Layer aktiviert sind, mit ***https://***. Dahinter kann eine IP-Adresse, ein Domainname oder ein Hostname folgen. Als Nächstes kann eine Portangabe folgen, sofern diese vom Standardwert 80 abweicht. Im Anschluss daran folgt der absolute Pfadname der gewünschten Datenressource. »Absolut« bezieht sich dabei jedoch nicht auf das Wurzelverzeichnis des Server-Rechners, sondern auf das im Webserver eingerichtete Startverzeichnis für Webseiten, eben der Document Root.

Wenn Sie in Ihrer lokalen Entwicklungsumgebung einen Webserver installiert haben, können Sie die Document Root auf jeden Fall über den URI ***http://127.0.0.1/*** erreichen. Dies ist die so genannte Loopback-Adresse für den eigenen Hostrechner. Wenn sich Ihr Rechner in einem LAN befindet und innerhalb des LAN eine LAN-typische IP-Adresse hat, ist die Document Root auch darüber erreichbar. IP-Adressen in LANs werden dann vergeben, wenn TCP/IP als Übertragungsprotokoll im LAN eingesetzt wird und/oder wenn der Anschluss ans Internet letztlich über das LAN realisiert wird, sei es durch einen Hardwarerouter oder einen als Router fungierenden Rechner.

LAN-typische IP-Adressen beginnen in kleineren LANs mit 192.168. Dahinter folgt meist noch eine fixe Zahl, die bei sternförmigen LANs mit Internetanschluss vom Router vorgegeben wird. Die vierte Zahl bestimmt dann den individuellen PC. In großen Netzen wird auch mit IP-Adressen gearbeitet, die mit 10 beginnen. Alles dahinter ist vom Netzverwalter frei definierbar. Im öffentlichen IP-Adressraum sind diese IP-Adressräume bewusst ausgespart, so dass sie in jedem LAN intern verwendet werden können.

Wenn Ihr PC also eine LAN-typische IP-Adresse wie 192.168.101.3 hat, dann können Sie die Document Root des lokalen Webservers auch über *http://192.168.101.3/* erreichen.

Außerdem sind noch die Einträge in der Datei *hosts* maßgeblich, die sich auf neueren Windows-Systemen unter *[windows]\system32\drivers\etc* befindet und auf Linux-Systemen unter */etc*. Dort sollte auf jeden Fall eine Zuweisung von `127.0.0.1` an den Hostnamen `localhost` erstellt werden, sofern diese noch nicht existiert. Anschließend ist die lokale Document Root auch unter *http://localhost/* erreichbar. Durch weitere Zuweisungen von Hostnamen an die Loopback-Adresse oder die IP-Adresse im LAN ist die Document Root auch unter diesen anderen Hostnamen erreichbar. Falls im Apache Webserver Virtual Hosts eingerichtet werden, sind auf diese Weise auch mehrere Document Roots durch unterschiedliche Hostnamen ansprechbar.

Verzeichnisse in Pfadnamen werden bei URIs stets durch einfache Schrägstriche getrennt. Außerdem ist die Regelung für Sonderzeichen in URIs zu beachten, wie in ***Abschnitt 4.8.6*** behandelt. Ein Verzeichnis- oder Dateiname *münz* muss in einem URI in der Form *m%FCnz* notiert werden. Die Kodierung ist notwendig, weil beim Übertragen über Gateways immer noch der ASCII-Zeichensatz als sicherster kleinster Nenner gilt. Server und Browser lösen solche Kodierungen jedoch korrekt auf.

11.1.2 Default-Dateinamen und Verzeichnis-Browsing

In der Konfiguration des Webservers können Sie so genannte Default-Dateinamen für Verzeichnisse innerhalb der Document Root festlegen. Beim Apache Webserver geschieht dies in der Konfigurationsdatei *httpd.conf* oder in .htaccess-Dateien mit der Direktive *DirectoryIndex*. Angenommen, in der Konfiguration sind hierbei *index.htm* und *index.html* genannt. Dann kann eine Webseite, die unter diesem Namen in einem Verzeichnis unterhalb der Document Root abgelegt ist, beim HTTP-Aufruf ohne Dateinamen aufgerufen werden.

Verfolgen wir jedoch genauer, was geschieht, wenn ein Webserver beispielsweise eine Anfrage für den URI *http://127.0.0.1/* erhält. Bei dieser Schreibweise ist die gewünschte Datenressource lediglich durch den einzelnen Schrägstrich am Ende referenziert. Wird auch dieser weggelassen, ergänzen ihn Browser bei der Anfrage automatisch. Referenziert wird damit das Startverzeichnis selbst, also die Document Root.

Findet der Apache Webserver in dem Verzeichnis eine Datei, die in der Direktive *DirectoryIndex* genannt ist, so liefert er automatisch diese an den aufrufenden Client aus. Existieren mehrere bei *DirectoryIndex* genannte Dateinamen im Verzeichnis, wird diejenige Datei ausgeliefert, die bei *DirectoryIndex* zuerst genannt ist.

Findet er dagegen keine der bei *DirectoryIndex* angegebenen Default-Dateien, so ermittelt er als Nächstes, ob für das angeforderte Verzeichnis das so genannte *directory browsing* (Verzeichnis-Browsing) erlaubt ist oder nicht. Erlaubt ist es beim Apache Webserver dann, wenn für das Verzeichnis die Option *Indexes* angegeben ist. Letztere lässt sich entweder in der *httpd.conf* zentral angeben oder in verzeichnisspezifischen .htaccess-Dateien (durch die Direktive `Options Indexes`).

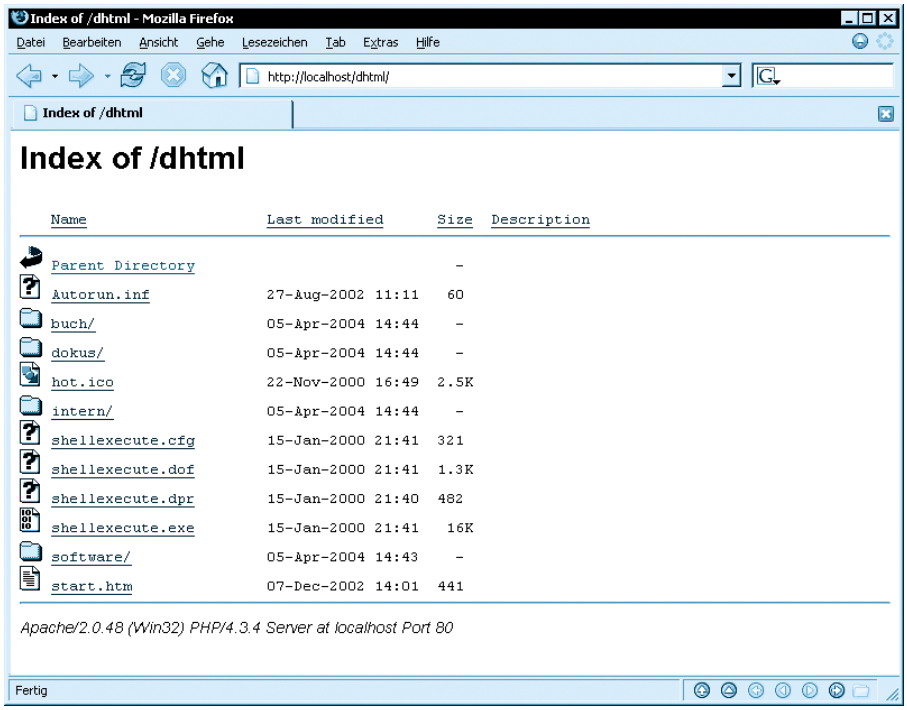


Abbildung 11.1: Typisches Verzeichnis-Listing im Browser

Wenn der Webserver herausfindet, dass kein Verzeichnis-Browsing erlaubt ist, sendet er einen Statuscode 403 (»forbidden«) sowie die entsprechende Fehlerseite an den Browser. Ermittelt er hingegen, dass der Inhalt des angeforderten Verzeichnisses aufgelistet werden darf, so erzeugt er selbst ein HTML-Dokument mit den Einträgen des Verzeichnisses. Das Ergebnis sieht typischerweise so aus wie in obiger Abbildung. Dateien und Unterverzeichnisse sind als Hyperlinks anklickbar. Vor dem Namen steht jeweils ein Symbol, das den Dateityp charakterisiert.

Optionen für Verzeichnis-Browsing

Falls Sie Verzeichnis-Browsing in bestimmten Bereichen Ihres Webangebots sogar explizit wünschen, haben Sie die Möglichkeit, die Optik der Listings in gewissem Umfang an eigene Vorstellungen anzupassen. Dazu bietet der Apache Webserver eigene Direktiven an. Diese können entweder innerhalb der zentralen Konfiguration, also in der *httpd.conf*,

zwischen <Directory> und </Directory> notiert werden oder in verzeichnisspezifischen .htaccess-Dateien.

Notieren Sie auf jeden Fall zunächst folgende Direktive:

```
FancyIndexing On
```

Damit teilen Sie dem Apache Server mit, dass er die Angaben zur individuellen Darstellung von Verzeichnis-Listings beachten soll. Anschließend können Sie nachfolgend beschriebene Möglichkeiten notieren.

```
AddIcon /src/html_icon.gif html htm
AddIcon /src/grafik_icon.gif gif jpg png
AddIcon /src/dir_icon.gif ^^DIRECTORY^^
AddIcon /src/notype_icon.gif ^^BLANKICON^^
```

Mit der Direktive AddIcon können Sie bestimmen, welches Grafiksymbol vor welchen Dateitypen angezeigt werden soll. Geben Sie zuerst den Pfad zur Grafik des gewünschten Symbols an und notieren Sie dahinter jeweils durch Leerzeichen getrennt die Dateiendungen, bei denen dieses Symbol verwendet werden soll. Sonderbedeutung haben die Angaben für ^^DIRECTORY^^ (Eintrag ist ein Unterverzeichnis) und ^^BLANKICON^^ (Eintrag hat keine Dateiendung).

```
DefaultIcon /src/default_icon.gif
```

Mit der Direktive DefaultIcon können Sie eine Symbolgrafik für alle Verzeichniseinträge bestimmen, die sich keinem bestimmten Typ zuordnen lassen.

```
AddAlt "Symbol für HTML" /src/html_icon.gif
```

Mit der Direktive AddAlt können Sie bei AddIcon oder DefaultIcon verwendeten Grafiken einen Alternativtext für den Fall zuordnen, dass die Grafik nicht angezeigt werden kann. Notieren Sie den Alternativtext in hohen Anführungszeichen und dahinter den Pfadnamen der Grafikdatei.

```
AddDescription "Webseite" *.htm *.html
AddDescription "Grafik " *.gif *.jpg *.png
AddDescription "Gepackter Inhalt, downloadbar" *.zip *.gzip *.gz
AddDescription "Projekt-IWAN-Dateien" iwan*
```

Mit der Direktive AddDescription können Sie für bestimmte Dateimuster einen Beschreibungstext bestimmen. Diesen fügt der Apache Server beim Erzeugen des HTML-Codes mit dem Verzeichnisinhalt hinten an jeden Eintrag an. Das Verzeichnislisting wird dadurch für unerfahrene Anwender leichter lesbar. Notieren Sie zuerst den gewünschten Beschreibungstext in hohen Anführungszeichen und dahinter, durch Leerzeichen getrennt, die gewünschten Dateimuster, auf welche die Beschreibung angewendet werden soll.

```
HeaderName /src/dir-listing-header.htm
ReadmeName /src/dir-listing-footer.htm
```

Mit der Direktive `HeaderName` können Sie die Default-Überschrift »Index of [Verzeichnispfad]« umgestalten. Dabei geben Sie eine vollständige HTML-Datei an, deren Inhalt den Kopfbereich individuell gestaltet, z. B. passend zum Layout Ihrer übrigen Webseiten. Die Direktive `ReadmeName` leistet das Gleiche im Fußbereich der erzeugten Seite, also unterhalb des Verzeichnislistings.

```
IndexOrderDefault Descending Name
```

Mit der Direktive `IndexOrderDefault` können Sie die Sortierreihenfolge der Verzeichniseinträge beeinflussen. Beim ersten Parameter bestimmen Sie mit `Descending` eine absteigende Sortierung und mit `Ascending` eine aufsteigende. Beim zweiten Parameter geben Sie an, nach was sortiert werden soll. Möglich sind die Angaben `Name` für Dateinamen, `Date` für Zeitstempel und `Size` für Dateigröße.

```
IndexOptions IconHeight=12 IconWidth=14 IconsAreLinks ScanHTMLTitles
```

Mit der Direktive `IndexOptions` können Sie weitere Details festlegen. Im Beispiel wird bestimmt, dass Icon-Grafiken mit der festen Höhe von 12 Pixel und der festen Breite von 14 Pixel dargestellt werden, dass die Icons ebenso wie der Dateiname als Hyperlink anklickbar sind und dass die Hyperlinks auf HTML-Dateien ein title-Attribut verpasst bekommen, dessen Inhalt aus dem title-Element der verlinkten Datei ermittelt wird.

11.1.3 GET- und POST-Daten

Zum Anfordern von Daten von einem Webserver kennt das HTTP-Protokoll die beiden üblichen Anforderungsmethoden GET und POST. Wenn nichts anderes angegeben wird, wird die GET-Methode verwendet. Wenn also beispielsweise ein Anwender im Browser eine Webadresse eingibt, fordert der Browser den entsprechenden Inhalt beim entsprechenden Webserver mit der GET-Methode an. Die POST-Methode wird eigentlich nur dann verwendet, wenn umfangreichere Daten an den Webserver gesendet werden, die dieser verarbeiten soll. Deshalb sollten Sie bei umfangreicheren HTML-Formularen im method-Attribut des einleitenden `<form>`-Tags am besten die POST-Methode angeben. Voraussetzung ist natürlich, dass ein Script auf dem Server, welches beim action-Attribut zur Datenverarbeitung angegeben ist, den Eingabekanal für POST-Daten abfragt.

Allerdings lassen sich auch mit der GET-Methode Daten an den Server übermitteln. Diese werden als so genannter Query String oder GET-String in den URI eingefügt, und zwar durch ein Fragezeichen getrennt hinter dem Pfadnamen zur gewünschten Datenressource auf dem Server. Ein typischer URI mit GET-String sieht so aus:

http://localhost/index.php?jump=456&view=normal&user= m%FCnz

Der Query- oder GET-String im Beispiel lautet *jump=456&view=normal&user=stefan*. Typischerweise werden im GET-String feldorientierte Daten übergeben, also mehrere Daten der Art »Bezeichner = Wert«. Mehrere solcher Bezeichner-Wert-Paare werden inner-

halb des GET-Strings durch ein kaufmännisches Und (&) voneinander getrennt. Bezeichner und Wert werden durch ein =-Zeichen getrennt. Wichtig ist auch im GET-String, dass alle Zeichen, die nicht im ASCII-Zeichensatz vorkommen oder in URIs Sonderbedeutung haben, so wie in ***Abschnitt 4.8.6*** behandelt notiert werden.

11.2 Server Side Includes (SSI)

Server Side Includes (SSI) sind Code-Fragmente innerhalb von HTML, die der Webserver beim Ausliefern des HTML-Dokuments an einen aufrufenden Browser ausliest, interpretiert und durch gewünschte Inhalte ersetzt. Die Vorteile von SSI liegen auf der Hand:

- HTML-Dateien können schlanker und besser bearbeitbar gemacht werden, indem sich wiederholender Code, etwa für Navigationslinks, einfach in eine externe Datei ausgelagert wird. Diese wird dann mit SSI an der gewünschten Stelle eingebunden. Die sich wiederholenden Inhalte brauchen dann auch nur noch an einer Stelle zentral gepflegt zu werden. Umfangreiche und fehleranfällige Suche-Ersetze-Orgien in HTML-Dateien lassen sich damit vermeiden.
- Es ist keine serverseitige Programmiersprache nötig. Sites, die beispielsweise PHP nur einsetzen, um mit einfachen HTML-Templates zu arbeiten, wären möglicherweise performanter, wenn sie nur mit SSI arbeiten würden, da hierbei keine Interpretation durch einen externen Scriptsprachen-Interpreter nötig ist, der selber CPU-Ressourcen und Arbeitsspeicher benötigt.
- Es besteht jedoch die Möglichkeit, auch Scripts dynamisch einzubinden, die Daten ermitteln und diese genau an der gewünschten Stelle im HTML-Code ausgeben. Das kann beispielsweise auch ein Shellscript sein.

11.2.1 Voraussetzungen

Damit Server Side Includes funktionieren, müssen folgende Voraussetzungen erfüllt sein:

- In der Konfiguration des Webservers muss die Verwendung von SSI ermöglicht werden. Beim Apache Webserver müssen Sie dazu in der ***httpd.conf*** oder in verzeichnisspezifischen .htaccess-Dateien bei der Direktive `Options` den Wert `Includes` hinzufügen. Beispiel:

  ```
  Options Indexes Includes
  ```

  Damit wird Verzeichnis-Browsing und die Verwendung von SSI in Webverzeichnissen erlaubt.

  Denken Sie gegebenenfalls auch daran, bei der Direktive `DirectoryIndex` (ebenfalls in der ***httpd.conf*** oder in verzeichnisspezifischen .htaccess-Dateien) einen Default-Dateinamen wie z. B. `index.shtml` für Verzeichnisse hinzuzufügen.

Ferner müssen Sie bei zwei Zeilen, die in der *httpd.conf* per Default auskommentiert sind, die Kommentarzeichen # am Zeilenanfang entfernen:

```
#AddType text/html .shtml
#AddOutputFilter INCLUDES .shtml
```

- HTML-Dateien, die SSI-Passagen enthalten, müssen die Dateiendung *.shtml* erhalten. Bereits vorhandene HTML-Dokumente, die Sie nachträglich für SSI verwenden wollen, müssen Sie entsprechend umbenennen und gegebenenfalls neu verlinken.

SSI testen

Wenn die Voraussetzungen erfüllt sind, können Sie loslegen. Zum Testen können Sie in einem für den Webserver erreichbaren Verzeichnis, also irgendwo unterhalb der Document Root, eine Datei unter einem Namen wie *ssi-test.shtml* mit folgendem Inhalt ablegen:

Listing 11.1: SHTML-Datei mit Server Side Includes

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>SSI-Test</title>
</head>
<body>
<h1><!--#echo var="SERVER_NAME" --></h1>
<p>Diese Datei hat <!--#fsize virtual="ssi-test.shtml" --> Bytes</p>
</body>
</html>
```

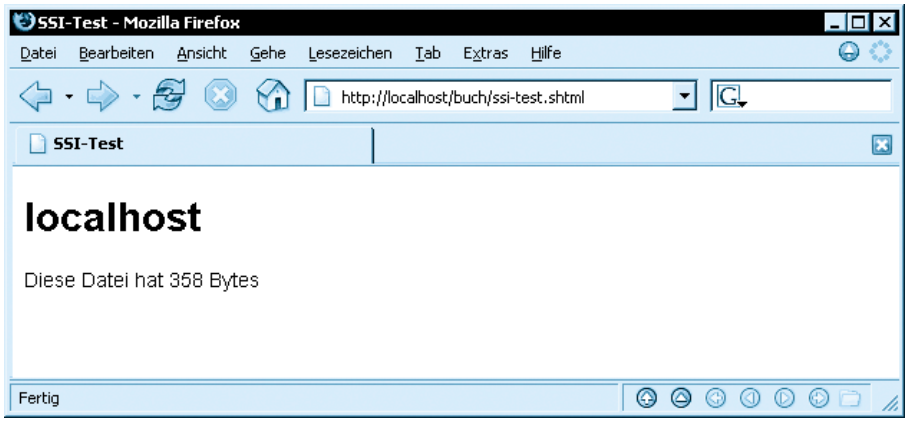


Abbildung 11.2: SHTML-Datei mit Server Side Includes im Browser

Die Beispieldatei gibt in einer h1-Überschrift den Domain- oder Hostnamen des Servers aus und in einem Textabsatz unterhalb davon die Größe der Datei in Byte.

Syntax von Includes

Server Side Includes werden innerhalb von HTML-Kommentaren notiert, also zwischen den Zeichenfolgen `<!--` und `-->`. Unmittelbar hinter der einleitenden Kommentarsequenz muss ein Gatterzeichen (#) folgen. Durch die so entstehende Zeichenfolge `<!--#` weiß der Webserver bei SHTML-Dateien, dass er nachfolgend eine SSI-Anweisung zu erwarten hat. Hinter dem Gatterzeichen wird der Name der gewünschten SSI-Anweisung notiert. Im obigen Quelltextbeispiel werden die Anweisungen `echo` und `fsize` verwendet. Dahinter folgen, abhängig von der angegebenen Anweisung, Angaben, die der Anweisung als übergebene Parameter dienen. Die Parameter folgen dem Syntax-Schema `Parameter-name="Wert"`.

11.2.2 HTML-Templates mit SSI

In einem Beispiel soll gezeigt werden, wie Sie Server Side Includes dazu nutzen können, um wiederholt benötigten HTML-Code auszulagern und dynamisch einzubinden. Wir greifen dazu auf das Praxisbeispiel aus ***Abschnitt 8.2*** zurück. Dort ging es um die Realisierung eines Explorer-Menüs mit JavaScript/DOM. Nachteil der dort realisierten Lösung war, dass die gesamte Navigation in jeder einzelnen Seite wiederholt werden musste. Viele Seitenanbieter greifen zu Frames, um das Problem der umständlichen Wartbarkeit einer solchen Navigation zu umgehen. Doch viel eleganter geht es mit SSI. Eine HTML-Seite des Explorer-Menü-Beispiels könnte damit so aussehen:

Listing 11.2: SHTML-Datei mit eingebundener Datei für Navigation

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Explorer-Navigation</title>
<link rel="stylesheet" type="text/css" href="explorer.css">
<script type="text/javascript" src="explorer.js"></script>
</script>
</head>
<body onLoad="init_nav()">
<div id="navigation">
<!--#include file="explorer.inc" -->
</div>
<div id="content">
<h1>FAUNA-WELT</h1>
</div>
</body>
</html>
```

Das Listing entspricht Listing 8.3. Nur der umfangreiche Code-Teil mit den verschachtelten Listen zur Navigation ist verschwunden. Innerhalb des div-Bereichs mit dem id-Namen `navigation` ist stattdessen eine SSI-Anweisung notiert. Mit der Anweisung `include` können Sie Inhalte anderer Dateien an der aktuellen Stelle im HTML-Code einfügen. In unserem Beispiel binden wir die Datei `explorer.inc` ein. In eine Datei dieses Namens haben wir den HTML-Code mit der verschachtelten Aufzählungsliste und den Navigationslinks ausgelagert. Um die Navigation zu ändern, muss also nur noch diese Datei geändert werden. Die inhaltstragenden Dateien bleiben davon unberührt.

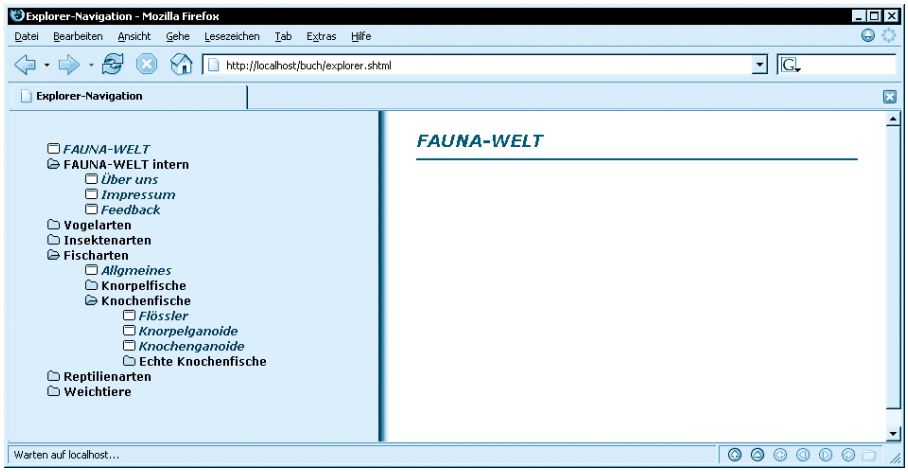


Abbildung 11.3: Explorer-Navigation, ausgelagert in eigene Datei und eingebunden über SSI

Bei der include-Anweisung können Sie als Parameter entweder `file="[Pfad/]Datei"` angeben oder `file="[Pfad/]Datei"`.

Bei relativen Angaben wie im Beispiel ist es egal, welche Variante Sie angeben. Wenn Sie die einzubindende Datei jedoch mit einem absoluten Pfadnamen referenzieren, bedeutet der virtual-Parameter, dass der absolute Pfadname ab Document Root verstanden wird, also ab dem Startverzeichnis der Webseiten, und der file-Parameter bedeutet, dass der Pfadname ab dem Wurzelverzeichnis des Server-Rechners verstanden wird. Wenn also eine einzubindende Datei im Dateisystem unter */var/www/inc/datei.txt* abgespeichert ist, könnte sie mit `include file="/var/www/inc/datei.txt"` eingebunden werden. Angenommen, das Startverzeichnis für Webseiten ist */var/www*, dann könnte die gleiche Datei auch mit `include virtual="/inc/datei.txt"` eingebunden werden.

11.2.3 Variablenausgabe mit SSI

Eine weitere nützliche Eigenschaft von Server Side Includes ist die Ausgabe variabler Daten. Auch das kann helfen, den Verwaltungsaufwand für HTML-Dateien zu verringern. Nachfolgendes Beispiel zeigt einen ausgedehnten Footer-Bereich mit dynamischen Informationen:

Listing 11.3: SHTML-Datei mit Variablenausgaben

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<meta name="DC.Identifier" content="">
<title>Variablen</title>
</head>
<body style="margin-left:30px; margin-right:30px;">
<h1>Blindtext</h1>
<p>Lorem ipsum dolor sit amet, ...</p>
<p style="border-top:solid 1px blue; margin-top:20px;
          padding-top:10px; font-size:90%">
<b>URI dieses Dokuments:</b>
http://<!--#echo var="SERVER_NAME" -->
<!--#echo var="DOCUMENT_URI" --><br>
<!--#config timefmt="%d.%m.%Y, %H.%M Uhr" -->
<b>Letzte Änderung:</b> <!--#echo var="LAST_MODIFIED" --><br>
<!--#config sizefmt="bytes" -->
<b>Seitenumfang:</b> <!--#fsize virtual="variablen.shtml" -->
Bytes<br>
<b>Serviert von:</b> <!--#echo var="SERVER_SOFTWARE" -->
</p>
</body>
</html>
```

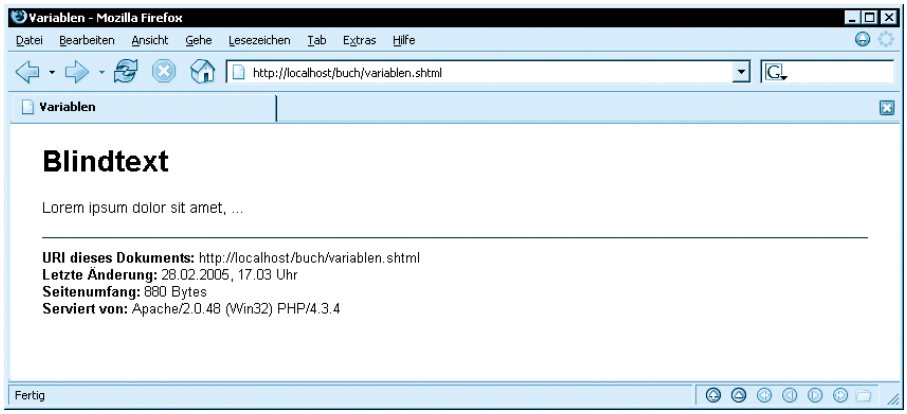


Abbildung 11.4: SHTML-Datei mit Variablenausgabe

Zu den SSI-eigenen »Variablenanweisungen« gehören `fsize` (zur Ausgabe der Größe einer beliebigen Datei) und `flastmod` (zur Ausgabe des Zeitstempels einer beliebigen Datei). Da sowohl Größenangaben als auch Zeitangaben verschiedene typische Ausgabeformate kennen, steht mit `config` eine Anweisung zur Verfügung, die das Formatieren von solchen Angaben ermöglicht. Besonders bei Zeitangaben steht eine flexible Syntax zur Verfügung.

Wie ein Zeitstempel dargestellt werden soll, wird mit `config timefmt="[Formatstring]"` festgelegt. Diese SSI-Anweisung bewirkt selbst keine Ausgabe, sondern nur eine Definition für nachfolgende Ausgaben. Nachfolgende Tabelle listet die Prozentbezeichner auf, die im Formatstring vorkommen dürfen:

Bezeichner	Bedeutung
%a	Wochentag abgekürzt, z. B. »Mon«
%A	Wochentag ausgeschrieben, z. B. »Monday«
%b	Monatsname abgekürzt, z. B. »Feb«
%B	Monatsname ausgeschrieben, z. B. »Febuary«
d	Tag im Monat zweistellig, z. B. 03
%e	Tag im Monat, z. B. 3
%H	Uhrzeit 24-stündig, z. B. 17
%I	Uhrzeit 12-stündig, z. B. 5
%j	Julianisches Datum, z. B. 123
%m	Monat als Zahl, z. B. 02
%M	Minuten, z. B. 07
%p	Bei 12-stündiger Uhrzeit AM/PM als Zusatz
%S	Sekunden, z. B. 08
%U	Kalenderwoche, z. B. 34
%w	n-ter Tag der Woche, z. B. 1
%y	Jahr zweistellig, z. B. 05
%Y	Jahr vierstellig, z. B. 2005
%Z	Zeitzonenangabe

Tabelle 11.1: Prozentbezeichner für Formatstring bei SSI-Anweisung config timefmt

Darüber hinaus können Sie über `echo var="[Variablenname]"` Inhalte von Variablen ausgeben. Das betrifft vor allem die so genannten CGI-Umgebungsvariablen, die durch den Webserver zur Verfügung gestellt werden. Nachfolgend eine Auflistung für SSI nützlicher Variablen:

Variable	Bedeutung
DOCUMENT_ROOT	Pfadname des Startverzeichnisses für Webseiten.
HTTP_COOKIE	Eventuelle vom Browser gesendete Cookies.
HTTP_REFERER	URI der Seite, von der aus die aktuelle Seite aufgerufen wurde.
HTTP_USER_AGENT	Produktkennung, die der aufrufende Browser gesendet hat.
REMOTE_HOST	IP-Adresse oder Hostname des aufrufenden Hostrechners. Das kann der Rechner des Clients sein, aber etwa auch ein Proxy-Server.

Tabelle 11.2: Wichtige Umgebungsvariablen für echo var

Variable	Bedeutung
REMOTE_USER	Enthält z. B. bei htaccess-geschützten Seiten den Benutzernamen eines angemeldeten Benutzers.
SERVER_ADMIN	Mailadresse des in der *httpd.conf* eingetragenen Webmasters.
SERVER_NAME	Domain- oder Hostname des Server-Rechners.
SERVER_SOFTWARE	Produktkennung des Webservers.

Tabelle 11.2: Wichtige Umgebungsvariablen für echo var (Fortsetzung)

11.2.4 Script-Einbindung mit SSI

Eine der mächtigsten Möglichkeiten von Server Side Includes ist das dynamische Einbinden von Scripts oder Programmen aller Art. Wichtig ist nur, dass die Scripts oder Programme einen Output auf die Standardausgabe schreiben. Dieser wird im SHTML-Dokument an der gewünschten Stelle eingefügt.

Shellscript einbinden: Anzahl aktueller Webserver-Prozesse auf dem Server

Ein beliebtes Ausgabedatum auf vielen Websites ist die »Anzahl Besucher online«. Die dabei ausgegebenen Zahlen sind allerdings in vielen Fällen mit Vorsicht zu genießen, da HTTP nicht wirklich in der Lage ist, eine Verbindung zu halten. Webserver wie Apache sind zwar in der Lage, Server-Prozesse im Speicher zu halten, damit diese eine Weile lang auf erneute Anfragen des gleichen Clients warten. Doch der Webserver kann zu keinem Zeitpunkt überprüfen, ob der Client überhaupt noch online ist oder nicht längst auf anderen Webangeboten surft. Wenn man jedoch statt von »Anzahl Besucher online« von »Anzahl aktueller Webserver-Prozesse« spricht, dann ist auch der ermittelte Wert korrekt. In unserem Beispiel ermitteln wir diesen Wert mit einem kleinen Shellscript, das wir *count-httpd* nennen. Sein Inhalt lautet:

```
#!/bin/bash
ps -A | awk '{print $4}' | grep httpd | wc -l | tr -d '\n'
```

Zunächst werden mit dem ps-Kommando alle aktuellen Prozesse aller Benutzer auf dem Server aufgelistet. Die dadurch erzeugte Liste wird über eine Pipe an das awk-Kommando übergeben, das die vierte Spalte davon herausfiltert. In der vierten Spalte stehen die Kommando- oder Programmnamen, denen ein Prozess zugeordnet ist. Die so auf Programmnamen reduzierte Liste wird wieder per Pipe an das grep-Kommando übergeben. Dieses reduziert die Liste auf Einträge, in denen `httpd` vorkommt – der Programmname des Webservers. Die Liste enthält anschließend lauter Zeilen, in denen `httpd` steht. Die gewünschte Anzahl ermitteln wir durch Zählen dieser Zeilen. Deshalb wird die Liste per Pipe an das wc-Kommando übergeben, das mit der Option `-l` die Zeilen zählt. Am Ende wird mit einem tr-Kommando noch das abschließende Zeilenumbruchzeichen entfernt und übrig bleibt die nackte Anzahl von httpd-Prozessen. Diese gibt das Script am Ende aus.

Im SHTML-Dokument kann dieses Shellscript wie folgt aufgerufen werden:

```
<!--#exec cmd="/usr/bin/count-http" -->
```

Die exec-Anweisung führt Programme, Scripts oder CGI-Scripts auf dem Server-Rechner aus. In unserem Fall ist es ein Shellscript, das außerhalb der Document Root des Webservers abgelegt ist und nicht den Status eines CGI-Scripts hat. Deshalb verwenden wir den Parameter `cmd` und weisen ihm den vollständigen Pfadnamen des Ablageorts des Shellscripts zu. Der Pfadname muss natürlich angepasst werden, je nachdem, wo das Shellscript abgelegt ist.

Es wäre übrigens ebenso möglich, das Kommando direkt in SSI einzubauen, da es sich in unserem Fall um nur ein einziges, wenn auch zusammengesetztes Kommando handelt. Die SSI-Anweisung würde dann so aussehen:

```
<!--#exec cmd=" ps -A | awk '{print $4}' | grep httpd | wc -l | tr -d '\n'" -->
```

CGI-Script einbinden: deutschsprachige Ausgabe von Datum und Uhrzeit

Das nachfolgende Beispiel zeigt den Quelltext eines Perl-Scripts. Perl ist die verbreitetste Sprache für klassische CGI-Scripts. Das nachfolgende Beispielscript erzeugt einfach ein paar deutschsprachige Datums- und Zeitformate für den aktuellen Server-Zeitpunkt:

Listing 11.4: Perl-Script für Datums- und Uhrzeitausgabe

```
#!/usr/bin/perl -w
use strict;
use CGI::Carp qw(fatalsToBrowser);
my ($ss, $ii, $hh, $dd, $mm, $yyyy, $dow, $doy, $sumtime)
   = localtime(time);
my $CTIME_String = localtime(time);
$mm+=1;
$doy+=1;
$mm = $mm < 10 ? $mm = "0".$mm : $mm;
$dd = $dd < 10 ? $dd = "0".$dd : $dd;
$hh = $hh < 10 ? $hh = "0".$hh : $hh;
$ii = $ii < 10 ? $ii = "0".$ii : $ii;
$ss = $ss < 10 ? $ss = "0".$ss : $ss;
$yyyy+=1900;
my @Wochentage = ("Sonntag","Montag","Dienstag","Mittwoch",
                  "Donnerstag","Freitag","Samstag");
my @Monatsnamen = ("","Januar","Februar","M&auml;rz","April",
                   "Mai","Juni","Juli","August","September",
                      "Oktober","November","Dezember");
print "Content-type: text/html\n\n";

print "Datum (Variante 1): $dowe[$dow], ";
print "der $dd.$mm.$yyyy<br>\n";
print "Datum (Variante 2): $dowe[$dow], ";
print "der $dd. $mmsnamen[$mm] $yyyy<br>\n";
print "Uhrzeit (Variante 1): $hh.$ii Uhr<br>\n";
```

```
print "Uhrzeit (Variante 2): $hh.$ii:$ss Uhr<br>\n";
print "Der heutige Tag ist der $doy. Tag im Jahr<br>\n";
print "Als CTIME-Ausgabe: $CTIME_String<br>\n";
$sumtime ? print "Es ist Sommerzeit\n" :
           print "Es ist Normalzeit (keine Sommerzeit)\n";
```

Angenommen, dieses Script wird im CGI-Verzeichnis (typischerweise */cgi-bin*) unter dem Namen *showtime.pl* abgelegt. Mit folgender SSI-Anweisung kann es dann in SHTML eingebunden werden:

```
<!--#exec cgi="/cgi-bin/showtime.pl" -->
```



Abbildung 11.5: In SHTML-Datei eingebundenes CGI-Script showtime.pl

Auch CGI-Scripts werden über die exec-Anweisung eingebunden. Im Unterschied zu Kommandos oder Programmen, die mit auf den Server-Rechner bezogenen Pfadnamen referenziert werden, muss in diesem Fall jedoch der Parameter `cgi` verwendet werden. Referenziert wird dann der Webpfad zum CGI-Script.

11.3 Die CGI-Schnittstelle

Wenn ein serverseitiges Script angefordert wird, so muss der Webserver wissen, dass er es ausführen soll. Dazu bietet etwa der Apache Webserver zwei grundsätzliche Varianten an. Die eine besteht darin, den zugehörigen Script-Interpreter als Apache-Modul einzubinden. Ein entsprechendes, die API-Schnittstelle des Apache bedienendes Modul muss natürlich existieren. So wird es in der heutigen Praxis mit PHP in der Regel gemacht.

Die zweite und ältere Variante ist die so genannte CGI-Schnittstelle. CGI steht für *Common Gateway Interface*. Die CGI-Schnittstelle besteht darin, dass bestimmte Verzeichnisse als CGI-Verzeichnisse definiert werden. In solchen Verzeichnissen (und in Unterverzeichnisstrukturen davon) dürfen Scripts abgelegt werden, die der Webserver als für ihn ausführbar akzeptiert. In welcher Programmiersprache die Scripts geschrieben sind, ist dem

Webserver egal. Wichtig ist nur, dass er den Script-Interpreter ermitteln kann, z.B. über die Dateiendung der Scripts oder über deren Shebang-Zeile. Es kann sich natürlich auch um kompilierte Programme handeln, die ohne Interpreter direkt ausführbar sind.

Ein CGI-Script kann serverseitig Daten verarbeiten, mit Datenbanken kommunizieren usw. Am Ende sollte es jedoch stets Daten auf die Standardausgabe schreiben. Da es vom Webserver ausgeführt wird, kann dieser die Ausgaben des Scripts auffangen und zum aufrufenden Browser senden. Bei seiner Antwort an den Browser beschränkt sich der Webserver im HTTP-Protokoll auf das reine HTTP-Kommando ohne zusätzliche HTTP-Header. Diese muss das CGI-Script selber erzeugen. Der Vorteil davon ist eine große Flexibilität. CGI-Scripts erzeugen in der Regel HTML-Code, aber sie können ebenso gut Binärdaten einer dynamisch erzeugten Grafik oder andere Daten ausgeben. Wichtig ist, dass die ausgegebenen HTTP-Header und die ausgegebenen Daten vom aufrufenden Browser, der die Daten vom Webserver gesendet bekommt, verarbeitbar sind.

CGI-Scripts erzeugen jedoch nicht nur Ausgaben, sondern verarbeiten meistens auch Input. Den Input erhalten sie beim Aufruf durch den Browser über die beiden möglichen Eingabekanäle GET oder POST. Bei GET werden die Input-Daten bekanntlich als Query-String im URI mit übertragen. Bei POST werden sie über den Standardeingabekanal des Server-Rechners zur Verfügung gestellt.

11.3.1 CGI-Konfiguration

Welche Verzeichnisse als CGI-Verzeichnisse gelten, wird in der Konfiguration des Webservers festgelegt. Beim Apache Webserver kann dies wahlweise in der zentralen Konfiguration der Datei *httpd.conf* geschehen oder in verzeichnisspezifischen .htaccess-Dateien.

Zentrales CGI-Verzeichnis einrichten

Um in der *httpd.conf* ein zentrales CGI-Verzeichnis zu bestimmen, müssen Sie die Direktive *ScriptAlias* verwenden. Suchen Sie in der *httpd.conf* nach einer entsprechenden Direktive. Per Default ist bereits eine Direktive für CGI-Scripts eingetragen, jedoch auskommentiert. Entfernen Sie die Auskommentierung, also das führende Gatterzeichen (#). Ein typischer Eintrag lautet:

```
ScriptAlias /cgi-bin/ "/var/www-scripts/cgi-bin/"
```

Die Direktive erwartet zwei Parameter. Im ersten Parameter wird der absolute Pfadname aus URI-Sicht angegeben und im zweiten Parameter der absolute Pfadname aus Sicht des Dateisystems am Server-Rechner. Der zweite Parameter steht in hohen Anführungszeichen. Beachten Sie, dass beide Pfadnamen am Ende noch mal einen Schrägstrich enthalten müssen!

Das CGI-Verzeichnis muss übrigens nicht innerhalb der Document Root liegen. Angenommen, im obigen Beispiel wurde `DocumentRoot /var/www` festgelegt. Dann befindet sich das angegebene CGI-Verzeichnis parallel dazu auf gleicher Verzeichnisebene, aber nicht

unterhalb davon. Der Webserver löst Client-Anfragen wie *http://server/cgi-bin/test.cgi* jedoch auf Grund der Zuordnung bei der `ScriptAlias` korrekt auf.

Damit CGI-Scripts funktionieren, muss beim Apache ferner das CGI-Modul geladen werden. Dazu darf in der *httpd.conf* der Eintrag, der mit *LoadModule cgi_module ...* beginnt, nicht auskommentiert sein. Sollte dies der Fall sein, müssen Sie das Kommentargatterzeichen entfernen.

In jedem Fall ist es ferner sinnvoll, in folgender Zeile einer *httpd.conf* das Kommentarzeichen zu entfernen:

```
AddHandler cgi-script .cgi
```

Damit dürfen CGI-Scripts die Dateiendung *.cgi* haben.

Dezentral CGI-Verzeichnisse einrichten

Die zweite Möglichkeit besteht darin, mehrere CGI-Verzeichnisse einzurichten. Das kann entweder innerhalb einer Direktivengruppe zwischen *<Directory>* und *</Directory>* geschehen oder innerhalb des gewünschten Verzeichnisses selbst in einer .htaccess-Datei. In letzterem Fall muss sich das Verzeichnis allerdings innerhalb der Document Root befinden. In beiden Fällen können Sie folgende Direktiven notieren:

```
Options +ExecCGI
AddHandler cgi-script .cgi
```

CGI-Scripts in allen Webverzeichnissen erlauben

Es ist auch möglich, CGI-Scripts in allen Verzeichnissen unterhalb der Document Root zu erlauben. Dazu müssen Sie in der *httpd.conf* die Direktivengruppe *<Directory>...</Directory>* für das Verzeichnis suchen, das als Document Root definiert wurde. Wenn Sie dort bei der Direktive `Options` den Wert `ExecCGI` mit angeben, können CGI-Scripts in allen Webverzeichnissen abgelegt werden.

Voraussetzungen für CGI-Scripts

Auch die CGI-Scripts selber müssen Voraussetzungen erfüllen, damit die CGI-Schnittstelle tatsächlich bedient werden kann:

- Das Script muss in einem als CGI-Verzeichnis konfigurierten Verzeichnis abgelegt werden.
- Sowohl das CGI-Verzeichnis als auch das CGI-Script müssen auf einem Linux-/Unix-System für den Webserver ausführbar sein. Da der Webserver als ganz normaler Benutzer vom Typ »Rest der Welt« auf dem System läuft, müssen die Ausführrechte entsprechend gesetzt werden. Am sinnvollsten ist es, wenn sowohl Dateieigentümer als auch seine Primärgruppe und der Rest der Welt Ausführrechte erhalten. Das Schreibrecht sollte jedoch dem Eigentümer oder maximal noch seiner Gruppe vor-

behalten sein. In der Regel ist `chmod 755` für das CGI-Verzeichnis und die Scripts darin eine sinnvolle Wahl.

- Das Script muss zumindest einen HTTP-Header für das Feld `CONTENT-TYPE` ausgeben. Im Beispiel in ***Abschnitt 11.3.3*** gehen wir darauf noch näher ein.

11.3.2 CGI-Aufrufmöglichkeiten in HTML

In HTML gibt es verschiedene Orte, an denen ein CGI-Script aufgerufen werden kann:

- ***Über ein Formular:*** Beispiel: `<form action="/cgi-bin/guestbook.pl" method="post">`. Durch Absenden des Formulars sendet der Browser an den Webserver über HTTP im Beispiel eine POST-Anforderung für die Datenressource ***/cgi-bin/guestbook.pl***. Der Webserver erkennt, dass es sich um ein CGI-Script handelt, und führt es aus. Die vom Anwender eingegebenen oder ausgewählten Daten stehen dem CGI-Script zur Verarbeitung zur Verfügung – im obigen Beispiel über den POST-Eingabekanal. Formulare sind auch die einzige Möglichkeit in HTML, CGI-Scripts über den POST-Eingabekanal mit Daten zu versorgen.
- ***Über Verweise:*** Es genügt, als URI des Verweisziels das ausführbare CGI-Script anzugeben, z. B. `<a href="/cgi-bin/statistik.pl">Statistik</a>`. Dies ist vor allem sinnvoll für CGI-Scripts, die keinen Input vom Anwender benötigen, sondern lediglich feste Datenausgaben erzeugen, zum Beispiel für ein CGI-Script, das aktuelle Zugriffsstatistiken für Webseiten ausgibt.
- ***Über eine Grafikreferenz:*** Auch dabei genügt es, als URI in der src-Angabe des `<img>`-Tags das ausführbare CGI-Script anzugeben, z. B. `<img src="/cgi-bin/counter.pl">`. Dabei muss das CGI-Script allerdings eine Grafikdatei im GIF- oder JPEG-Format an den Webbrowser zurücksenden. Die meisten grafischen Zugriffszähler basieren auf diesem Prinzip.
- ***Über Server Side Includes:*** Über eine SSI-Anweisung in einer SHTML-Datei wie z. B. `<!-- #exec cgi="/cgi-bin/counter.pl" -->` kann ein CGI-Script automatisch beim Aufruf der SHTML-Datei ausgeführt werden. Das ist sehr praktisch, um mithilfe eines CGI-Scripts dynamische Information in Textform in eine HTML-Datei einzubinden.

11.3.3 Beispiel eines Form-Mailers in Perl

In einem kleinen Praxisbeispiel möchten wir den typischen Zusammenhang zwischen HTML und CGI-Script demonstrieren. Das Beispiel realisiert einen so genannten Form-Mailer. Das ist die ideale Anwendung, wenn vom Anwender eingegebene Formulardaten per Mail an den Site-Betreiber gesendet werden sollen. Der Anwender erhält nach Absenden des Formulars eine freundliche Danke-fürs-Ausfüllen-Seite am Bildschirm. Seine eingegebenen Daten werden zu einer ordentlich formatierten E-Mail aufbereitet und an die gewünschte Mailadresse des Site-Betreibers gesendet.

Das Beispiel ist in Perl geschrieben, da Perl immer noch die verbreitetste Sprache für klassische CGI-Scripts ist. Auf den meisten bei Hosting-Providern zu mietenden Root-Servern

ist Perl auch schon vorinstalliert, so dass Sie den Perl-Interpreter nicht noch installieren müssen. Sollte Perl nicht installiert sein, was Sie am Shell-Prompt sehr einfach durch das Kommando `which perl` herausbekommen können, dann müssen Sie es downloaden und auf dem Server-Rechner installieren. Perl kann von *http://www.perl.com/* bezogen werden und ist ebenso wie Linux, Apache oder PHP OpenSource. Lesen Sie zur Installation auf einem Server-Rechner mit Linux-System auch ***Abschnitt 10.2.8*** in diesem Buch.

Für unser Form-Mail-Beispiel benötigen wir drei Dateien:

- Eine HTML-Datei mit dem Formular
- Eine HTML-Datei, die als Danke-Seite fungiert
- Das Perl-Script, das die Daten als CGI-Script verarbeitet und die Danke-Seite sendet

Teil 1: HTML-Datei mit Formular

Im Beispiel soll ein typisches Feedback-Formular realisiert werden. Die Datei mit dem Feedback-Formular ist eine gewöhnliche statische HTML-Datei. Der Quelltext lautet:

Listing 11.5: HTML-Datei mit Feedback-Formular

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Feedback</title>
<link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<h1>Feedback</h1>
<form action="/cgi-bin/form-mail.cgi" method="post">
<input type="hidden" name="return" value="/danke.htm">
<input type="hidden" name="delimiter" value=": ">
<input type="hidden" name="subject" value="Feedback-Formular">
<fieldset><legend>Hinweis:</legend>
<div class="explain">Felder mit Stern * müssen ausgefüllt
werden.</div>
</fieldset>
</td>
</tr><tr>
<td><fieldset><legend>Daten:</legend>
<div>Vorname:<br>
<input type="text" name="Vorname" class="text"
       style="width:400px">
</div>
<div>Zuname:<br>
<input type="text" name="Zuname" class="text"
       style="width:400px">
</div>
<div>E-Mail: <b>*</b><br>
```

```
<input type="text" name="Mail" class="text"
       style="width:400px">
</div>
<div>Feedback: <b>*</b><br>
<textarea name="Text" class="text"
       style="width:400px; height:300px"></textarea>
</div>
</fieldset>
<fieldset><legend>Daten:</legend>
<p>
<input type="submit" class="button" value="Absenden">
<input type="reset" class="button" value="Löschen">
</p>
</fieldset>
</form>
</body>
</html>
```

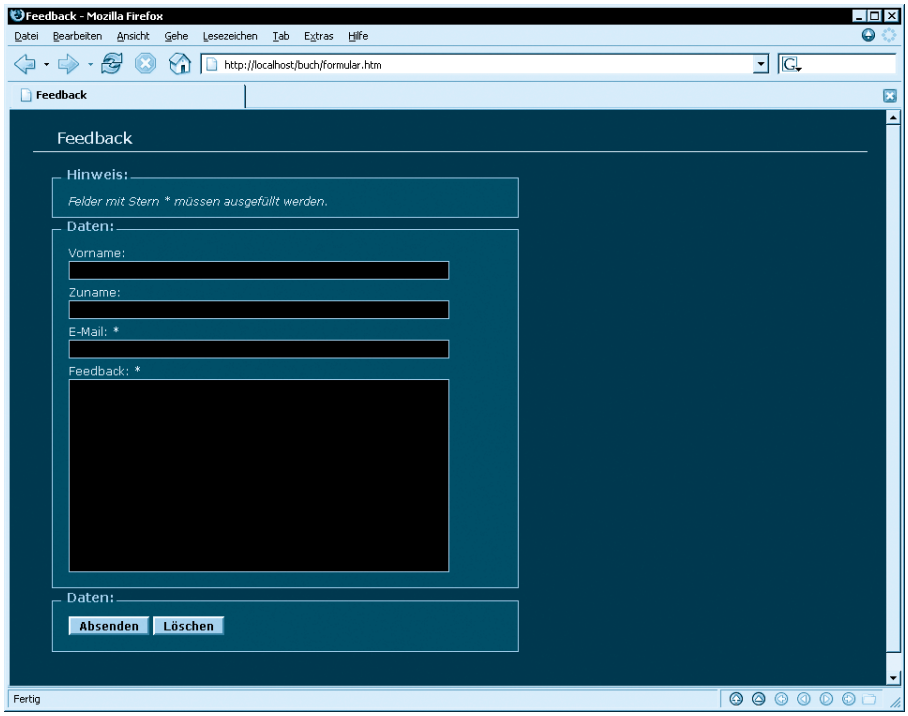


Abbildung 11.6: Feedback-Formular im Browser

Layout und Optik der Seite und des Formulars werden in der eingebundenen Datei *styles.css* definiert. Darauf gehen wir an dieser Stelle nicht weiter ein. Wichtig ist im aktuellen Zusammenhang nur das Formular selbst. Im einleitenden `<form>`-Tag wird im action-Attribut das CGI-Script aufgerufen, das die Daten nach Absenden des Formulars verarbeitet.

Die sichtbaren Formularfelder sind frei benennbar, Art und Anzahl der Felder ebenfalls. Denn unser Form-Mail-Script ist so geschrieben, dass es alle Arten von Formularen verarbeitet. Der Form-Mailer erwartet jedoch bestimmte hidden-Felder. Die Werte der name-Attribute dieser Felder sind durch das Form-Mail-Script vorgegeben. Die Werte (value) müssen Sie dagegen im HTML-Code anpassen.

```
<input type="hidden" name="return" value="/danke.htm">
```

Damit geben Sie den URI der Seite an, die nach erfolgreichem Versenden des Formulars und Verarbeitung an den Seitenbesucher gesendet werden soll.

```
<input type="hidden" name="delimiter" value=": ">
```

Damit geben Sie an, wie Feldnamen und Feldinhalte in der Mail, die versendet wird, dargestellt werden sollen. Im Beispiel stehen in den Mails, die das CGI-Script erzeugt, Zeilen wie:

Feldname: vom Benutzer eingegebener Wert

```
<input type="hidden" name="subject" value="Feedback-Formular">
```

Damit bestimmen Sie das Subject der Mails, die vom CGI-Script versendet werden.

Teil 2: Danke-Seite

Bevor wir auf das Form-Mail-Script kommen, noch kurz der Beispielcode für die Danke-Seite. Hierbei ist aus technischer Sicht nichts Besonderes zu beachten. Es kann sich um eine normale statische HTML-Datei handeln. Die Seite sollte den inhaltlichen Bezug zum gerade abgesendeten Formular herstellen. In unserem Beispiel lautet der Quelltext:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Danke für Ihr Feedback</title>
<link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<h1>Danke für Ihr Feedback</h1>
<p>Ihre eingegebenen Daten wurden uns zugesendet.
Wir werden uns mit Ihnen in Verbindung setzen.</p>
</body>
</html>
```

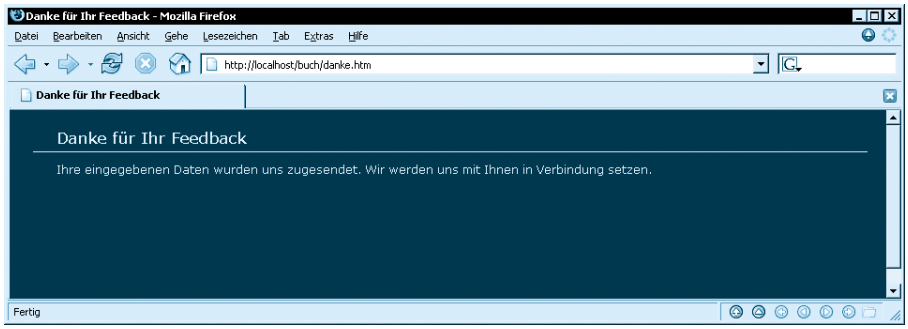


Abbildung 11.7: Danke-Seite für Form-Mailer

Teil 3: Perl-Script für Form-Mailer

Das CGI-Script ist in unserem Beispiel der serverseitige Vermittler zwischen der Seite mit dem Feedback-Formular und der Danke-Seite. Nach dem Absenden des Formulars sendet der Browser des Anwenders einen POST-Request an den Webserver. Einen POST-Request deshalb, weil im HTML-Formular als Übertragungsmethode `type="hidden"` angegeben wurde.

Daraus ergeben sich folgende Aufgaben für das CGI-Script:

- Es muss die Formulardaten über den POST-Eingabekanal einlesen.
- Es muss aus den Formulardaten eine E-Mail zusammenstellen.
- Es muss die E-Mail an die gewünschte Mailadresse versenden.
- Es muss die Danke-Seite ausgeben. Der Webserver sendet diese dann an den aufrufenden Browser.

Der Quelltext unseres Perl-Scripts lautet:

Listing 11.6: Perl-Script für einen einfachen Form-Mailer

```
#!/usr/bin/perl

$mailto = "IhrName@irgendwo";
$Sendmail_Prog = "/usr/lib/sendmail";
use CGI;
$query = new CGI;
@names = $query->param;
# interne Daten aus den erwarteten hidden-Feldern auslesen:
$delimiter = $query->param('delimiter');  # ---> Begrenzerzeichen
$returnhtml = $query->param('return');    # ---> Dankeseite
$subject = $query->param('subject');      # ---> E-Mail-Subject
# Text der E-Mail aus den Formulardaten ermitteln:
$mailtext = "";
foreach(@names) {
   $name = $_;
   @values = "";
   @values = $query->param($name);
```

```
    if($name ne "mailto" && $name ne "return"
       && $name ne "subject" && $name ne "delimiter") {
       foreach $value (@values) {
          $mailtext = $mailtext.$name;
          $mailtext = $mailtext.$delimiter;
          $mailtext = $mailtext.$value."\n";
       }
    }
}
# E-Mail versenden:
open(MAIL,"|$Sendmail_Prog -t") ||
     print STDERR "Mailprogramm nicht gestartet werden\n";
print MAIL "To: $mailto\n";
print MAIL "Subject: $subject\n\n";
print MAIL "$mailtext\n";
close(MAIL);
# Dankeseite an Browser senden:
print "Location: $returnhtml\n\n";
```

In der Shebang-Zeile wird der Pfad zum Perl-Interpreter angegeben. Üblicherweise ist die ausführbare Datei des Perl-Interpreters auf Linux-Systemen unter */usr/bin/perl* erreichbar. Andernfalls müssen Sie hier den Pfadnamen anpassen. Aus Sicht von Perl ist das eine Kommentarzeile, da sie mit einem Gatterzeichen (#) beginnt.

Im Script-Code selber müssen Sie dann noch die erste Zeile anpassen:

```
$mailto = "IhrName@irgendwo";
```

Als Mailadresse ist hier diejenige Adresse einzusetzen, an die Daten abgesendeter Formulare gesendet werden sollen. Es hat übrigens einen sehr wichtigen Grund, dass wir diese Adresse erst im Form-Mail-Script konfigurieren und nicht schon in einem hidden-Feld im HTML-Formular. Der Grund ist, dass wir durch die Definition in HTML ein öffentlich aufrufbares CGI-Script hätten, dass beliebige Inhalte an beliebige E-Mail-Adressen senden kann, wenn es nur per POST-Methode entsprechenden Input erhält. Solch ein Script würde einen erheblichen Risikofaktor darstellen, da es von Massenspammern wie ein »offener Mailserver« missbraucht werden könnte. Dadurch, dass das Script nun aber alle Mails an die angegebene Adresse sendet, wird es für Spammer uninteressant.

Auf weitere Details im Perl-Code wollen wir nicht eingehen, da Perl nicht Thema dieses Buches ist. Die auskommentierten Zeilen im Code weisen darauf hin, was in den jeweils nachfolgenden Code-Abschnitten passiert. Die einleitend genannten Script-Aufgaben sind wiedererkennbar.

Damit das Form-Mail-Script jedoch funktioniert, müssen noch zwei wichtige Voraussetzungen erfüllt sein:

- Das CGI-Modul für Perl muss installiert sein. Seit Perl-Version 5.004 gehört es zu den Standardmodulen, die mit jeder Perl-Installation automatisch mit installiert werden. Da diese Perl-Version schon seit Jahren überholt ist, dürfte das CGI-Modul jedoch mit

ziemlicher Sicherheit installiert sein. Im Zweifelsfall können Sie am Shell-Prompt eingeben: `perl -v`. Damit gibt der installierte Perl-Interpreter seine Versionsnummer aus.

- Das Programm ***sendmail*** muss installiert sein, und zwar unter ***/usr/lib/sendmail***. Dieses Programm ist für das Versenden von Mails verantwortlich und wird innerhalb des Perl-Scripts aufgerufen. Geben Sie im Zweifelsfall am Shell-Prompt des Servers `which sendmail` ein, um zu überprüfen, ob und wo das Programm installiert ist. Bei Linux-Installationen gehört ***sendmail*** jedoch meistens zur Basis-Software.

In Zusammenhang mit PHP werden wir ebenfalls noch eine Form-Mail-Lösung vorstellen.

11.4 Geschützte Webseiten

Gründe dafür, Webinhalte nur bestimmten, registrierten Benutzern zugänglich zu machen, gibt es viele, und das sind wesentlich mehr Gründe, als nur der einiger Anbieter, die auf diese Weise ihre »heißen« Pornoseiten vor nicht registrierten Benutzern schützen. Wenn Sie beispielsweise ein Forum oder Board anbieten, ist es oft sinnvoll, nur registrierte Benutzer zuzulassen, um Randalierer ausschließen zu können. Oder Sie betreiben einen Business-to-Business-Bereich, in dem Sie Informationen oder Dateien für Geschäftspartner bereitstellen, wobei diese Seiten zwar in den Webauftritt Ihrer Firma integriert sind, aber nicht allen Seitenbesuchern zugänglich sein sollen.

Besucher, die geschützte Seiten dieser Art aufrufen wollen, müssen sich mit Benutzernamen und Passwort anmelden. Den Anmeldevorgang wickelt der Browser ab, d.h., Sie benötigen dazu keine Login-Seite oder Ähnliches.

Datei .htaccess (Verzeichnisschutz)

Erstellen Sie zunächst im gewünschten Web-Dokumentverzeichnis eine Textdatei mit dem Namen ***.htaccess***. Das nachfolgende Listing zeigt den typischen Inhalt einer solchen Datei mit Angaben zum Verzeichnisschutz:

```
# .htaccess-Datei fuer Web-Verzeichnis /fotos
AuthType Basic
AuthName "Interne Vereinsfotos"
AuthUserFile .htusers
AuthGroupFile .htgroups
require user  simson-01 ricki anton-tirol fever089 butzibaer
require group members fotoviewers
```

Kommentarzeilen können in .htaccess-Dateien ebenso wie in der zentralen Apache-Konfiguration durch ein Gatterzeichen (#) am Beginn notiert werden.

Bei der Direktive `AuthName` vergeben Sie einen sprechenden Namen für den geschützten Bereich.

Bei `AuthUserFile` notieren Sie den Dateinamen, gegebenenfalls mit Pfadangabe, in dem die Benutzernamen und Passwörter von Anwendern stehen, die Zugang zum geschützten Bereich erhalten sollen. Auf den Inhalt dieser Datei kommen wir weiter unten zu sprechen.

Die Direktive `AuthGroupFile` benötigen Sie nur, wenn Sie zusätzlich zu registrierten Benutzern mit logischen Benutzergruppen arbeiten möchten. Auch hierzu geben Sie den Namen einer Datei an, in der entsprechende Gruppen definiert werden.

In beiden Fällen legen wir im obigen Beispiel die Dateien für Benutzer und Gruppen im gleichen Verzeichnis wie die .htaccess-Datei an (daher kein Pfadname). Es gibt aber auch gute Gründe dafür, solche Dateien außerhalb des Verzeichnisbereichs von `DocumentRoot` abzuspeichern, einfach um die potentielle Gefahr des Ausspionierens zu verringern. Zumindest haben wir aber mit den mit Punkt beginnenden Dateinamen ***.htusers*** und ***.htgroups*** dafür gesorgt, dass diese Dateien ebenfalls zu den »heimlichen« Konfigurationsdateien des Verzeichnisses gehören.

Mit der Direktive `require user` können Sie alle Benutzernamen angeben, die Zugang zu diesem Verzeichnis und seinen Unterverzeichnissen haben sollen. Die Benutzer müssen unter diesem Namen im Beispiel in der Datei ***.htusers*** notiert sein.

Wenn Sie mit Gruppen arbeiten, können Sie mit `require groups` anstelle von oder zusätzlich zu Einzelbenutzern Gruppen angeben, die Zugang haben.

Sowohl die zugewiesenen Benutzernamen als auch die Gruppennamen werden durch Leerzeichen voneinander getrennt.

Datei .htusers (autorisierte Benutzer)

Dies ist ebenfalls eine reine Textdatei, die z. B. auf dem Server direkt mit einem Editor wie vi angelegt und bearbeitet werden kann.

Einträge in der Benutzerdatei haben folgendes Format:

```
Benutzername:Passwort
```

Ein wichtiger Unterschied besteht hier jedoch zwischen MS Windows und Unix-/Linux-basierten Systemen. Während unter Windows die Passwörter im Klartext eingetragen werden, müssen sie unter Unix/Linux mithilfe des Dienstprogramms ***crypt*** verschlüsselt werden.

Testen Sie dazu am Shell-Prompt, ob das Kommando `crypt` verfügbar ist. Falls ja, gehen Sie bei jeder Zeile der Datei, also für jeden Benutzer wie folgt vor:

1. Vergeben Sie ein gewünschtes Passwort, z. B. `blue-28-art`.
2. Denken Sie sich irgendeine Zeichenfolge aus, etwa ein kurzes Wort wie `mit`.
3. Geben Sie nun ein: `crypt mit blue-28-art`.

Als Ergebnis erhalten Sie vom crypt-Kommando eine Zeichenfolge wie beispielsweise `mi.TFqQbP5i.g`.

4. Notieren Sie Benutzernamen mit dieser Zeichenfolge in der Benutzerdatei, wie etwa so:

```
simson-01:mi.TFqQbP5i.g
```

Unter Windows müssen Sie dagegen notieren: `simson-01:blue-28-art`.

Die Passwortverschlüsselung mit `crypt` ist eine Einwegverschlüsselung. Der Beispielanwender kann sich nun im Browser mit dem Benutzernamen `simson-01` und Passwort `blue-28-art` anmelden. Falls er sein Passwort allerdings nach Monaten vergisst und Sie als Webmaster bittet, es ihm nochmals mitzuteilen, dann werden Sie das nicht können, sofern Sie es sich nicht selbst gemerkt haben. Vergeben Sie in einem solchen Fall einfach mit `crypt` ein neues Passwort, notieren das geänderte Passwort in der Benutzerdatei hinter dem Benutzernamen und teilen dem Benutzer das neue Passwort mit.

Falls das Kommando `crypt` nicht verfügbar ist, stattdessen aber die Scriptsprache Perl installiert ist, können Sie auch folgendes Kommando am Shell-Prompt testen, um eine gültige Verschlüsselung zu erzeugen (Beispiel):

```
perl -e 'print crypt("blue-28-art","mit")'
```

Falls das auch nicht funktioniert, können Sie auch webbasierte Services nutzen, etwa den von SELFHTML. Im obigen Beispiel können Sie ein direktes Ergebnis erzeugen durch Eintippen folgender Adresse in der Browser-Adresszeile:

```
http://aktuell.de.selfhtml.org/cgi-bin/
   cryptform.pl?password=blue-28-art
```

Die erzeugten Ergebnisse für ein und dasselbe Passwort können übrigens vollkommen voneinander abweichen, abhängig vom so genannten salt-Parameter (in unserem Beispiel das Wörtchen `mit`, beim Webservice von SELFHTML etwa die Zeichenfolge `IN`).

Am Ende könnte unsere Beispieldatei *.htusers* (unter Linux) also etwa so aussehen:

```
simson-01:mi.TFqQbP5i.g
ricki:miaQ2gn81prVQ
anton-tirol:inVT82KKpJ5FM
fever089:abjYwAQpy8BG6
butzibaer:unUyEGoIsrqAw
```

Datei .htgroups (autorisierte Gruppen)

Der Aufbau einer Gruppendatei ist sehr einfach. In unserem Beispiel könnte der Inhalt etwa lauten:

```
members: simson-01 anton-tirol butzibaer
fotoviewers: ricki fever089
```

Das Schema für jede Zeile lautet also:

```
Gruppenname: Benutzername Benutzername ...
```

Für jede Gruppe, die in der *.htaccess*-Datei bei `require group` genannt wird, muss eine Gruppe angelegt werden. Jeder Gruppe dürfen nur Benutzernamen zugewiesen werden, die in der Benutzerdatei angelegt sind.

Ergebnis für Verzeichnisschutz testen

Rufen Sie, um den eingerichteten Verzeichnisschutz zu testen, eine Webseite im betroffenen Verzeichnis über HTTP auf. Der Browser sollte dann ein Authentifizierungsfenster anzeigen. Testen Sie einen eingerichteten Benutzernamen und sein Passwort aus und melden Sie sich auf diese Weise an.

Die entsprechende Abfrage besorgt der Browser mithilfe eines Dialogs wie diesem:

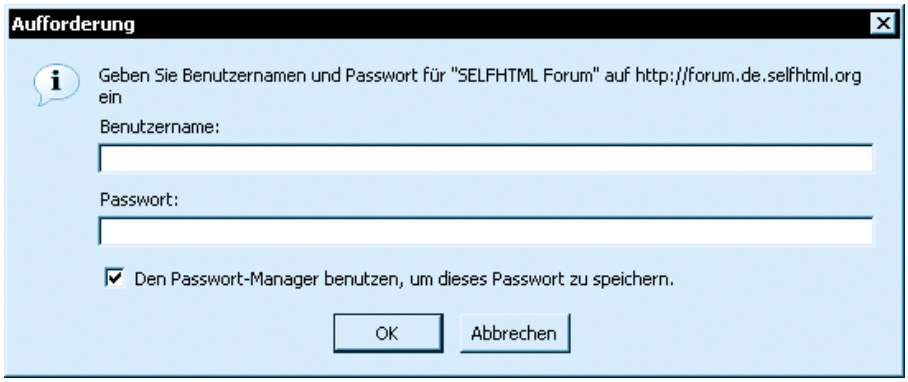


Abbildung 11.8: Authentifizierungsdialog eines Browsers bei geschützten Verzeichnissen

11.5 Automatische Weiterleitungen

Nach dem Relaunch einer Website, also nach der Umstellung vorhandener Seiten auf ein neues Layout, eine neue Technik (z.B. PHP, wo vorher statische HTML-Dateien ihren Dienst taten) und manchmal auch inhaltlichen Umstrukturierungen existieren hinterher die alten Dateinamen nicht mehr. Das bedeutet aber auch, dass die alten URIs nicht mehr existieren. Ärgerlich ist das für Anwender, die beispielsweise in ihrem Browser ein Lesezeichen (Bookmark, Favoriten) auf eine bestimmte Seite gesetzt haben, die nun nicht mehr aufrufbar ist. Auch Suchmaschinen präsentieren oftmals noch über längere Zeit die alten Adressen.

Auf dem Webserver wird dann ein 404-Fehler erzeugt. Das gleiche Problem entsteht, wenn Seiten auf eine andere Domain oder Subdomain verlagert werden.

Im Web gibt es einen interessanten Artikel vom W3-Konsortium mit dem Titel *Cool URIs don't change* (*http://www.w3.org/Provider/Style/URI.html*). Als Kompromiss für den Fall,

dass sich URIs doch mal ändern, bieten sich automatische Weiterleitungen an. Die alten Adressen bleiben erreichbar, leiten den Anwender jedoch automatisch auf die neuen um.

Eine Möglichkeit für solche Weiterleitungen haben wir bereits in Kapitel 4 kennen gelernt:

```
<meta http-equiv="Refresh" CONTENT="0;URL=http://server/">
```

Dieses HTML-Tag bewirkt eine »umgehende« Weiterleitung zu der bei URL genannten Adresse. Allerdings ist diese Form der Weiterleitung die unprofessionellste. Der Grund ist, dass an der alten Adresse eine HTML-Datei existieren muss. In der Regel ist es außerdem so, dass diese Seite erst mal zum Browser gesendet wird, bevor dieser reagieren kann, um vom Server die neue Adresse anzufordern. Im Anzeigefenster des Browsers ist das sichtbar. Es gibt jedoch serverseitige Weiterleitungsmöglichkeiten, die wesentlich eleganter sind.

Für einzelne Dateien geeignet: Weiterleitung über symbolische Links

In Kapitel 10 lernten Sie die Möglichkeit kennen, auf Linux-Systemen symbolische Links zu setzen. Dies ist eine praktische Möglichkeit für effiziente Weiterleitungen, von denen der Browser überhaupt nichts mitbekommt. Er fordert die Daten einer Datenressource an, bekommt aber die einer anderen geliefert. Angenommen, Sie haben ab Document Root betrachtet eine ehemalige Datei */impressum.html*, die nun in ein Unterverzeichnis gewandert ist und den Webpfad */kontakt/impressum.html* hat. Um eine automatische Weiterleitung im Browser zu bewirken, können Sie im früheren Verzeichnis auf Shell-Ebene eingeben:

```
ln -s kontakt/impressum.html impressum.html
```

Damit wird ein symbolischer Link unter dem Namen `imressum.html` erzeugt, also unter dem alten Namen. Er verweist auf `kontakt/impressum.html`. Jeder Versuch, die `impressum.html` zu öffnen, wird dann auf `kontakt/impressum.html` umgelenkt.

Beim Apache Webserver muss allerdings die Konfiguration so eingerichtet sein, dass der Webserver symbolischen Links folgen soll. Dazu müssen Sie bei der Direktive `Options`, die wahlweise für das gesamte Webangebot, also für alle Verzeichnisse gilt, oder innerhalb von *<Directory>...</Directory>* notiert nur für bestimmte Verzeichnisse, den Parameter `FollowSymLinks` mit angeben. Auch dezentral in .htaccess-Dateien können Sie die options-Direktive notieren und `FollowSymLinks` für das jeweilige Verzeichnis und Unterverzeichnisse festlegen.

Symbolische Links eignen sich allerdings nur für Fälle, in denen sich einzelne URIs ändern, da für jede Datei ein eigener symbolischer Link erforderlich ist. Außerdem können sie nur auf Ziele im Dateisystem des Rechners verweisen, nicht aber beispielsweise auf andere Domain-Adressen.

Für Umzüge geeignet: die Redirect-Direktive

In der Konfiguration des Apache Webservers können Sie mithilfe der Redirect-Direktive sehr effektiv ganze URI-Verzeichnisbäume umleiten. Geeignet ist diese Form der Weiterleitung vor allem, wenn ein Webangebot zu einem anderen Domain-Namen umgezogen ist, die alte Domain aber noch verfügbar ist.

Die Direktive `Redirect` kann sowohl in der zentralen Konfigurationsdatei *httpd.conf* als auch in dezentralen .htaccess-Dateien notiert werden. Ein Beispiel:

```
Redirect / http://neuerserver/
```

Damit werden alle Inhalte unterhalb der Document Root (`/`) des aktuellen Webangebots umgeleitet zu `http://neuerserver/`. Erhält der Webserver z.B. eine Anforderung für */procucts/kiwi/typ06.htm*, dann antwortet er dem aufrufenden Browser, dass diese Ressource nicht verfügbar ist und dass er stattdessen *http://neuerserver/products/kiwi/typ06.htm* anfordern soll.

Die Redirect-Direktive mappt also automatisch alte Pfadnamen, die unter das beim ersten Parameter angegebene Verzeichnis fallen, auf Pfadnamen unterhalb des Verzeichnisses, das beim zweiten Parameter angegeben ist. Allerdings muss beim zweiten Parameter ein vollständiger URI angegeben werden. Theoretisch könnte das zwar auch der eigene Server bzw. die eigene Domain sein. Doch gedacht ist die Direktive eher für Weiterleitungen zu anderen Servern bzw. zu anderen Domains.

Umstrukturierungen sind allerdings durchaus möglich, z.B.:

```
Redirect /angebote/ http://neuerserver/shop/produkte/
```

Eine Anforderung für */angebote/kiwi/typ06.htm* wird dann automatisch auf die neue Adresse *http://neuerserver/shop/produkte/kiwi/typ06.htm* gemappt und dem Browser wird diese Adresse zum Aufruf empfohlen.

Nun unterscheidet das HTTP-Protokoll zwischen temporär woanders verfügbaren Ressourcen und solchen, die dauerhaft (permanent) unter einer anderen Adresse erreichbar sind. Dafür gibt es auch unterschiedliche HTTP-Statuscodes, die der Webserver an den Browser sendet, wenn er ihm in seiner Server-Antwort mitteilt, dass die Ressource unter genannter anderer Adresse verfügbar ist. Per Voreinstellung sendet der Apache Webserver, wenn eine Redirect-Direktive greift, einen Statuscode 302 (*temporary redirect*) an den Browser. Wenn der Umzug der Daten dauerhaft ist, ist es deshalb sinnvoll, zusätzlich mit anzugeben, dass stattdessen ein Statuscode 301 (*permanent redirect*) gesendet werden soll. Die entsprechende Option wird als neuer erster Parameter angegeben. Beispiel:

```
Redirect permanent / http://neuerserver/
```

Die beiden Pfadangaben wandern also nach hinten und als erster Parameter der Redirect-Direktive wird `permanent` angegeben. An der gleichen Stelle kann auch `temp` stehen, um explizit eine temporäre Weiterleitung zu definieren, oder `seeother`, um dem Browser einen

Statuscode 303 zu senden. Letzterer bedeutet, dass es eine andere Adresse gibt, die seiner Anfrage entspricht, dass deren Inhalt jedoch nicht exakt dem entspricht, was früher unter der eigentlich angeforderten Adresse stand.

11.6 Anpassung von Server-Fehlerseiten

Eine Website sollte auch dann noch benutzerfreundlich sein, wenn der Benutzer sich mal verläuft oder in eine verbotene Zone gerät. Bei den vielen Sites beschränken sich Style und Layout nur auf die Seiten, die zum Webangebot selbst gehören. Erzeugt der Anwender jedoch einen Fehler, erhält er vom Server eine Standard-Fehlerseite in nacktem HTML ohne Bezug zum Layout des Webangebots. Der MS Internet Explorer ersetzt eine solche Seite durch eine interne eigene Seite, die aber auch nicht weniger trist aussieht.

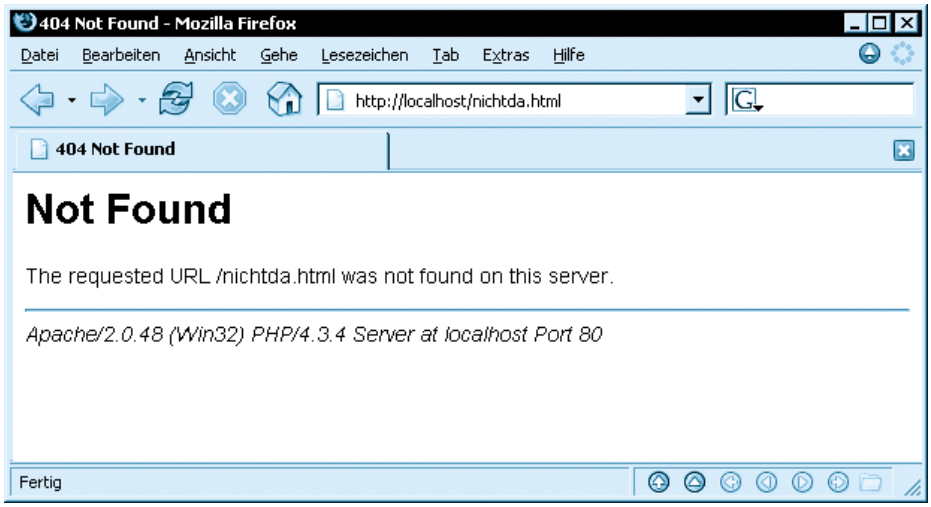


Abbildung 11.9: Standard-Fehlerseite für HTTP-Statuscode 404 vom Apache Webserver

Als ambitionierter Website-Betreiber sollten Sie dieser Tristesse ein Ende bereiten. Wenn Sie schon nicht verhindern können, dass der Server manchmal gezwungen ist, Fehlermeldungen an den aufrufenden Client zu senden, so können Sie zumindest dafür sorgen, dass dies aus Anwendersicht nicht so unerfreulich aussieht.

In der ***httpd.conf*** des Apache Webservers gibt es dazu ein paar per Default auskommentierte Direktiven mit dem Namen ***ErrorDocument***. Sie müssen solche Direktiven nur explizit in die Konfigurationsdatei zu schreiben, und zwar für jeden Fehler-Statuscode je eine. Um dabei das Layout Ihrer Webseiten zu integrieren, empfiehlt es sich, innerhalb des Webangebots ein Verzeichnis wie z.B. ***errordocs*** einzurichten. Unterhalb davon legen Sie HTML-Dateien ab, die Ihrem Weblayout entsprechen und deren Namen sinnvollerweise auf den HTTP-Fehler hinweisen. Nachfolgend ein Komplettbeispiel.

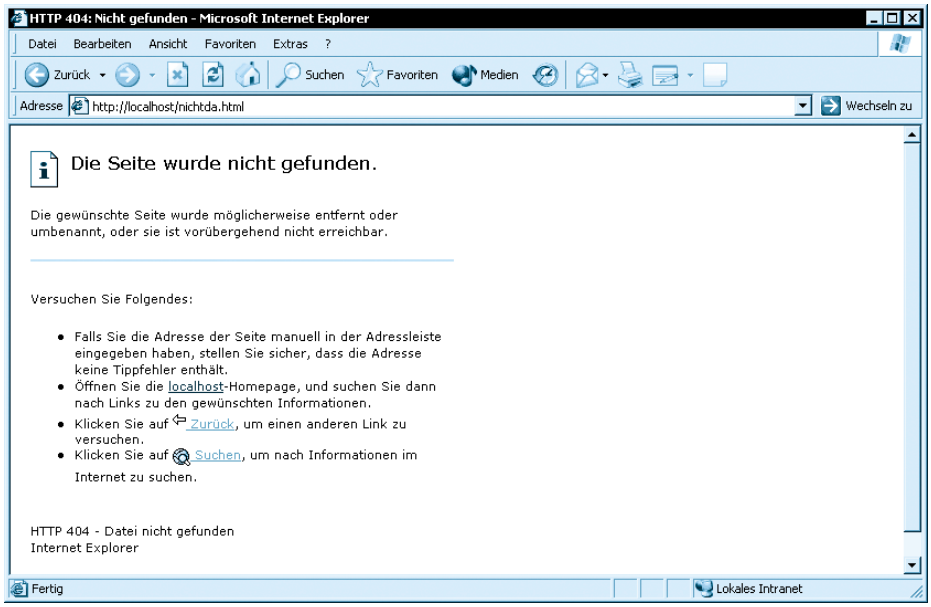


Abbildung 11.10: Interne Seite im Internet Explorer bei HTTP-Statuscode 404

Einträge in der httpd.conf

Für die einzelnen HTTP-Fehlercodes wird in der *httpd.conf* im Abschnitt für die Direktive *ErrorDocument* eine solche Direktive eingetragen:

```
ErrorDocument 400 /errordocs/http400.html
ErrorDocument 401 /errordocs/http401.html
ErrorDocument 403 /errordocs/http403.html
ErrorDocument 404 /errordocs/http404.html
ErrorDocument 405 /errordocs/http405.html
ErrorDocument 406 /errordocs/http406.html
ErrorDocument 408 /errordocs/http408.html
ErrorDocument 409 /errordocs/http409.html
ErrorDocument 414 /errordocs/http414.html
ErrorDocument 415 /errordocs/http415.html
ErrorDocument 500 /errordocs/http500.html
ErrorDocument 501 /errordocs/http501.html
ErrorDocument 503 /errordocs/http503.html
```

Die HTTP-Statuscodes über 400 und über 500 betreffen Fehlermeldungen. In unserer Liste haben wir diejenigen abgedeckt, die in der Praxis am ehesten vorkommen und so gut wie 100% ausmachen.

Das Aussehen der verzeichneten HTML-Dateien ist frei wählbar. Es muss sich übrigens auch nicht um statische HTML-Dateien handeln. Angaben wie */index.php?error=404* sind ebenso möglich. In diesem Fall könnte das PHP-Script den GET-String erkennen und daraus ablesen, welcher Inhalt dynamisch zu generieren ist.

Das HTML-Dokument kann also beispielsweise die komplette Navigation der normalen Website bereitstellen und alle üblichen Identitätsmerkmale des eigenen Layouts enthalten. Nachfolgende Abbildung greift auf ein Layoutbeispiel aus Kapitel 6 zurück und zeigt, wie eine Fehlerseite der wohl häufigsten HTTP-Fehlermeldung »404 not found« im eigenen Layout aussehen könnte:



Abbildung 11.11: HTTP-404-Meldung im eigenen Layout und benutzerfreundlich

Wenn beim Apache Webserver für einen HTTP-Statuscode ein URI angegeben ist, wird die entsprechende Seite auch im Internet Explorer angezeigt.

Verständliche Fehlertexte

Benutzerfreundliche Fehlerseiten mit verständlichen Texten setzen natürlich die Kenntnis der einzelnen HTTP-Statuscodes voraus. Nachfolgende Tabelle enthält Textvorschläge für die weiter oben in der *httpd.conf* eingetragenen Statuscodes:

Statuscode	Textvorschlag
400	Die Anfrage an den Server enthielt formale Fehler. Dies kann ein Fehler im Browser oder durch die Übertragung sein. Versuchen Sie, die Seite erneut aufzurufen. Sollte der Fehler wiederholt auftreten, wählen Sie bitte eine andere Seite aus dem Webangebot.
401	Sie haben eine Adresse aufgerufen, die eine Anmeldung erfordert. Vermutlich haben Sie vom Browser eine Eingabeaufforderung mit Anmeldedaten erhalten und den Dialog durch Abbruch beendet. Um die angeforderte Seite aufrufen zu können, müssen Sie sich hier registrieren. Sie erhalten dann Benutzername und Passwort, die Sie bei der Anmeldeaufforderung eingeben können.

Tabelle 11.3: Textvorschläge für eigene HTTP-Fehlerseiten

Statuscode	Textvorschlag
403	Sie haben eine Adresse aufgerufen, die der Server nicht anzeigen darf, z. B. ein Verzeichnis ohne Seiten, die zum Webangebot gehören. Wählen Sie in Ihrem Browser die Zurück-Funktion oder wählen Sie eine andere Seite aus unserem Webangebot aus.
404	Die angeforderte Adresse existiert nicht. Überprüfen Sie, falls Sie die Adresse selber eingegeben haben, ob Sie sich vertippt haben. Falls Sie durch Anklicken eines Links auf einem anderen Webangebot zu dieser Seite gelangt sind, sind wir dankbar, wenn Sie den Betreiber des Angebots darüber informieren, dass der Link nicht zum gewünschten Ziel führt. Wenn Sie durch einen Link auf unserem eigenen Webangebot hierhin gelangt sind, melden Sie uns die Adresse der Seite mit dem fehlerhaften Link bitte an .... Wählen Sie in Ihrem Browser die Zurück-Funktion oder wählen Sie eine andere Seite aus unserem Webangebot aus.
405	Ihr Browser hat eine nicht erlaubte Anforderungsmethode beim Aufrufen dieser Adresse gesendet. Die Anforderung kann deshalb nicht bearbeitet werden. Wählen Sie in Ihrem Browser die Zurück-Funktion oder eine andere Seite aus unserem Webangebot.
406	Ihr Browser hat ein Datenformat angefordert, das er jedoch nicht interpretieren oder zuordnen kann. Die Daten werden deshalb nicht gesendet. Wählen Sie in Ihrem Browser die Zurück-Funktion oder eine andere Seite aus unserem Webangebot.
408	Der Server hat die Verbindung beendet, weil Ihr Browser nicht im vorgegebenen Zeitraum geantwortet hat. Rufen Sie die Seite erneut auf.
409	Die angeforderten Daten können nicht übertragen werden, weil auf dem Server ein Konflikt besteht. Möglicherweise werden die Daten gerade bearbeitet und können in dieser Zeit nicht gelesen werden. Wählen Sie in Ihrem Browser die Zurück-Funktion oder eine andere Seite aus unserem Webangebot. Versuchen Sie zu einem späteren Zeitpunkt noch einmal, diese Adresse aufzurufen.
414	Ihr Browser hat eine zu lange Adresse an den Server übermittelt. Möglicherweise haben Sie Formulardaten eingegeben, deren Gesamtlänge zu umfangreich war. Wählen Sie in Ihrem Browser die Zurück-Funktion oder wählen Sie eine andere Seite aus unserem Webangebot aus.
415	Ihr Browser hat Daten an den Server übermittelt, die dieser nicht verarbeiten oder zuordnen kann. Möglicherweise haben Sie in einem Feld für Datei-Upload einen Dateityp gewählt, der vom Server nicht erkannt wird. Wählen Sie in Ihrem Browser die Zurück-Funktion oder eine andere Seite aus unserem Webangebot.
500	Bei der Ausführung eines Scripts oder Programms auf dem Server ist ein Problem aufgetreten. Möglicherweise enthält das Programm oder Script einen Fehler. Wir sind dankbar, wenn Sie uns diesen Vorfall mit dem in der Adresszeile des Browsers angezeigten Inhalt an ... melden. Wählen Sie in Ihrem Browser die Zurück-Funktion oder eine andere Seite aus unserem Webangebot.
501	Der Server kann die Anfrage Ihres Browsers nicht verarbeiten. Ein möglicher Grund ist, dass dieser eine unbekannte Anfragemethode verwendet hat. Wählen Sie in Ihrem Browser die Zurück-Funktion oder eine andere Seite aus unserem Webangebot.
503	Ihre Anfrage kann vom Server momentan nicht bearbeitet werden. Ein möglicher Grund ist Überlastung wegen zu vielen Besuchern auf der Seite. Wählen Sie in Ihrem Browser die Zurück-Funktion oder andere Seite aus unserem Webangebot.

Tabelle 11.3: Textvorschläge für eigene HTTP-Fehlerseiten (Fortsetzung)

Teil 5 – PHP und MySQL

PHP (***PHP: Hypertext Preprocessor***) ist die heute am weitesten verbreitete Script-Sprache für serverseitig dynamische Webseiten und für Datenverarbeitung in Zusammenhang mit Webseiten. MySQL ist dasjenige Datenbank-Management-System, das bei der serverseitigen Datenverwaltung in der heutigen Webpraxis am häufigsten zum Einsatz kommt. PHP unterstützt die Kommunikation mit MySQL optimal. Auf den meisten Root-Servern, die von Hosting-Providern angeboten werden, sind beide Produkte bereits vorinstalliert. Auch bei Hosting-Angeboten im Bereich Virtual Hosting haben Kunden häufig Zugriff auf PHP und MySQL. Beide Produkte sind OpenSource und lassen sich auf unterschiedlichsten Systemen installieren, unter anderem auch in lokalen PC-Entwicklungsumgebungen unter MS Windows.

12 Basiswissen PHP

Dieses Kapitel richtet sich an Betreiber von Websites, die einen Bedarf an serverseitiger Programmierung haben und sich dabei für eine Lösung mit PHP entscheiden. Ein solcher Bedarf kann schnell entstehen:

- Daten aus Feedback-Formularen auf Webseiten sollen verarbeitet werden, z. B. an den Site-Betreiber gemailt oder auf dem Server gespeichert werden.
- Das Projekt besteht aus sehr vielen Seiten, die leichter wartbar sind, wenn mit HTML-Templates gearbeitet wird statt mit statischen HTML-Dateien.
- Es sollen Daten in Seiten eingebunden werden, die in anderer Form gespeichert sind, z. B. in Textdateien, in XML-Dateien oder in Datenbanken.
- Bestimmte Bereiche im Webangebot sollen nur registrierten Anwendern zugänglich sein. Dabei soll jedoch eine eigene Benutzerverwaltung und Zugriffsüberwachung realisiert werden.
- Der Anwender soll nicht nur Informationen abrufen, sondern auch Aktionen durchführen können, wie z. B. in einem elektronischen Shop einkaufen, Kommentare zu Inhalten verfassen, in Foren diskutieren oder sich in einem Gästebuch eintragen.
- Der Anwender soll für ihn individualisierte Seiten angeboten bekommen.

Die Liste ließe sich beliebig weiter fortführen.

PHP ist für all diese Anwendungsfälle und für viele andere gerüstet. Keine andere Webprogrammiersprache bietet so viele vordefinierte Funktionen an, um mit möglichst wenig eigenem Programmieraufwand möglichst viel zu erreichen. Nicht einmal, wenn Sie dynamisch PDF-Dokumente oder Grafiken oder Flash-Movies erzeugen wollen, lässt sie PHP im Stich, und erst recht nicht, wenn es um Datenbankanbindung, Mailversand, Socketverbindungen für beliebige Anwendungsprotokolle, XML oder DOM geht. Vom einfachen Form-Mailer über einen anwenderfreundlichen Shop bis hin zum Content-Management-System lässt sich PHP für so ziemlich alle Bereiche einsetzen.

PHP ist so konzipiert, dass der Einstieg leicht fällt. Erste Erfolge sind meist schnell erzielt. Vor allem macht PHP den Umstieg von statischem HTML sehr einfach, da sich PHP-Code auch einfach mitten im ansonsten statischen HTML-Code notieren lässt.

Gleichzeitig ist die Sprache aber auch flexibel genug, um den Anforderungen professioneller Softwareentwicklung gerecht zu werden. Von objektorientierter Programmierung

bis hin zu einem ausgeklügelten Include-Automatismus steht fast alles zur Verfügung, was jenseits von »Quick&Dirty-Programmierung« benötigt wird.

PHP ist zwar vom Ansatz her leicht erlernbar, kann aber auch sehr komplex werden. Das betrifft weniger die syntaktischen Konstrukte – diese sind in der Regel besser lesbar als etwa bei der Shell-Programmierung unter Unix oder auch bei manchen Konstrukten in Perl. Schwer überschaubar ist aber dagegen die unglaubliche Menge an vordefinierten Funktionen, von denen viele ihre Besonderheiten haben.

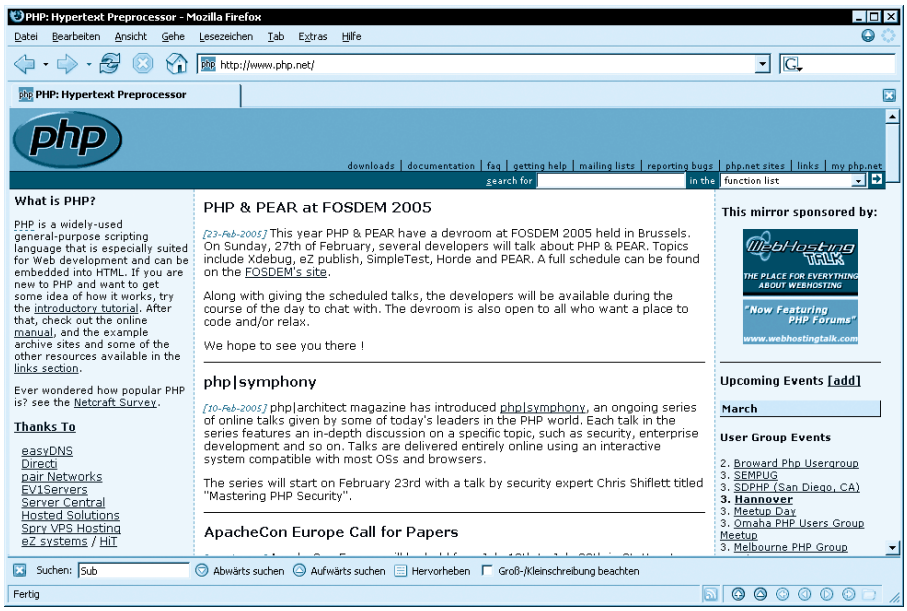


Abbildung 12.1: www.php.net ist die Startseite für alles rund um PHP

Vieles ist in Programmiersprachen, die letztlich auf der Welt der Programmiersprache C basieren, syntaktisch gleich oder ähnlich gelöst, beispielsweise die verfügbaren Variablentypen, die möglichen Konstrukte bei Kontrollstrukturen, das Prinzip von Funktionen, Parametern und Rückgabewerten und ein Basis-Set an Funktionen für verschiedene Zwecke. PHP gehört ebenso wie JavaScript zu diesen Sprachen. Wenn Sie in diesem Buch bereits die Kapitel über JavaScript und DOM durchgearbeitet haben, werden Sie in dieser Hinsicht in PHP einiges Bekanntes wiederfinden und können von dem bereits erworbenen Know-how profitieren.

12.1 PHP als Programmier- und Script-Sprache

PHP ist eine Programmier- und Script-Sprache. Bis einschließlich Version 4.x gehört PHP zu den prozeduralen Programmiersprachen, in denen eine Anweisungsreihenfolge existiert, die abgearbeitet wird. Ab Version 5.0 kann PHP zusätzlich auch zu den objektorien-

tierten Sprachen gezählt werden, die regulierbare Datenkapselung bei Objekten und Vererbung beherrscht. PHP bietet beide Formen der Programmierung an.

PHP-Code wird im üblichen Betrieb jedoch nicht in statische, für das Betriebssystem ausführbare Dateien übersetzt, sondern von einem Interpreter zur Laufzeit »on the fly« kompiliert und ausgeführt. Insofern gehört es zur Familie der Script-Sprachen. Für den Programmierer entfällt also das zeitraubende Neukompilieren vor jedem neuen Austesten.

PHP entbindet den Programmierer wie alle Script-Sprachen auch davon, sich um die Arbeitsspeicherverwaltung selbst zu kümmern. Die einzige Schnittstelle in dieser Hinsicht bietet PHP in seinen Konfigurationseinstellungen an. Dort lässt sich angeben, wie viel Speicher PHP für die Ausführung eines Scripts maximal reservieren darf.

PHP ist zwar mittlerweile auch ähnlich wie Perl als Script-Sprache auf Konsolenebene einsetzbar. Doch der eigentliche und konsequent verfolgte Einsatzzweck ist der als Script-Umgebung eines Webservers. PHP kann in HTML eingebettet werden und PHP-Dateien können wie HTML-Dateien in allen Webverzeichnissen abgelegt und ausgeführt werden (sofern es im Webserver als Modul eingebunden wird und nicht nur für CGI-Scripts erlaubt ist). PHP hat eine besonders enge Anbindung an die Gegebenheiten von HTML und HTTP. Viele vordefinierte Funktionen übernehmen Aufgaben, die speziell der Webprogrammierung entgegen kommen.

12.1.1 Geschichte und heutige Bedeutung von PHP

Die Geschichte von PHP reicht bis ins Jahr 1995 zurück. Der Aufstieg von einem kleinen Toolset zu einer ernst zu nehmenden Technologie begann 1998 auf Basis der Version 3.0. Damit wuchs allerdings auch der Druck auf Stabilität und Features der Script-Sprache.

Für die Sprachversion 4.0 wurde der Script-Interpreter von Grund auf neu entwickelt. Sein neuer Kern war die so genannte ***Zend Engine*** (ein Akronym aus den Vornamen der Chef-Entwickler Zeev Suraski und Andi Gutmans). Schon in der langen Beta-Phase zog der neue Script-Interpreter immer mehr interessierte Webprogrammierer in seinen Bann. Nachdem PHP 4.0 im Mai 2000 offiziell freigegeben wurde, sorgte eine für ein Open-Source-Projekt wirklich ungewöhnliche PR-Maschine dafür, dass PHP in Sachen Webprogrammierung alsbald zum Maß aller Dinge wurde und zuvor dominierende Sprachen wie Perl aus diesem Bereich weitgehend verdrängte.

Auch Microsoft hat PHP bis heute keine Konkurrenz machen können. Das .NET-Framework, Nachfolger der Active Server Pages (ASP), ist den meisten Webprogrammierern zu komplex und zu proprietär. Andere Konkurrenten, wie etwa die Java Server Pages (JSP), haben sich bestimmte Nischen gesichert, vor allem bei Banken und Großunternehmen. Dort wird Java auch für andere Zwecke verwendet und es haben immer noch Entscheider das Sagen, die einem kommerziellem Anbieter (Java wird von Sun Microsystems vertrieben) mehr vertrauen als einem Open-Source-Projekt wie PHP.

Laut *http://www.php.net/usage.php* war PHP im Dezember 2004 auf 18,4 Mio. virtuellen Hosts (Domains) bzw. auf 1,3 Mio. echten Hosts (IP-Adressen) mit öffentlichen Web-

servern installiert. Das entspricht etwa einem Drittel aller im öffentlichen Web erreichbaren Hosts. Der Anteil wächst weiter rapide. Ab der Produktversion 5.0 befriedigt PHP auch die Bedürfnisse von Großprojekten, die schon aus organisatorischen Gründen streng objektorientiert arbeiten. Davon ist ein weiterer Ausbreitungsschub zu erwarten.

Weiterführende Links:

Offizielles deutschsprachiges PHP-Handbuch:
http://www.php.net/manual/de/
PHP-FAQ (de), häufig gestellte Fragen:
http://www.php-faq.de/
Wikipedia-Artikel (mit diversen ausführlichen Unterartikeln):
http://de.wikipedia.org/wiki/PHP

12.1.2 Bezug und Installation von PHP

Auf den meisten Root-Servern, die sich heute bei Hosting-Providern anmieten lassen, ist PHP längst vorinstalliert. Da die Sprache jedoch weiterentwickelt wird und in zahlreichen neuen Zwischenversionen erscheint, sind hin und wieder Updates sinnvoll. Allerdings muss dabei auch vor Unvorsichtigkeit gewarnt werden: PHP sorgte schon in manch einer Zwischenversion für böse Überraschungen. Scripts liefen plötzlich nicht mehr oder falsch, weil bestimmte Funktionen andere Parameter erwarteten oder eine andere Art von Rückgabewert erzeugten. Vor einem PHP-Upgrade in einer Produktivumgebung ist daher stets erst ein Upgrade in einer möglichst ähnlichen Testumgebung zu empfehlen.

Auf jeden Fall sollten Sie sich PHP auch in Ihrer lokalen Entwicklungsumgebung, also auf Ihrem PC installieren. Nachdem dort nach Lektüre der vorangegangenen Kapitel dieses Buches auch schon der Apache Webserver installiert sein sollte, ist die Installation von PHP nur der nächste logische Schritt.

Download und Versionen

PHP können Sie von *http://www.php.net/downloads.php* downloaden. Derzeit werden sowohl die 4er-Versionsreihe als auch die 5er-Versionsreihe weitergepflegt. Für Neuinstallationen und neu zu erstellende Webprojekte spricht nichts dagegen, gleich eine 5er-Version zu nehmen.

Für MS Windows stehen fertig kompilierte Fassungen zur Verfügung. Am einfachsten ist es, eine »Installer«-Variante herunterzuladen. Damit können Sie PHP wie andere Windows-Anwendungen auch bequem über ein Setup-Programm installieren. Für Unix/Linux-Umgebungen stehen die üblichen tar.gz- oder tar.bz2-Formate zur Verfügung. Das generelle Handling zum Installieren solcher Software wurde in ***Abschnitt 10.2.8*** beschrieben. Auch für andere Betriebssysteme, wie Macintosh OS oder AS/400 werden Versionen angeboten.

PHP ist Open Source. Sie können es also ohne Lizenzgebühren downloaden und nutzen, auch für kommerzielle Webanwendungen.

Voraussetzung für den Betrieb von PHP als Script-Sprache für Webseiten ist eine funktionierende Webserver-Umgebung. Da PHP mit dem Apache Webserver optimal zusammenarbeitet, ist dieser Webserver wärmstens zu empfehlen. Download, Installation und Inbetriebnahme des Apache Webservers wurden in den ***Abschnitten 9.4.1, 9.4.2*** und ***9.4.3*** beschrieben.

Installation unter Unix/Linux

Zur Installation benötigen Sie root-Rechte.

Gehen wir von der Annahme aus, dass Sie eine tar.gz-Datei von PHP auf den Rechner geladen haben, auf dem es installiert werden soll. Zunächst müssen Sie die Datei dekomprimieren:

```
gunzip php-xxx.tar.gz
```

Dabei steht `xxx` für die Versionsnummer. Achten Sie also auf den genauen Dateinamen.

Als Nächstes entpacken Sie die entsprechende Archivdatei:

```
tar -xvf php-xxx.tar
```

Stellen Sie vor der Installation sicher, dass ein Ansi-C-Compiler auf dem System installiert ist. Das Standardprodukt auf Linux-Distributionen ist der GNU C-Compiler GCC. Über die Shell können Sie beispielsweise Folgendes eingeben:

```
gcc --help
```

Wenn dann Versionsinformationen und Aufrufmöglichkeiten zu GCC erscheinen, ist alles in Ordnung.

Als Nächstes sind grundsätzliche Konfigurationen für die Installation nötig. Das betrifft vor allem den Ablageort der endgültigen Installation sowie den Ablageort der zentralen PHP-Konfigurationsdatei ***php.ini***. Dazu dient das ***.configure***-Script. Zahlreiche weitere Einstellungen, die mit diesem Script durchführbar sind, können jedoch später in Ruhe auch in der ***php.ini*** oder durch erneuten Aufruf des Scripts vorgenommen werden. Daher genügt zunächst eine Grundkonfiguration wie diese:

```
./configure --prefix=/usr/local/bin/php --with-config-file-path=/etc
```

In diesem Beispiel wird PHP im Verzeichnis ***/usr/local/bin/php*** installiert. Die wichtige Konfigurationsdatei ***php.ini*** wird im zentralen Konfigurationsverzeichnis ***/etc*** abgelegt. Dies ist übrigens durchaus der übliche Ablageort dieser Datei.

Anschließend kann der Kompilierungsvorgang gestartet werden:

```
make
```

Nachdem dieser Vorgang abgeschlossen ist, kann die Installation gestartet werden:

```
make install
```

Die zentrale Konfigurationsdatei *php.ini* existiert nach der Installation noch nicht. Bei der Distribution ist lediglich eine Beispieldatei unter dem Namen *php.ini-dist* dabei. Kopieren Sie diese aus dem Installationsverzeichnis ins Verzeichnis */etc* unter dem endgültigen Namen:

```
cp php.ini-dist /etc/php.ini
```

Installation unter MS Windows

Wenn Sie wie empfohlen eine Windows-Installer-Version benutzen, können Sie sich auf das Setup verlassen. Dabei können Sie auch das Verzeichnis für den installierten Zustand festlegen.

Auch für die Windows-Distribution gilt: Die wichtige Konfigurationsdatei *php.ini* existiert zunächst noch nicht. Sie muss erst erstellt werden. Außerdem muss sie im Pfad liegen. Kopieren Sie dazu aus dem endgültigen Ablageverzeichnis von PHP die dort vorhandene Datei *php.ini-dist* in Ihr Windows-Verzeichnis, also z. B. in *c:\windows* oder *c:\winnt*. Benennen Sie die Datei dort um in *php.ini*.

Ferner wird empfohlen, aus dem endgültigen Ablageverzeichnis von PHP die dort vorhandene Datei *php4ts.dll* bzw. *php5ts.dll* in Ihr Windows-Systemverzeichnis zu kopieren, also z. B. in *c:\windows\system32* oder *c:\winnt\system32*.

12.1.3 Einbindung von PHP in die Apache-Konfiguration

Es gibt zwei Möglichkeiten, PHP-Scripts von Apache ausführen zu lassen: entweder als Apache-Modul oder als CGI-Scripts. Die Lösung als Apache-Modul ist dann vorzuziehen, wenn PHP intensiv eingesetzt werden soll und auch Priorität gegenüber anderen Script-Sprachen auf dem Server haben soll. Außerdem entfallen bei der Variante »PHP als Modul« die typischen CGI-Merkmale wie Beschränkung auf bestimmte Verzeichnisse usw. Nachteil ist jedoch, dass Apache damit insgesamt mehr Arbeitsspeicher pro Prozess benötigt, und zwar auch dann, wenn gar keine PHP-Scripts ausgeführt werden müssen. Wenn PHP als Apache-Modul läuft, laufen PHP-Scripts auf Unix-/Linux-Umgebungen auch unter dem Benutzernamen von Apache, sie haben also die gleichen Rechte wie dieser.

Auf den meisten öffentlichen Server-Rechnern ist PHP als Apache-Modul konfiguriert.

Einbindung unter Unix/Linux

Um PHP als Apache-Modul zu konfigurieren, müssen Sie die Installation von PHP korrigieren, indem Sie in dem Verzeichnis, in dem Sie PHP dekomprimiert und entpackt haben, das *.configure*-Script erneut aufrufen mit:

```
./configure --with-apxs=[Pfad/zu/apxs]
```

Dabei ist *apxs* eine ausführbare Datei von Apache. Wo sie genau liegt, können Sie herausbekommen durch Eingabe von:

```
find / -name apxs
```

Nachdem das .configure-Script durchgelaufen ist, müssen Sie erneut `make` und `make install` ausführen.

Wechseln Sie nun in das Verzeichnis, in dem der Apache Webserver installiert ist, also z. B. in */usr/local/bin/apache*, und dort ins Unterverzeichnis *libexec*. Dort sollte sich nun eine Datei *libphp4.so* oder *libphp5.so* befinden, je nachdem, ob Sie PHP 4.x oder PHP 5.x installiert haben.

Sind diese Voraussetzungen erfüllt, müssen Sie in der *httpd.conf*, also in der zentralen Konfigurationsdatei des Apache Webservers, folgende Einträge vornehmen:

- Das Apache-Modul von PHP muss geladen werden. Das geschieht für PHP 4.x durch folgenden Eintrag:

  ```
  LoadModule php4_module  libexec/libphp4.so
  ```

  Und für PHP 5.x durch folgenden Eintrag:

  ```
  LoadModule php5_module  libexec/libphp5.so
  ```

  Notieren Sie die Einträge innerhalb der *httpd.conf* dort, wo LoadModule-Einträge für dynamische Module (DSO) stehen sollten.

- Suchen Sie als Nächstes in der *httpd.conf* nach *AddType*. Notieren Sie zusätzlich zu eventuell vorhandenen Einträgen dieser Art den folgenden:

  ```
  AddType application/x-httpd-php .php
  ```

- Bei der Direktive *DirectoryIndex* sollte ein Dateiname des Typs *.php* mit aufgenommen werden, in den meisten Fällen *index.php*. Das gilt auch für eventuell definierte virtuelle Hosts.

Möchten Sie dagegen PHP nicht als Apache-Modul einbinden, sondern einfach als möglichen Handler für CGI-Scripts, sind folgende Änderungen in der *httpd.conf* erforderlich:

- Überprüfen Sie, ob das CGI-Modul des Apache geladen wird. Dazu muss folgende Zeile ohne Kommentarzeichen davor existieren:

  ```
  LoadModule cgi_module modules/mod_cgi.so
  ```

- Stellen Sie sicher, dass mindestens eine geeignete ScriptAlias-Direktive existiert, z. B.:

  ```
  ScriptAlias /cgi-bin/ "/var/www/cgi-bin/"
  ```

- Suchen Sie nach folgender Zeile: `AddHandler  cgi-script  .cgi`. Notieren Sie daran anschließend folgende beiden Zeilen:

  ```
  Action php-script /cgi-bin
  AddHandler php-script .php
  ```

Nachdem Sie die *httpd.conf* geändert haben, speichern Sie die Datei und führen Sie einen Neustart des Apache Webservers durch wie in ***Abschnitt 9.4.2*** beschrieben.

Einbindung unter MS Windows

Unter MS Windows sollten Sie zunächst die Dateien *php4apache.dll* bzw. *php4apache2.dll* (je nach Apache-Version) aus dem PHP-Unterverzeichnis *sapi* in Ihr Windows-Systemverzeichnis kopieren, also z. B. in *c:\windows\system32* oder *c:\winnt\system32*. Dadurch liegen diese Dateien zusätzlich im Pfad und mögliche Probleme des Apache Webservers beim Laden dieser Bibliotheken werden vermieden.

Auch unter Windows können Sie PHP wahlweise als Apache-Modul oder als Handler für CGI-Scripts einbinden. In beiden Fällen müssen Sie die zentrale *httpd.conf* des installierten Apache Webservers bearbeiten.

Um PHP als Apache-Modul zu installieren, sind folgende Einträge erforderlich:

- Das Apache-Modul von PHP muss geladen werden. Das geschieht für PHP 4.x durch folgenden Eintrag:

  ```
  LoadModule php4_module  [Pfad zu PHP]/sapi/php4apache*.dll
  ```

  Und für PHP 5.x durch folgenden Eintrag:

  ```
  LoadModule php5_module  [Pfad zu PHP]/sapi/php5apache*.dll
  ```

  Dabei ist `[Pfad zu PHP]` der vollständige Laufwerkspfad zum Programmverzeichnis von PHP, also etwa `c:/php` oder `c:/Programme/php`. Benutzen Sie einfache Schrägstriche zur Verzeichnistrennung, nicht den unter Windows üblichen Backslash. Welche Datei Sie genau adressieren müssen, hängt von der PHP- und der Apache-Version ab. Bei PHP 4.x und Apache 1.3.x heißt sie *php4apache.dll*, bei PHP 4.x und Apache 2.0.x *php4apache2.dll*, bei PHP 5.x und Apache 1.3.x *php5apache.dll* und bei PHP 5.x und Apache 2.0.x *php5apache2.dll*.

  Notieren Sie die Einträge innerhalb der *httpd.conf* dort, wo LoadModule-Einträge für dynamische Module (DSO) stehen sollten.

- Suchen Sie als Nächstes in der *httpd.conf* nach *AddType*. Notieren Sie zusätzlich zu eventuell vorhandenen Einträgen dieser Art den folgenden:

  ```
  AddType application/x-httpd-php .php
  ```

- Bei der Direktive *DirectoryIndex* sollte ein Dateiname des Typs *.php* mit aufgenommen werden, in den meisten Fällen *index.php*. Das gilt auch für eventuell definierte virtuelle Hosts.

Möchten Sie dagegen PHP nicht als Apache-Modul einbinden, sondern einfach als möglichen Handler für CGI-Scripts, sind folgende Änderungen in der *httpd.conf* erforderlich:

- Überprüfen Sie, ob das CGI-Modul des Apache geladen wird. Dazu muss folgende Zeile ohne Kommentarzeichen davor existieren:

```
LoadModule cgi_module modules/mod_cgi.so
```

- Stellen Sie sicher, dass mindestens eine geeignete ScriptAlias-Direktive existiert, z. B.:

```
ScriptAlias /cgi-bin/ "c:/webdocs/cgi-bin/"
```

- Suchen Sie nach folgender Zeile: `AddHandler cgi-script .cgi`. Notieren Sie daran anschließend folgende beiden Zeilen:

```
Action php-script /cgi-bin
AddHandler php-script .php
```

Nachdem Sie die *httpd.conf* geändert haben, speichern Sie die Datei und führen Sie einen Neustart des Apache Webservers durch wie in ***Abschnitt 9.4.2*** beschrieben.

Austesten von PHP unter Apache: ein erstes Script

Wenn Sie alles richtig installiert haben, sollten PHP-Scripts nun ausführbar sein. Wir nehmen in unseren nachfolgenden Beispielen an, dass PHP als Apache-Modul installiert ist. PHP-Dateien können dann überall unterhalb der Document Root, also unterhalb des Startverzeichnisses für Webdokumente, abgelegt werden. Erstellen und bearbeiten lassen sie sich mit jedem Texteditor. Bestens geeignet sind auch code-basierte HTML-Editoren. Viele davon unterstützen von Haus aus Syntax-Highlighting für PHP-Code, was in der Praxis sehr hilfreich ist.

Folgendes Listing eignet sich, unter einem Dateinamen wie *php-test.php* irgendwo unterhalb der Document Root abgespeichert, zu einem ersten Test, ob PHP korrekt funktioniert:

Listing 12.1: Erste PHP-Datei zum Testen von PHP

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>PHP-Test</title>
</head>
<body>
<h1><?php echo "Hier ist PHP!" ?></h1>
</body>
</html>
```

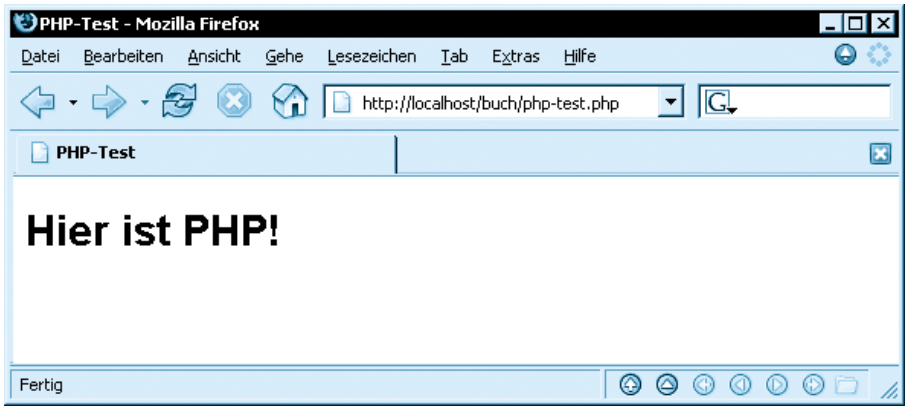


Abbildung 12.2: Erstes PHP-Script im Browser

Wenn es in Ihrem Browser so aussieht wie abgebildet, ist PHP korrekt installiert und in Apache eingebunden.

Das Beispiel zeigt auch schon, wie simpel PHP sein kann. Unser Listing sieht aus wie eine gewöhnliche HTML-Datei. Nur an einer Stelle, nämlich im Elementinhalt der h1-Überschrift, ist ein Bereich notiert, der mit `<?php` beginnt und mit `?>` endet. Innerhalb davon kann PHP-Code stehen. In unserem Beispiel wird einfach nur etwas Text ausgegeben.

Wenn Sie sich im Browser den HTML-Quelltext ansehen (»view source«), dann werden Sie feststellen, dass von dem PHP-Script-Bereich nichts mehr zu sehen ist. Dort steht innerhalb von `<h1>...</h1>` nur noch der von PHP ausgegebene Text.

Genau das ist das Prinzip serverseitiger Scripts. Die Scripts werden ausgeführt, ***bevor*** die Daten zum Browser übertragen werden. Ein serverseitiges Script muss dafür sorgen, dass sinnvolle Daten zum Browser gelangen, also z.B. HTML-Code. Beim Browser selbst kommt dann nur noch der ausgegebene HTML-Code an, aber nichts mehr vom Programmcode des Scripts. Der Browser weiß nicht einmal, dass ein Script den HTML-Code erzeugt hat, bzw. es ist ihm völlig egal. Er erhält seine Daten vom Webserver genauso, als würde es sich um eine statische HTML-Datei handeln.

Umgekehrt ist es Aufgabe des Webservers, richtig zu reagieren. Versuchen wir zu verstehen, was genau passiert:

1. Der Webserver erhält vom Browser eine Anfrage, in der dieser mit GET oder POST dazu auffordert, ihm die Daten einer Datenressource namens ***php-test.php*** zu senden.
2. Der Webserver erkennt an der Endung, dass in diesem Fall der PHP-Interpreter ausgeführt werden muss. Die Endung ist ihm durch seine Konfiguration bekannt.
3. Zunächst muss der Webserver prüfen, ob er PHP-Scripts im angeforderten Verzeichnis überhaupt ausführen darf. Angenommen, PHP ist als Modul konfiguriert, dann darf es in allen Webverzeichnissen ausgeführt werden.

4. Der Webserver startet den als Modul geladenen PHP-Interpreter mit der angeforderten Scriptdatei. Der PHP-Interpreter führt das Script aus und reicht den vom Script erzeugten Datenoutput an den Webserver durch.
5. Der Webserver sendet den Datenoutput an den anfragenden Browser.

12.1.4 Konfiguration von PHP

Bevor wir uns genauer mit der PHP-Programmierung beschäftigen, ist es nötig, einen Blick in die zentrale Konfigurationsdatei von PHP, *php.ini*, zu werfen. Jeder PHP-Programmierer sollte zumindest mit den wichtigsten Einträgen in dieser Datei vertraut sein. Denn sie beeinflussen das Verhalten von PHP zum Teil erheblich.

Auf Linux-Systemen wird die *php.ini* gerne unter */etc* abgelegt. Auf Windows-Systemen sollte sie sich im Windows-Verzeichnis befinden. Bearbeitet werden kann sie mit jedem Texteditor.

Die Optionen in der *php.ini* haben das Format *Bezeichner = Wert*. Alle Zeilen, die mit einem Semikolon beginnen, sind Kommentarzeilen. Wenn Sie die *php.ini* wie empfohlen durch Kopieren der *php.ini-dist* erzeugt haben, dann finden Sie eine reichhaltig auskommentierte Datei vor. Die Kommentare unterstützen beim Verständnis der einzelnen Konfigurationseinstellungen.

Änderungsberechtigungen

Es gibt Möglichkeiten, die Einträge in der *php.ini* zu überschreiben, und zwar:

- In der Konfiguration des Apache Webservers, insbesondere in der dezentralen Konfiguration für bestimmte Verzeichnisse, also etwa in .htaccess-Dateien. Dann gilt die dort festgelegte Einstellung im entsprechenden Verzeichnis und in seinen eventuellen Unterverzeichnissen. Außerhalb des Verzeichnisses gilt weiterhin die Einstellung der *php.ini*.
- In PHP-Scripts. Dann gilt die dort programmierte Einstellung für das Script, in dem sie notiert ist. Ansonsten gilt weiterhin die Einstellung der *php.ini*.

Für jede Option in der *php.ini* ist daher in PHP festgelegt, ob und wo der zugewiesene Wert überschreiben darf. Dafür gibt es folgende vier Berechtigungsklassen:

- *PHP_INI_USER:* Optionen mit dieser Klasse dürfen durch ein PHP-Script überschrieben werden.
- *PHP_INI_PERDIR:* Optionen mit dieser Klasse dürfen durch ein PHP-Script überschrieben werden. Ferner dürfen sie durch entsprechende Einträge in der verzeichnisspezifischen Apache-Konfiguration überschrieben werden, also z. B. in einer .htaccess-Datei.
- *PHP_INI_SYSTEM:* Optionen mit dieser Klasse dürfen durch ein PHP-Script überschrieben werden. Ferner dürfen sie in der zentralen Apache-Konfiguration (nicht verzeichnisspezifisch) überschrieben werden.

- *PHP_INI_ALL:* Optionen mit dieser Klasse dürfen durch ein PHP-Script oder auf jede andere Art und Weise überschrieben werden.

Wenn PHP im Apache Webserver als Modul installiert ist, können Sie PHP-Optionen in der Apache-Konfiguration mithilfe der Direktive `php_value` notieren. Beispiel:

```
php_value variables_order "EPGCS"
```

Diese Direktive, in einer .htaccess-Datei notiert, bewirkt unter anderem, dass PHP in dem entsprechenden Verzeichnis die Daten über den POST-Kanal schneller ermittelt als die über den GET-Kanal.

Für die Praxis sind die Änderungsberechtigungen nur dann von Bedeutung, wenn Sie solche Feinjustierungen in der Apache-Konfiguration vornehmen möchten. PHP-Scripts dürfen alle Optionen überschreiben. *PHP_INI_USER* ist also die Mindestberechtigung aller Optionen.

Optionen für Sprachverhalten

Diese Optionen regeln allgemeine Einstellungen zu PHP. In der *php.ini* finden Sie die Einstellungen unterhalb von:

```
;;;;;;;;;;;;;;;;;;;
; Language Options ;
;;;;;;;;;;;;;;;;;;;
```

Nachfolgende Tabelle beschreibt einige wichtige Optionen:

Option	Default-Wert	Bedeutung	Berechtigung
engine	*On*	Nur bei *On* funktioniert die Einbindung von PHP unter Apache.	
short_open_tag	*On*	Wenn *On*, dürfen Sie PHP-Code in <? ... ?> einschließen. Wenn *Off*, müssen Sie <?php ... ?> notieren. Letzteres ist auf jeden Fall vorzuziehen, wenn Sie mit PHP XML verarbeiten wollen.	*PERDIR, SYSTEM*
asp_tags	*Off*	Wenn *On*, kann PHP so wie ASP innerhalb von <% ... %> notiert werden.	*PERDIR, SYSTEM*
precision	*14*	Anzahl der ausgegebenen Stellen bei Fließkommazahlen.	*ALL*
y2k_compliance	*On*	Jahr-2000-Kompatibilität.	*ALL*

Tabelle 12.1: Sprachverhaltensoptionen in der php.ini

Optionen für Ressourcenbelegung

Diese Optionen regeln, wie stark PHP das System, auf dem es läuft, belasten darf. In der *php.ini* finden Sie die Einstellungen unterhalb von:

```
;;;;;;;;;;;;;;;;;;;
; Resource Limits ;
;;;;;;;;;;;;;;;;;;;
```

Nachfolgende Tabelle beschreibt diese Optionen:

Option	Default-Wert	Bedeutung	Berechtigung
max_execution_time	*30*	Wenn ein Script zur Ausführung mehr Sekunden benötigt als angegeben, wird es vom Parser abgebrochen.	
max_input_time	*60*	Wenn ein Script mehr Sekunden als angegeben keinen Input z.B. über GET oder POST erhält, wird es vom Parser abgebrochen.	
memory_limits	*8M*	Arbeitsspeicher, den ein Script maximal belegen darf. Integer-Wert mit M dahinter für Megabyte, oder –1 für »keine Begrenzung«.	*ALL*

Tabelle 12.2: Ressourcen-Belegungsoptionen in der php.ini

Optionen für Fehlerbehandlung

Diese Optionen regeln, wie PHP bei Syntaxfehlern oder problematischem Code reagieren soll. Gerade während der Entwicklungsphase von größeren Scripts ist es sinnvoll, die Fehleranzeige und das Fehler-Logging etwas großzügiger einzustellen. In Produktivumgebungen dagegen sollten Fehler überhaupt nicht angezeigt werden. In der *php.ini* finden Sie die Einstellungen unterhalb von:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Error handling and logging ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Nachfolgende Tabelle beschreibt die wichtigsten dieser Optionen:

Option	Default-Wert	Bedeutung	Berechtigung
error_reporting	*E_ALL & ~E_NOTICE*	Bestimmt, welche Vorkommnisse behandelt werden. Mögliche Werte und Beispiele siehe Kommentare in *php.ini*.	*ALL*
display_errors	*On*	Gibt Fehler im Browser aus. Sollte bei Produktivumgebungen auf *Off* gestellt werden.	*ALL*
display_startup_errors	*Off*	Gibt bei *On* Fehler beim Interpreter-Start aus.	*ALL*

Tabelle 12.3: Fehlerbehandlungsoptionen in der php.ini

Option	Default-Wert	Bedeutung	Berechtigung
log_errors	*Off*	Protokolliert Fehler in der Logdatei statt sie im Browser auszugeben. Empfehlenswert für Produktivumgebungen. Die Logdatei kann bei *error_log* festgelegt werden.	*ALL*
log_errors_max_len	*1024*	Länge für Fehlermeldungen begrenzen (Anzahl Byte). 0 bedeutet: keine Begrenzung.	*ALL*
ignore_repeated_errors	*Off*	Wenn eine PHP-Datei eine Meldung mehrmals produziert, wird sie nur einmal protokolliert.	*ALL*
ignore_repeated_source	*Off*	Wenn eine PHP-Datei eine Meldung mehrmals produziert, wird die betroffene Quelltextumgebung nur einmal protokolliert.	*ALL*
report_memleaks	*On*	Fehler bei der Speicherfreigabe werden ausgegeben bzw. protokolliert.	*ALL*
track_errors	*Off*	Die jeweils letzte Meldung steht dem Script selbst in einer Variablen namens *$php_errormsg* zur Verfügung.	*ALL*
html_errors	*On*	HTML-Markup in Meldungsausgaben verwenden.	*ALL*
error_log		Absoluter Pfadname der gewünschten Logdatei. Der Webserver muss für diese Datei Schreibrecht haben.	*ALL*

Tabelle 12.3: Fehlerbehandlungsoptionen in der php.ini (Fortsetzung)

Optionen für Datenbehandlung

Diese Optionen regeln wichtige Voreinstellungen, wie PHP mit Daten umgeht. In der *php.ini* finden Sie die Einstellungen unterhalb von:

```
;;;;;;;;;;;;;;;;;;
; Data Handling ;
;;;;;;;;;;;;;;;;;;
```

Nachfolgende Tabelle beschreibt die wichtigsten dieser Optionen:

Option	Default-Wert	Bedeutung	Berechtigung
track-vars	*On*	Wenn *On*, stehen GET- und POST-Daten, Umgebungsvariablen, Servervariablen und Cookies in so genannten Superglobal-Arrays zur Verfügung. Sollte unbedingt auf On stehen und die Superglobals sollten heutzutage anstelle älterer Lösungen für entsprechende Datenzugriffe verwendet werden.	

Tabelle 12.4: Datenbehandlungsoptionen in der php.ini

Option	Default-Wert	Bedeutung	Berechtigung
variables_order	*»EGPCS«*	Reihenfolge der Variablen beim Parsen: E = Umgebungsvariablen, G = GET-Daten, P = POST-Daten, C = Cookie-Daten, S = Servervariablen.	*ALL*
register_globals	*Off*	Wenn *On*, stehen GET- und POST-Daten, Umgebungsvariablen, Servervariablen und Cookies in bestimmten, normalen Variablen zur Verfügung. Sollte heutzutage nicht mehr verwendet werden, also auf *Off* stehen.	*PERDIR, SYSTEM*
post_max_size	*8M*	Integer-Wert mit M dahinter für Megabyte. Maximaler Umfang von Daten, die mittels POST-Methode an ein Script übermittelt werden können. Interessant vor allem, wenn Datei-Uploads verarbeitet werden müssen.	*PERDIR, SYSTEM*

Tabelle 12.4: Datenbehandlungsoptionen in der php.ini (Fortsetzung)

Diverse Optionen

Die *php.ini* enthält zahlreiche weitere Optionen, die zum Teil das Zusammenspiel zwischen PHP und bestimmten anderen Softwareprodukten regeln. Viele davon werden in der Praxis nur sehr selten angefasst. Es gibt jedoch auch Optionen, die Sie möglicherweise ändern oder zumindest kennen sollten. Nachfolgend eine Liste solcher Optionen:

Option	Default-Wert	Bedeutung	Berechtigung
doc_root		Nur von Bedeutung, wenn PHP im CGI-Modus läuft. In diesem Fall das Stammverzeichnis für PHP-Scripts als absolute Pfadangabe.	*SYSTEM*
file_uploads	*On*	Wenn *On*, werden Inhalte aus File-Upload-Feldern in HTML an PHP-Scripts übertragen, wenn *Off*, dann nicht.	*PERDIR, SYSTEM*
upload_max_filesize	*2M*	Integer-Wert mit M dahinter für Megabyte. Wenn *file_uploads* auf *On* steht, kann hier die maximale Größe notiert werden, die hochgeladene Dateien haben dürfen.	*PERDIR, SYSTEM*
allow_url_fopen	*On*	Wenn On, dürfen Scripts die Methoden zum Öffnen von Dateien auch auf URIs anwenden. PHP stellt dann im Hintergrund eine Socketverbindung her und lädt die URI-Inhalte wie normale Dateien.	*SYSTEM*
SMTP		Bei lokalen PHP-Installationen unter Windows von Bedeutung. IP-Adresse des Servers für ausgehende Mails. Wird für Mailversand aus PHP benötigt.	*ALL*

Tabelle 12.5: Diverse weitere Optionen in der php.ini

Option	Default-Wert	Bedeutung	Berechtigung
session.save_handler	*files*	Name einer Funktion, die Session-Dateien speichern kann. Sollte auf *files* stehen bleiben.	*ALL*
session.save_path	*»/tmp«*	Absoluter Pfadname des Verzeichnisses, unter dem Session-Dateien gespeichert werden. Es sollte sich um ein typisches Verzeichnis für temporäre Daten handeln. Muss unter MS Windows in der Regel geändert werden, da dort */tmp* üblicherweise nicht existiert.	*ALL*
session.name	*»PHPSESSID«*	Default-Name für Session-Cookies.	*ALL*
session.use_cookies	*On*	Wenn *On*, versucht PHP für Session-Daten ein Session-Cookie an den Browser zu senden.	*ALL*
session.use_only_cookies	*Off*	Wenn *On*, funktionieren Sessions nur, wenn der Anwender Session-Cookies akzeptiert. Scripts laufen dann möglicherweise nicht wie gewünscht, wenn der Anwender keine Cookies akzeptiert. Dafür wird aber dem Ausspähen von Session-Daten vorgebeugt, was auch im Interesse des Anwenders ist.	*ALL*
session.cache_expire	*180*	Anzahl Minuten, die Session-Seiten im Cache-Speicher bleiben.	*ALL*
session.use_trans_sid	*Off*	Die Session-ID wird an den GET-String angehängt, wodurch sich Sessions via URI ansprechen lassen. Sollte normalerweise auf *Off* stehen, da es das Ausspähen von Session-Daten erleichtert.	*ALL*

Tabelle 12.5: Diverse weitere Optionen in der php.ini (Fortsetzung)

Für einige der genannten Optionen, vor allem diejenigen zu Sessions, sind Kenntnisse in PHP-Sessions erforderlich. Wir werden dieses wichtige Thema im vorliegenden Kapitel behandeln.

Die Funktion phpinfo()

Mit dieser Funktion können Sie im Browser eine komplett im Layout der Seiten von *php.net* formatierte Ausgabe mit allen aktuellen Konfigurationseinstellungen von PHP ausgeben lassen. Notieren Sie dazu einfach in einer Datei, die Sie beispielsweise *phpinfo.php* nennen, nichts weiter als folgende Zeile:

```
<?php phpinfo(); ?>
```

Rufen Sie diese Datei via HTTP im Browser auf.



Abbildung 12.3: Ausgabe von phpinfo() im Browser

12.2 PHP-Scripting für den Anfang

Nach so viel grauer Theorie wenden wir uns nun wieder den Code-Beispielen zu.

PHP-Scripts erhalten die Standarddateiendung *.php*. Je nachdem, was in der Konfiguration des Webservers Apache bei *AddType application/x-httpd-php* angegeben wurde, sind auch andere dort genannte Dateiendungen erlaubt. PHP-Scripts, die mittels URI, also etwa im Browser, aufrufbar sein sollen, müssen unterhalb der Document Root abgelegt werden, also ebenso wie HTML-Dateien, die über einen Webserver aufrufbar sein sollen.

Auf Linux/Unix-Systemen sollten PHP-Dateien und die Verzeichnisse, in denen sie abgelegt sind, stets auf »ausführbar« gesetzt werden, und zwar in der Regel für alle Berechtigungstypen (Dateieigentümer, seine Primärgruppe und Rest der Welt). Eine typische Wahl ist `chmod 755`.

In diesem Abschnitt stellen wir bereits einige praxistypische Beispiele für den Einsatz von PHP vor, ohne dabei jedoch näher auf die Sprachdetails einzugehen. Ziel ist es, ein Gefühl dafür zu bekommen, wie PHP einsetzbar ist und wie PHP-basierte Websites strukturiert werden können. Verzweifeln Sie nicht, wenn Sie einzelne Passagen in den hier vorgestellten Beispielen zu diesem frühen Zeitpunkt noch nicht ganz verstehen. Auf syntaktische Details gehen wir erst später ein.

12.2.1 PHP in HTML

Bereits im ersten Testscript haben wir gesehen, dass sich PHP problemlos in HTML einbetten lässt. Im Gegensatz zu JavaScript kann PHP sogar an beliebigen Stellen im HTML-Code eingebettet werden, d.h., die Syntax von HTML darf durch PHP-Bereiche verletzt werden. Wichtig ist nur, dass am Ende fehlerfreies HTML an den Browser gesendet wird. Nachfolgendes Beispiel zeigt, wie PHP-Code an den unterschiedlichsten Stellen innerhalb eines HTML-Dokuments nützliche Dienste leisten kann:

Listing 12.2: PHP in HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<meta http-equiv="expires" content="<?php
$in_3_days = mktime(date("h"),date("i"),date("s"),date("n"),
                    date("j"),date("Y")) + (72 * 60 * 60);
echo gmdate("D, d M Y H:i:s", $in_3_days)," GMT";
?>">
<title>Bilder</title>
<style type="text/css">
td.thumb {
  vertical-align:top;
  padding-right:15px;
  padding-bottom:10px;
}
td.info {
  vertical-align:top;
}
img.thumb_h {
  width:120px;
  height:96px;
  border:solid 1px black;
}
img.thumb_v {
  width:96px;
  height:120px;
  border:solid 1px black;
}
h1 {
  border-bottom:1px solid black;
}
</style>
</head>
<body>
<h1>Bilder</h1>

<table>
<tr>
```

```
<td class="thumb">
<a href="bild.php?bild=01">
<img src="bild01thumb.jpg" class="thumb_h"
     alt="Bild 1 Vorschau"></a>
</td>
<td class="info">
<b>Bild 1:</b><br>
<?php
$bytes = filesize("bild01.jpg");
$pixel = GetImageSize("bild01.jpg");
echo "Breite &#215; Höhe: ",
     $pixel[0]," &#215; ",$pixel[1]," Pixel<br>";
echo "Größe: ",round($bytes / 1024)," KByte";
?>
</td>
</tr><tr>
<td class="thumb">
<a href="bild.php?bild=02">
<img src="bild02thumb.jpg" class="thumb_v"
     alt="Bild 2 Vorschau"></a>
</td>
<td class="info">
<b>Bild 2:</b><br>
<?php
$bytes = filesize("bild02.jpg");
$pixel = GetImageSize("bild02.jpg");
echo "Breite &#215; Höhe: ",
     $pixel[0]," &#215; ",$pixel[1]," Pixel<br>";
echo "Größe: ",round($bytes / 1024)," KByte";
?>
</td>
</tr><tr>
<td class="thumb">
<a href="bild.php?bild=03">
<img src="bild03thumb.jpg" class="thumb_h"
     alt="Bild 3 Vorschau"></a>
</td>
<td class="info">
<b>Bild 3:</b><br>
<?php
$bytes = filesize("bild03.jpg");
$pixel = GetImageSize("bild03.jpg");
echo "Breite &#215; Höhe: ",
     $pixel[0]," &#215; ",$pixel[1]," Pixel<br>";
echo "Größe: ",round($bytes / 1024)," KByte";
?>
</td>
</tr>
</table>
<?php
echo "<p style=\"border-top:solid 1px black;\">";
echo $_SERVER['SERVER_NAME'],"</p>";
?>
```

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

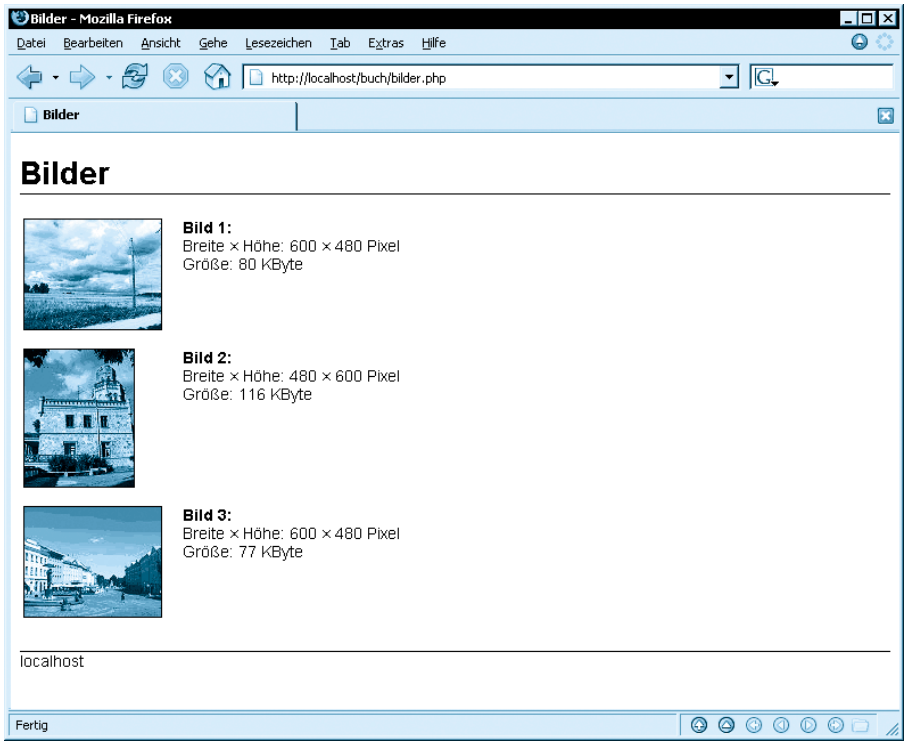


Abbildung 12.4: Browser-Ausgabe des PHP-Scripts

Das Beispiel gibt ein paar anklickbare Vorschaubilder aus (auch als *Thumbnails* oder *Thumbs* bezeichnet), die zum Betrachten ihrer größeren Varianten einladen sollen. Neben jedem Vorschaubild stehen Informationen über Bildausmaße und Dateigröße des Originalbilds. Diese Informationen stehen jedoch nicht fest im HTML-Quelltext, sondern werden von PHP dynamisch ermittelt. Würde also etwa eines der aufrufbaren Bilder durch eine andere Version ausgetauscht, so würden beim Aufruf der Seite jederzeit die aktuellen Daten zum Originalbild angezeigt.

PHP mitten im HTML-Tag: die dynamische Meta-Angabe

Bevor wir näher auf die Passagen mit den Vorschaubildern eingehen, erst einmal zum ersten notierten PHP-Bereich im HTML-Quelltext. Dieser befindet sich in den Kopfdaten und zwar innerhalb der Wertzuweisung an das content-Attribut eines Meta-Tags:

```
<meta http-equiv="expires" content="<?php
$in_3_days = mktime(date("h"),date("i"),date("s"),date("n"),
                    date("j"),date("Y")) + (72 * 60 * 60);
echo gmdate("D, d M Y H:i:s", $in_3_days)," GMT";
?>">
```

Aus HTML-Sicht stellt dies eine syntaktische Verletzung dar, da innerhalb von Wertzuweisungen an Attribute HTML-eigene Zeichen wie < oder > nicht unmaskiert vorkommen dürfen. Da der PHP-Bereich jedoch nach Ausführung des Scripts durch seinen Output ersetzt wird, erscheint am Ende HTML-konformer Inhalt an der betroffenen Stelle. In der Quelltextansicht des an den Browser übertragenen HTML-Dokuments steht dann beispielsweise:

```
<meta http-equiv="expires" content="Mon, 07 Mar 2005 10:06:37 GMT">
```

Die Zuweisung an das content-Attribut wurde dabei von PHP ermittelt. Das Script erzeugt als Ausgabe einen formatgerechten String, der den Zeitpunkt genau drei Tage nach Aufruf der Seite darstellt. Auf diese Weise erhält die Meta-Angabe zum Cache-Verfallszeitpunkt eine relative Angabe, nämlich »ignoriere das Dokument im Cache, wenn es dort älter als 3 Tage ist«, aber die Zeitangabe selbst ist eine absolute.

Ermittelt wird die gewünschte Ausgabe mithilfe der PHP-Funktionen `mktime()` und `gmdate()`. Auf deren Eigenheiten und die Bedeutung ihrer Parameter gehen wir in ***Abschnitt 12.4.4*** näher ein. An dieser Stelle soll ein kurzer Abriss dessen genügen, was innerhalb des Scripts passiert: In einer Variablen namens `$in_3_days` wird der Rückgabewert der Funktion `mktime()` gespeichert. Variablen in PHP sind immer an dem führenden Dollarzeichen `$` zu erkennen. Die Parameter, die `mktime()` übergeben werden, bewirken, dass die Funktion den Zeitpunkt »in drei Tagen, von jetzt an gerechnet« zurückgibt. Den Zeitpunkt benötigen wir, um ihn der Funktion `gmdate()` zu übergeben. Mithilfe von `gmdate()` kann ein Zeitpunkt in Bestandteile wie Wochentag, Monat, Jahr, Uhrzeit usw. zerlegt werden. Die Funktion ermöglicht es, eine formatierte Ausgabe in gewünschter Form zu erzeugen. Das Ergebnis, das die Funktion zurückgibt, wird mit `echo` ausgegeben. »Ausgegeben« heißt, es wird im HTML-Code, der an den Browser gesendet wird, an der Stelle eingefügt, wo der PHP-Bereich notiert ist.

PHP für dynamisch erzeugten Text: die Angaben zu den Bildern

Die Vorschaubilder im Beispiel stehen in einer unsichtbaren Tabelle, links die Vorschaubilder und rechts davon die Informationen zum Bild. Bei jedem der drei Bilder im Beispiel ist in der rechten Tabellenzelle ein gleichartiger PHP-Bereich notiert. Hier noch mal der Ausschnitt für das erste Bild:

```
<b>Bild 1:</b><br>
<?php
$bytes = filesize("bild01.jpg");
$pixel = GetImageSize("bild01.jpg");
echo "Breite &#215; Höhe: ",
     $pixel[0]," &#215; ",$pixel[1]," Pixel<br>";
echo "Größe: ",round($bytes / 1024)," KByte";
?>
```

Die erste Zeile ist noch statisches HTML. Danach folgt der PHP-Bereich. Die Bedeutung der ersten Anweisung ist leicht erratbar: Mithilfe der PHP-Funktion `filesize()` wird die Dateigröße der Grafik ermittelt, die angezeigt werden soll, wenn der Anwender auf das Vorschaubild klickt. Der Rückgabewert der Funktion, die Dateigröße in Byte, wird in einer Variablen namens `$bytes` gespeichert.

Die nächste Anweisung ist nach dem gleichen Schema aufgebaut: In einer Variablen `$pixel` wird der Rückgabewert der PHP-Funktion `GetImageSize()` gespeichert. Auch dieser Funktion wird wieder die Datei der Grafik übergeben, die bei Anklicken des Vorschaubilds angezeigt werden soll. Im Unterschied zu `filesize()` gibt `GetImageSize()` jedoch keinen einzelnen Wert zurück, sondern gleich mehrere Werte – in einem Array. In `$pixel` ist also am Ende ein Array gespeichert.

Damit sind alle gewünschten Daten ermittelt und sie können ausgegeben werden. Von den Werten, die in `$pixel` gespeichert sind, interessieren uns nur die ersten beiden (Breite und Höhe). Diese geben wir mit `$pixel[0]` und `$pixel[1]` zusammen mit anderem HTML-Text aus. Auch die Dateigröße wird ausgegeben, allerdings nicht wie zunächst ermittelt in Byte, sondern in Kilobyte. Um den in `$bytes` gespeicherten Wert in Kilobyte auszudrücken, muss er durch 1024 dividiert werden, weil ein Kilobyte aus 1024 Byte besteht. Um sicherzustellen, dass eine Ganzzahl angezeigt wird, übergeben wir die Rechnung an die PHP-Funktion `round()`.

Dieses Beispiel zeigt, wie viele fertige Funktionen zu PHP gehören, die man sonst in Programmiersprachen oft vergeblich sucht bzw. aus Basisfunktionen selbst programmieren oder aus zusätzlichen Code-Bibliotheken besorgen muss. Hinter `GetImageSize()` etwa steckt eine Menge Intelligenz, nämlich die Header-Daten verschiedener Grafikformate auslesen und daraus bestimmte Werte ermitteln zu können.

PHP für die Ausgabe von globalen Daten: die Ausgabe des Servernamens

Am Ende unseres Beispiels ist noch ein kleiner PHP-Bereich notiert, in dem mit `$_SERVER['SERVER_NAME']` der Hostname des Rechners ausgegeben wird, auf dem der Webserver läuft. Diese Variable gehört zu den vordefinierten und global verfügbaren Variablen in PHP. Es gibt eine ganze Reihe solcher Variablen, unter anderem sämtliche Umgebungsvariablen des Webservers. Aber auch GET-Strings und POST-Daten kann das Script über solche Variablen ermitteln. Es handelt sich um die so genannten ***Superglobals***, ein Set von Arrays, in denen Umgebungsdaten gespeichert sind.

Fazit: PHP in HTML ist praktisch

Das Beispiel zeigt, warum PHP so beliebt ist. Ohne viel Aufwand lassen sich HTML-Dokumente aufpeppen oder optimieren. Die Strukturierung einer Website bleibt die gleiche wie beim Arbeiten mit statischen HTML-Dateien. Es gibt eine Reihe von Dateien, die untereinander verlinkt sind.

12.2.2 HTML in PHP

Solange nur »PHP in HTML« angewendet wird, ist kein strukturelles Umdenken nötig. Das ändert sich jedoch, wenn man PHP stattdessen in der Form »HTML in PHP« einsetzt. Während Einsteiger eher die Variante »PHP in HTML« bevorzugen, kommt bei größeren und professionellen Sites eher die Variante »HTML in PHP« zum Einsatz. Das strukturelle Umdenken besteht darin, dass ein PHP-Script bei der Variante »HTML in PHP« in der Regel ein Behälter für mehrere HTML-Dokumente ist, die auch unterschiedliche URIs haben.

Ein Beispiel soll dies demonstrieren. Dabei setzen wir unser Beispiel mit den Vorschaubildern fort. In den Links aller drei anklickbaren Vorschaugrafiken wurde dort die gleiche PHP-Datei aufgerufen, nur mit unterschiedlichem GET-String:

```
<a href="bild.php?bild=01">
<img src="bild01thumb.jpg" class="thumb_h"
     alt="Bild 1 Vorschau"></a>
<a href="bild.php?bild=02">
<img src="bild02thumb.jpg" class="thumb_v"
     alt="Bild 2 Vorschau"></a>
<a href="bild.php?bild=01">
<img src="bild03thumb.jpg" class="thumb_h"
     alt="Bild 1 Vorschau"></a>
```

Die verlinkte Datei ***bild.php*** fungiert also offensichtlich als eine Art »Allgemeindokument«, das in der Lage ist, verschiedene Bilder auszugeben. Nachfolgendes Listing enthält den vollständigen Quelltext von ***bild.php***:

Listing 12.3: HTML in PHP

```
<?php
$pic_number = (int) $_GET['bild'];
$pic_file = "bild".$_GET['bild'].".jpg";
$pixel = $pixel = GetImageSize($pic_file);
$css_width = $pixel[0]."px";
$css_height = $pixel[1]."px";
$csv = fopen("bilder.csv", "r");
while(!feof($csv))
  $csv_lines[] = fgets($csv);
fclose ($csv);
for($i = 0; $i < count($csv_lines); $i++) {
   $fields[$i] = explode(";", $csv_lines[$i]);
}
echo <<<END_1
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Bild $pic_number</title>
<style type="text/css">
```

```
td.left {
  vertical-align:top;
  padding-right:15px;
}
td.right {
  vertical-align:top;
}
h1 {
  border-bottom:1px solid black;
}
h3 {
  margin-top:0;
}
</style>
</head>
<body>
<h1>Bild $pic_number</h1>
<table>
<tr>
<td class="left">
<a href="bilder.php"><img src="$pic_file"
style="width:$css_width; height:$css_height;
border:solid 1px black" alt="Bild $pic_number"></a></p>
<p><b>zurück:</b> <a href="bilder.php">Vorschaubilder</a></p>
</td>
<td class="right">
<h3>Daten zum Bild:</h3>
<p>
<b>Aufnahmedatum:</b> {$fields[$pic_number][1]} <br>
<b>Aufnahmeort:</b> {$fields[$pic_number][2]} <br>
<b>Bildautor:</b> {$fields[$pic_number][3]} <br>
<b>Beschreibung:</b> {$fields[$pic_number][4]} <br>
</p>
</td>
</tr>
</table>
</body>
</html>
END_1;
?>
```

Im Beispielscript ist der gesamte Inhalt in einen großen PHP-Bereich eingeschlossen. Es gibt also gar keinen »HTML-Modus« mehr in dieser Datei, sondern nur noch Script-Anweisungen.

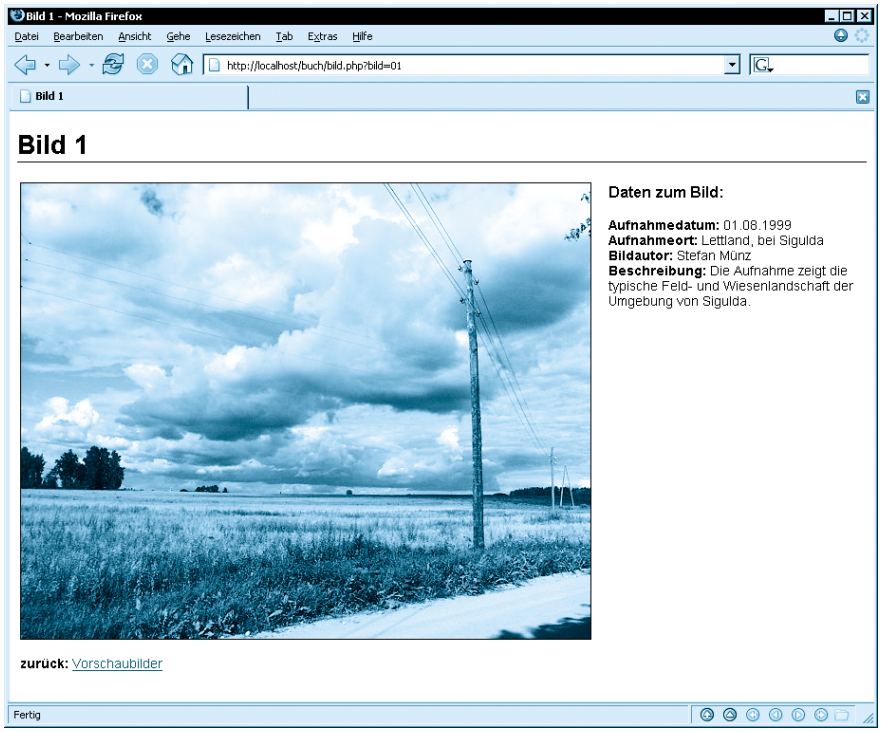


Abbildung 12.5: Beispielhafte Browser-Ausgabe des PHP-Scripts

Für größere HTML-Passagen: here-Dokumente

Ein großer Teil des Beispielcodes besteht jedoch aus durchgängig notiertem HTML. Das liegt daran, dass mithilfe der bereits bekannten Anweisung `echo` diesmal ein so genanntes *here-Dokument* ausgegeben wird. Die Syntax dazu lautet:

```
echo <<<MARKE
<!-- viel HTML -->
MARKE;
```

Dabei ist `MARKE` ein frei wählbarer Name. Wichtig sind die drei öffnenden spitzen Klammern vor dem Namen. Alles, was dann folgt, ist normaler HTML-Code, bis in einer Zeile alleinstehend der Name der Marke steht, abgeschlossen von einem Semikolon. Der HTML-Code eines here-Dokuments kann auch PHP-Variablen enthalten – der PHP-Interpreter erkennt diese und gibt an der entsprechenden Stelle den Wert der Variablen aus. Das here-Dokument in unserem Beispiel enthält an einigen Stellen Variablen.

`echo` ist übrigens keine PHP-Funktion. Deshalb wird der übergebene Inhalt auch nicht wie bei Funktionen sonst üblich in runden Klammern notiert. Der Inhalt kann entweder eine Zeichenkette sein, z. B.:

```
echo "<h1>Willkommen</h1>";
```

Die Zeichenkette darf auch Variablen enthalten:

```
echo "<h1>Willkommen, $name</h1>";
```

Bei Array-Variablen oder anderen komplexen Variablen müssen diese allerdings in geschweifte Klammern eingeschlossen werden:

```
echo "<h1>Willkommen, {$name[23]}</h1>";
```

Die andere Möglichkeit ist eine kommagetrennte Liste aus Zeichenketten und Variablen:

```
echo "<h1>Willkommen, ", $name[23], "</h1>";
```

Oder eben ein here-Dokument: Innerhalb von here-Dokumenten gelten die gleichen Regeln wie bei der Notation in einer Zeichenkette, d.h., einfache Variablen wie `$name` können normal notiert werden. Komplexe Variablen wie `$name[23]` müssen mit geschweiften Klammern notiert werden, also in der Form `{$name[23]}`.

Erst Datenverarbeitung, dann Datenausgabe

Unser Beispielscript hat einen typischen Aufbau. Im oberen Teil werden Daten ermittelt und für die Ausgabe vorbereitet, im unteren Teil erfolgt die Ausgabe.

Das Script bekommt beim Aufruf Daten mittels GET-String übergeben. Im HTML-Code der Vorschaubilder sah das so aus:

```
<a href="bild.php?bild=02">
<img src="bild02thumb.jpg" class="thumb_v"
     alt="Bild 2 Vorschau"></a>
```

Über die Superglobalvariable `$_GET` kann ein Script auf Daten zugreifen, die auf diesem Weg übergeben wurden. `$_GET` ist ein assoziativer Array. Deshalb kann das Beispielscript mit `$_GET['bild']` direkt auf ein Name-Wert-Paar wie `bild=02` zugreifen. In `$_GET['bild']` ist dann etwa der Wert `"02"` gespeichert.

Die ersten fünf Anweisungen in ***bild.php*** definieren verschiedene Variablen und weisen ihnen einen ermittelten Inhalt zu. Vieles davon sollte Ihnen bereits aus der ***bilder.php*** des vorhergehenden Abschnitts bekannt sein. Unser Augenmerk soll dem Teil gelten, der daran anschließend folgt. Die Textdaten, die im Browser neben dem Bild angezeigt werden, kommen nämlich aus einer externen Datei. Diese hat folgenden Inhalt:

```
picture_file;record_date;record_place;record_author;record_text
bild01.jpg;01.08.1999;Lettland, bei Sigulda;Stefan Münz;Die Aufnahme zeigt die
typische Feld- und Wiesenlandschaft der Umgebung von Sigulda.
bild02.jpg;02.08.1999;Lettland, Schloss Sigulda am Gauja-Fluss;Stefan Münz;Die
Aufnahme zeigt die Seitenansicht des Schlosses.
bild03.jpg;09.08.1999;Estland, Tartu;Stefan Münz;Die Aufnahme zeigt die zentrale
Einkaufsstraße der Stadt.
```

Es handelt sich also um eine typische kommagetrennte Datendatei, wie sie beispielsweise in Programmen wie Excel leicht erzeugbar ist. Jede Zeile (hier leider aus satztechnischen Gründen umgebrochen) stellt einen »Datensatz« dar. Trennzeichen für Felder der Datensätze ist das Semikolon. Die erste Zeile in der Datei weist die Felder aus. Diese Datei haben wir unter dem Namen ***bilder.csv*** abgespeichert. In PHP wird die Datei eingelesen. Hier noch mal der dazu relevante Code-Ausschnitt:

```
$csv = fopen("bilder.csv", "r");
while(!feof($csv))
  $csv_lines[] = fgets($csv);
fclose ($csv);
```

Auf die Funktionen zum Lesen aus und Schreiben in Dateien gehen wir in ***Abschnitt 12.4.1*** noch näher ein. Hier soll es genügen, zu verstehen, dass die Datei mit `fopen()` geöffnet wird. Dann wird sie so lange (`while`) zeilenweise mit der Funktion `fgets()` eingelesen, bis das Dateiende `feof()` erreicht ist, um schließlich mit `fclose()` geschlossen zu werden. Am Ende gibt es einen Array namens `$csv_lines`, bei dem in jedem Element eine Zeile der eingelesenen Datei gespeichert ist.

Damit haben wir zwar die Datensätze schon sauber getrennt, doch noch nicht die einzelnen Datenfelder. Das geschieht in der anschließenden for-Schleife:

```
for($i = 0; $i < count($csv_lines); $i++) {
   $fields[$i] = explode(";", $csv_lines[$i]);
}
```

Für jedes Element von `$csv_lines` wird dessen Inhalt mithilfe der PHP-Funktion `explode()` auseinander genommen, und zwar an Hand des vereinbarten Trennzeichens (`;`). Am Ende der for-Schleife gibt es einen verschachtelten Array namens `$fields`. Jedes Element dieses Arrays stellt zwar wie in `$csv_lines` einen Datensatz dar, doch ist jedes Element diesmal selbst ein untergeordneter Array, bestehend aus den jeweiligen Datenfeldern der Zeile.

Im Ausgabeteil des here-Dokuments ist zu sehen, wie die Daten durch Zugriff auf diesen Array ausgegeben werden. Der relevante Ausschnitt:

```
<b>Aufnahmedatum:</b> {$fields[$pic_number][1]} <br>
<b>Aufnahmeort:</b> {$fields[$pic_number][2]} <br>
<b>Bildautor:</b> {$fields[$pic_number][3]} <br>
<b>Beschreibung:</b> {$fields[$pic_number][4]} <br>
```

Die doppelten eckigen Klammern hinter `$fields` signalisieren den »doppelten Array«. In der Variablen `$pic_number` ist die Nummer des anzuzeigenden Bildes gespeichert. Die CSV-Datei ist so sortiert, dass dadurch gleich der Zugriff auf den richtigen Datensatz möglich ist. Angenommen, in `$pic_number` ist 1 gespeichert. Dann ist `$fields[$pic_number][1]` das Gleiche wie `$fields[1][1]` und greift auf das Datum `01.08.1999` aus den CSV-Daten zu. Grund ist die 0-basierte Zählweise von Arrays in PHP, die in dieser Hinsicht denen von JavaScript gleichen.

Fazit: Umdenken lohnt sich

Die Abkehr von dem Denken »eine PHP-Datei pro Webseite«, welche typisch ist für die einfachere Programmierweise »PHP in HTML«, hat unbestreitbare Vorteile: Der PHP-Code wird zentralisiert und so organisiert, dass er für eine beliebige Menge von Webseiten wie ein Container ist. Dadurch wird der Code leichter wartbar und muss nicht mehr in vielen Dateien wiederholt werden.

12.2.3 PHP mit Einsatz von HTML-Templates

Der Buchteil über HTML und CSS hat gezeigt, wie sich durch gezielten Einsatz von CSS Inhalt und Layout von Webseiten sinnvoll trennen lassen. Mit PHP können wir noch ein Stück weiter gehen: Wir trennen nicht nur HTML-Struktur und CSS, sondern auch HTML-Grundgerüst, Navigation und Seiteninhalte. Dadurch erreichen wir eine noch sauberere Trennung der Bereiche mit erheblichen Vorteilen:

- Das HTML-Grundgerüst, das ja zumindest die Basiselemente für das Layout mit CSS enthält, muss ebenso wie die CSS-Definitionen nur ein einziges Mal notiert werden, und nicht, wie bei statischen Seiten, in jeder einzelnen HTML-Datei. Selbst bei radikalen Layout-Relaunches müssen dann nur noch die HTML-Grundgerüstdatei und die CSS-Datei bearbeitet werden.
- Die Navigation muss ebenfalls nur noch an einer Stelle notiert werden und nicht mehr in jeder einzelnen Seite.
- Die eigentlichen Seiteninhalte bestehen aus schlichtem HTML und können durch kleine Änderungen im PHP-Script aus ganz variablen Quellen stammen. In unserem Beispiel wird es für jede Seite eine Inhaltsdatei geben. Ebenso gut könnten die Inhalte aber aus einer Datenbanktabelle, aus einer XML-Datei oder aus einer anderen Quelle stammen.

Auch im Template-Beispiel werden wir anstelle von statischen URIs wie schon im Abschnitt zuvor wieder mit dynamischen URIs arbeiten, d.h. also den anzuzeigenden Seiteninhalt aus dem übergebenen GET-String ermitteln. Eine einzige PHP-Datei ist der »Container« für sämtliche möglichen Inhalte.

Insgesamt benötigen wir also folgende Dateien:

- Eine Datei mit dem HTML-Grundgerüst ohne Inhalte,
- eine Datei mit dem HTML-Code der Navigation,
- eine CSS-Datei für zentrale Layout- und Formatierungsdefinitionen,
- je eine Datei für den Inhalt jeder Seite,
- ein PHP-Script, das alles zusammenklebt und die gewünschten Inhalte ausgibt.

Unser Template-Beispiel greift auf das Beispiel aus ***Abschnitt 5.1.3*** (»Winkel-Layout«) zurück.

HTML-Templates: Grundgerüst und Navigation

Die Datei mit dem HTML-Grundgerüst nennen wir im Beispiel *linkweb.tpl*. Der Dateiname ist natürlich frei wählbar. Die Endung *.tpl* vergeben wir einfach als Abkürzung für »Template«. Der Quelltext lautet:

Listing 12.4: Template-Datei mit HTML-Grundgerüst und [%Variablen%]

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>[%title%]</title>
<link rel="stylesheet" type="text/css" href="linkweb.css">
</head>
<body>
<div id="top_section">
<img src="logo.gif" id="logo" alt="Logo">
<div id="ticker"></div>
<form id="search" action="search.php" method="get">
<span id="search_label">Suche nach:</span>
<input type="text" id="search_text" name="search_text" value="Suche">
<input type="submit" src="ok.gif" id="search_button" value="OK">
</form>
</div>
<div id="main_section">
<div id="navigation">
[%navigation%]
</div>
<div id="content">
[%content%]
</div>
</div>
</body>
</html>
```

Es handelt sich um eine gewöhnliche HTML-Datei, welche das HTML-Grundgerüst mit den nötigen Kopfdaten enthält sowie die div-Bereiche, die für CSS zur optischen Bereichsverteilung im Browser dienen. Auffällig sind die Zeichenfolgen wie z. B. `[%content%]`. Dies sind selbst definierte Platzhalter, die wir später in PHP durch ermittelte Inhalte ersetzen werden.

Noch trivialer ist die Template-Datei für die Navigation – wir nennen sie *linkweb-nav.tpl*:

Listing 12.5: Template-Datei mit HTML-Code für die Navigation

```
<a class="navi" href="linkweb.php?page=home">
HOME</a>
<a class="navi" href="linkweb.php?page=impressum">
Impressum</a>
```

```
<a class="navi" href="linkweb.php?page=themen">
Themen</a>
<a class="navi" href="linkweb.php?page=branchen">
Branchen</a>
<a class="navi" href="linkweb.php?page=auskunft">
Auskunft</a>
<a class="navi" href="linkweb.php?page=geo">
Länder und Städte</a>
<a class="navi" href="linkweb.php?page=wissen">
Wissenschaft</a>
<a class="navi" href="linkweb.php?page=literatur">
Literatur</a>
<a class="navi" href="linkweb.php?page=lexika">
Lexika/Wörterbücher</a>
<a class="navi" href="linkweb.php?page=glossare">
Glossare</a>
<a class="navi" href="linkweb.php?page=wikis">
Wikis</a>
<a class="navi" href="linkweb.php?page=ftp">
FTP-Verzeichnisse</a>
<a class="navi" href="linkweb.php?page=wais">
WAIS-Services</a>
<a class="navi" href="linkweb.php?page=newsgroups">
Newsgroups</a>
<a class="navi" href="linkweb.php?page=foren">
Web-Foren</a>
```

Die Datei enthält ausschließlich die Hyperlinks der Navigation. Die Formatierung der Navigationsleiste, welche aus diesen Links gebildet wird, wird in der CSS-Datei vorgenommen. Deren Quelltext sparen wir uns jedoch, weil sie nichts enthält, was im Zusammenhang mit dem Funktionieren des Beispiels von Bedeutung ist.

Alle Navigationslinks rufen das zentrale, als »Container« fungierende PHP-Script `linkweb.php` auf, und zwar jeweils mit einem GET-String, in dem ein Parameter namens `page` einen jeweils unterschiedlichen Wert erhält.

Datendatei: schlicht und einfach

Zum besseren Verständnis noch der Quelltext einer Datendatei, also einer Datei mit eigentlichen Seiteninhalten. Wie erwähnt speichern wir die Seiteninhalte in unserem Beispiel in jeweils eigenen Dateien, die wir *0001.txt*, *0002.txt*, *0003.txt* usw. benennen. Die Inhalte könnten aber auch in anderer Form gespeichert werden. Der Inhalt der *0001.txt* lautet beispielsweise:

Listing 12.6: HTML-Code einer Datendatei

```
<h1>LinkWeb</h1>
<p>Bevor es große Suchmaschinen wie Google gab, waren es Linkverzeichnisse,
welche die zentrale Vernetzung im Web besorgten. Auch heute noch, im Zeitalter
der Suchmaschinen, gibt es nach wie vor gepflegte Linkverzeichnisse, meist auf
```

```
einen bestimmten Themenbereich beschränkt. Dazu kommen durchsuchbare
Spezialverzeichnisse, etwa für Adressen und Telefonnummern, aber auch für
Internet-Inhalte jenseits des World Wide Web.</p>
<p><b>LinkWeb</b> ist das Verzeichnis solcher Verzeichnisse.</p>
<h2>Die Kategorien</h2>
<ul>
<li>Themenverzeichnisse sind Linkverzeichnisse zu bestimmten Fachgebieten.</li>
<li>Branchenverzeichnisse sind nach Branchen sortierte Verzeichnisse von
Unternehmen und Geschäften.</li>
<li>Auskunft-Services sind durchsuchbare Verzeichnisse wie Telefonbücher.</li>
<li>usw.</li>
</ul>
```

Datendateien dieser Art enthalten nichts als strukturelles HTML für die darzustellenden Inhalte. Keine Spur von Layout und auch das HTML-Grundgerüst mit Kopfdaten usw. ist nicht nötig, da es ja in eine Template-Datei ausgelagert wurde. Wichtig ist für unser Beispiel nur eine Konvention: In der ersten Zeile einer Datendatei muss eine h1-Überschrift mit dem Titel der Seite notiert sein.

Das PHP-Script: Zusammenkleben und Ausgeben der Inhalte

Die zuvor behandelten Listings lassen bereits erkennen, welche Aufgaben das zentrale PHP-Script hat:

1. Es muss die Template-Datei mit dem HTML-Grundgerüst einlesen.
2. Es muss die Datei mit der HTML-Navigation einlesen.
3. Es muss den Code der in Schritt 2 eingelesenen Datei innerhalb des Codes der in Schritt 1 eingelesenen Datei dort einfügen, wo der Platzhalter `[%navigation%]` notiert ist.
4. Es muss den übergebenen GET-String auswerten, um herauszufinden, welche Seite angezeigt werden soll. Dazu muss es natürlich auch eine Liste möglicher Seiten kennen oder wissen, wie es an die Daten von Seiteninhalten herankommt.
5. Es muss den Inhalt der gewünschten Datendatei einlesen und dann an der Stelle einfügen, wo der Platzhalter `[%content%]` notiert ist.
6. Es muss der vereinbarten Konvention gemäß die h1-Überschrift der ersten Zeile der Datendatei auslesen und ihren Textinhalt dort einfügen, wo der Platzhalter `[%title%]` notiert ist. Das ist im `title`-Element in den Kopfdaten des Grundgerüsts der Fall.
7. Es muss die so zusammengefügte Seite ausgeben.

Der Quelltext des PHP-Scripts lautet:

Listing 12.7: Zentrales PHP-Script zum Zusammenfügen und Ausgeben der Inhalte

```
<?php
#-----------------------------------------
# Templates:
$layout_template = "linkweb.tpl";
$navigation_template = "linkweb-nav.tpl";
```

```
#-----------------------------------------
# Inhaltsdateien:
$content_files = array();
$content_files['home'] = "0001.txt";
$content_files['impressum'] = "0007.txt";
$content_files['themen'] = "0003.txt";
$content_files['branchen'] = "0009.txt";
$content_files['auskunft'] = "0005.txt";
$content_files['geo'] = "0002.txt";
$content_files['wissen'] = "0004.txt";
$content_files['literatur'] = "0008.txt";
$content_files['lexika'] = "0006.txt";
$content_files['glossare'] = "0010.txt";
$content_files['wikis'] = "0014.txt";
$content_files['ftp'] = "0011.txt";
$content_files['wais'] = "0015.txt";
$content_files['newsgroups'] = "0012.txt";
$content_files['foren'] = "0013.txt";
#=========================================
# Templates einlesen:
$layout_content = file_get_contents($layout_template);
$navigation_content = file_get_contents($navigation_template);
#-----------------------------------------
# Seite aus Templates zusammenfügen:
$page = $layout_content;
$page = preg_replace("/\[\%navigation\%\]/",
                     $navigation_content, $page);
#-----------------------------------------
# Inhalt seitenabhängig einlesen:
$get_page = "";
if(isset($_GET['page']))
   $get_page = $_GET['page'];
else
   $get_page = "home";
$content_lines = file($content_files[$get_page]);
$content = implode("", $content_lines);
#-----------------------------------------
# Inhalt in Seite einfügen:
$page = preg_replace("/\[\%content\%\]/", $content, $page);
#-----------------------------------------
# Titel ermitteln und einfügen:
preg_match("/<h1>(.*)<\/h1>/", $content_lines[0], $matches);
$page_title = strip_tags($matches[0]);
$page = preg_replace("/\[\%title\%\]/", $page_title, $page);
#-----------------------------------------
# Fertige Seite ausgeben:
echo $page;
?>
```

Das PHP-Script enthält zur besseren Übersicht Kommentarzeilen, die auch zur optischen Trennung von Bereichen im Script genutzt werden. Kommentare in PHP können unter anderem mit dem Gatterzeichen (#) beginnen.

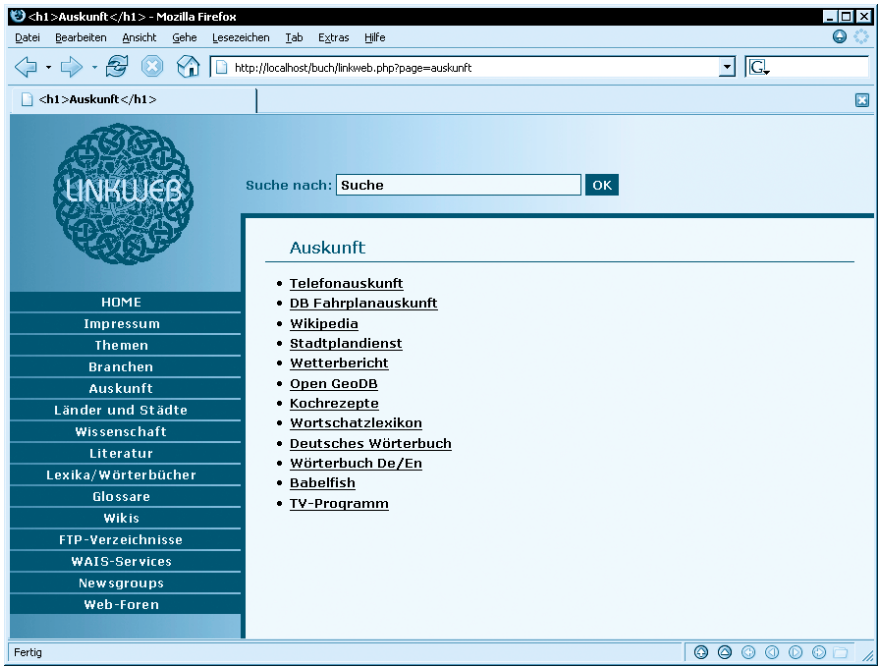


Abbildung 12.6: Das Linkweb-Script in Aktion, hier mit der Seite »Auskunft«

Vom Aufbau her betrachtet, ist in diesem Beispiel ein Script zu sehen, das nicht einmal mehr dem Typus »HTML in PHP« entspricht, sondern eher wie ein reines Programm ohne Bezug zu HTML aussieht. Dennoch tut es letztlich nichts anderes, als den HTML-Code einer Webseite zu erzeugen.

Zunächst werden einfach die beiden Namen der Template-Dateien für Grundgerüst und Navigation in den Variablen `$layout_template` und `$navigation_template` gespeichert. Auch der nächste Abschnitt besteht aus Definitionen. Ein Array namens `$content_files` wird erzeugt. Mittels Zuweisung von Schlüsselnamen und Dateinamen als Werten merkt sich das Script die einzelnen Datendateien. Die Schlüsselnamen wie etwa `home` in `$content_files['home'] = "0001.txt"` sind übrigens die gleichen, die auch beim GET-Parameter `page` übergeben werden können. Wenn dem Script also beispielsweise im GET-String `page=auskunft` übergeben wird, so kann es mit `$content_files['auskunft']` ohne Umwege die zugehörige Datendatei ermitteln.

Im Script-Abschnitt »Templates einlesen« werden die beiden Dateien mit dem HTML-Grundgerüst und mit dem HTML-Code für die Navigation eingelesen. Dazu verwenden wir die praktische PHP-Funktion `file_get_contents()`, die das sonst übliche Öffnen, Einlesen und Schließen der Datei in einem Schritt erledigt. Der gesamte eingelesene Dateiinhalt befindet sich anschließend in der Variablen, welcher der Funktionsaufruf zugewiesen ist. Im Beispiel steht also in `$layout_content` anschließend der komplette Inhalt der Datei mit dem HTML-Grundgerüst, und in `$navigation_content` der Inhalt der Datei mit dem HTML-Code für die Navigation.

Das »Zusammenkleben« der Inhalte beginnt im Script-Abschnitt »Seite aus Templates zusammenfügen«: Dort wird zunächst der Inhalt aus `$layout_content`, also der HTML-Code des Grundgerüsts, in eine neue Variable `$page` kopiert. Dies wird die Arbeitsvariable, die so versorgt wird, dass sie am Ende den kompletten HTML-Code der Seite enthält und ausgegeben werden kann.

Mit der PHP-Funktion `preg_replace()` lassen sich Zeichenketten suchen und ersetzen. In unserem Beispiel nutzen wir die Funktion zum Ersetzen der Platzhalterzeichenfolge `[%navigation%]`, die ja zum eingelesenen HTML-Grundgerüstcode gehört und folglich nun auch im Inhalt der Variablen `$page` vorkommt, durch den tatsächlichen HTML-Code der Navigation. Dieser ist ja in der Variablen `$navigation_content` gespeichert.

Der erste an `preg_replace()` übergebene Parameter lautet im Beispiel `/\[\%navigation\%\]/`, obwohl die zu suchende Zeichenkette doch eigentlich nur `[%navigation%]` lautet. Das etwas seltsame Konstrukt ist ein so genannter regulärer Ausdruck. Reguläre Ausdrücke benutzen Sonderzeichen, um Muster (Regularien) zu definieren, die auf eine oder auch mehrere Zeichenketten zutreffen. Die Funktion `preg_replace()` erwartet beim ersten Parameter das, wonach gesucht werden soll, in Form eines regulären Ausdrucks. Da reguläre Ausdrücke ein wichtiges, auch in ganz anderen Zusammenhängen immer wieder benötigtes Hilfsmittel sind, gehen wir in ***Abschnitt 12.3.9*** gesondert darauf ein.

Nach diesem Suche-Ersetze-Vorgang ist in `$page` bereits der HTML-Code des Seitengrundgerüsts inklusive des HTML-Codes mit den Navigationslinks gespeichert. Fehlt nur noch der Seiteninhalt. In diesem Zusammenhang muss das PHP-Script erst einmal ermitteln, welche Seite angezeigt werden soll.

Dazu wird die superglobale Variable `$_GET` ausgelesen, über welche PHP einem Script eventuell übergebene GET-Parameter zur Verfügung stellt. Da unser vereinbarter Parameter `page` heißt, fragen wir mit `if(isset($_GET['page']))` ab, ob ein Parameter `page` übergeben wurde. Falls ja, wird in der Variablen `$get_page` der Wert des übergebenen `page`-Parameters gespeichert. Falls nicht, muss das Script ja auch irgendwie reagieren. Es reagiert so, dass `$get_page` in diesem Fall den Wert `home` erhält. Das ist der Wert, den das Script übergeben bekommen würde, wenn die Startseite angezeigt werden soll. Im Klartext: Wenn die Script-Datei ohne `page`-Parameter aufgerufen wird, sorgt es dafür, dass die Startseite des Webangebots angezeigt wird.

Über `$content_files[$get_page]` lässt sich dann die zugehörige Datendatei ermitteln, da `$content_files` ja ein assoziativer Array mit den entsprechenden Schlüsselnamen ist. Wird also etwa als Wert des `page`-Parameters `auskunft` ermittelt und in `$get_page` gespeichert, dann ist `$content_files[$get_page]` so viel wie `$content_files['auskunft']`.

Die Datendatei wird etwas umständlicher eingelesen:

```
$content_lines = file($content_files[$get_page]);
$content = implode("", $content_lines);
```

Die PHP-Funktion `file()` ist wieder eine von den praktischen Funktionen, die uns das Öffnen, Lesen und Schließen der gewünschten Datei erspart. Die angegebene Datei wird

einfach direkt eingelesen, jedoch anders als bei `file_get_contents()` nicht in eine einzige Zeichenkette, sondern zeilenweise, wobei ein Array der einzelnen Dateizeilen zurückgegeben wird. In der zweiten oben gelisteten Anweisung werden die einzelnen Zeilen dann mit der PHP-Funktion `implode()` wieder zu einer einzigen Zeichenkette zusammengefügt.

Dass wir nicht gleich `file_get_contents()` verwendet haben, liegt daran, dass wir nun noch den Array `$content_lines` mit den einzelnen Zeilen haben. Diesen brauchen wir nämlich, da wir ja aus der ersten Datenzeile den Überschriftentext der dort erwarteten h1-Überschrift extrahieren und ins title-Element einsetzen sollen.

Bevor das geschieht, wird der eingelesene Inhalt der Datendatei jedoch erst einmal wieder in die am Ende auszugebende Variable `$page` eingefügt. Dazu wird wieder die Funktion `preg_replace()` verwendet. Eingesetzt wird der Inhalt der Datendatei an der Stelle, wo der Platzhalter `[%content%]` notiert ist.

Im vorletzten Abschnitt »Titel ermitteln und einfügen« extrahiert das Script dann noch wie schon erwähnt aus der ersten Zeile der Datendatei, gespeichert in `$content_lines[0]`, die h1-Überschrift, genauer, deren Elementinhalt. Diesen speichert sie in der Variablen `$page_title` zwischen und setzt deren Inhalt schließlich dort in `$page` ein, wo der Platzhalter `[%title%]` notiert ist (Elementinhalt des title-Elements).

Die Ausgabe der gesamten Seite ist dann nur noch eine einzige kleine echo-Anweisung, die am Ende des Scripts steht.

Fazit: Teilen und Herrschen ist professionell

Das Script ***linkweb.php*** ist vom grundsätzlichen Aufbau her ein Beispiel dafür, wie PHP in professionellen Kreisen eingesetzt wird. PHP wird hierbei ausschließlich zum Interpretieren von PHP-Code benutzt und muss nicht dauernd zwischen Interpreter-Modus und HTML-Modus hin- und herschalten. Der PHP-Code ist auch nicht voll von echo-Anweisungen, mit deren Hilfe direkt HTML-Code erzeugt wird. Das Script selber fügt nur aus Template-Dateien mit selbst definierten Platzhaltern und Datendateien dynamisch den Inhalt einer anzuzeigenden Webseite zusammen. Das Script erlaubt nicht nur die Trennung zwischen Markup und Layout, sondern auch die Trennung zwischen Navigation und Inhalt.

12.2.4 Formularverarbeitung und Mail-Funktion mit PHP

In ***Abschnitt 11.3.3*** stellten wir einen Form-Mailer vor, der in Perl geschrieben ist und auf dem Webserver als CGI-Script läuft. Das gleiche Script wollen wir nun in PHP realisieren. Dabei gehen wir wie folgt vor:

- Der HTML-Code des Feedback-Formulars, das als Beispiel dient, wird in einer Datei namens ***feedback.htm*** abgelegt.
- Der HTML-Code der Danke-Seite, die angezeigt wird, wenn die Formulardaten erfolgreich ausgewertet und versendet wurden, wird in einer Datei namens ***danke.htm*** abgelegt.

- Ein PHP-Script verwaltet das ganze Feedback-Handling, d.h., es zeigt das Formular an, prüft Eingaben auf fehlende Daten und versendet, wenn alles in Ordnung ist, dem Seitenanbieter eine Mail mit den Formulardaten.

Die HTML-Dateien: nicht für den direkten Aufruf bestimmt

Die Dateien *feedback.htm* und *danke.htm* sollen nicht vom Browser direkt aufgerufen werden. In unserem Beispiel werden wir sie zwar der Einfachheit halber im gleichen Verzeichnis wie das PHP-Script ablegen, doch in der Praxis ist es besser, solche Dateien gleich außerhalb der Document Root abzulegen, damit sie für Webclients gar nicht erst erreichbar sind. Die beiden HTML-Dateien enthalten nichts Ungewöhnliches bis auf zwei Platzhalter der Marke Eigenbau, wie wir sie im vorangegangenen Abschnitt bereits verwendet haben:

Listing 12.8: HTML-Datei feedback.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Feedback</title>
<link rel="stylesheet" type="text/css" href="styles.css">
</head><body>
<h1>Feedback</h1>
<form action="[%self%]" method="post">
<input type="hidden" name="return" value="/danke.htm">
<input type="hidden" name="subject" value="Feedback-Formular">
<fieldset><legend>[%legend%]:</legend>
<div class="explain">Felder mit Stern * müssen ausgefüllt
werden.</div>
</fieldset>
</td>
</tr><tr>
<td><fieldset><legend>Daten:</legend>
<div>Vorname:<br>
<input type="text" name="Vorname" class="text"
       style="width:400px">
</div>
<div>Zuname:<br>
<input type="text" name="Zuname" class="text"
       style="width:400px">
</div>
<div>E-Mail: <b>*</b><br>
<input type="text" name="Mail" class="text"
       style="width:400px">
</div>
<div>Feedback: <b>*</b><br>
<textarea name="Text" class="text"
       style="width:400px; height:300px"></textarea>
```

```
</div>
</fieldset>
<fieldset><legend>Daten:</legend>
<p>
<input type="submit" name="feedback" class="button"
       value="Absenden">
<input type="reset" class="button" value="Löschen">
</p>
</fieldset>
</form>
</body>
</html>
```

Listing 12.9: HTML-Datei danke.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Danke für Ihr Feedback</title>
<link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<h1>Danke für Ihr Feedback</h1>
<p>Ihre eingegebenen Daten wurden uns zugesendet.
Wir werden uns mit Ihnen in Verbindung setzen.</p>
</body>
</html>
```

In der Datei *feedback.htm* sind zwei Platzhalter notiert: `[%self%]` steht bei der Wertzuweisung an das action-Attribut im einleitenden `<form>`-Tag. Hier wird das PHP-Script seine eigene Adresse einsetzen, damit das Dateiensemble ohne Änderungen von einer Umgebung zu einer anderen portiert werden kann. Der zweite Platzhalter `[%legend%]`, Inhalt der Feldgruppenüberschrift, wird vom Script unterschiedlich versorgt, je nachdem, ob das Formular erstmalig angezeigt wird oder ob es wiederholt angezeigt wird, weil der Anwender zuvor erforderliche Daten nicht eingegeben hat.

Das PHP-Script: Verzweigungen, Funktionen und Selbstaufruf

Der Quelltext des PHP-Scripts sieht wie folgt aus:

Listing 12.10: PHP-Script für Form-Mailer

```
<?php
#-----------------------------------------
# Globale Daten:
$mailto = "jemand@irgend.wo";
$form_file = "feedback.htm";
$thanks_file = "danke.htm";
$standard_legend = "Hinweis";
```

```
$error_legend = "Fehler";

#-----------------------------------------
# Entscheiden was zu tun ist:
if(isset($_POST['feedback']))
   check_form_data();
else
   show_form_data(false);

#-----------------------------------------
# Funktion show_form_data():
# Formular anzeigen
function show_form_data($with_error) {
   global $form_file, $standard_legend, $error_legend;
   $page = file_get_contents($form_file);
   $page = preg_replace("/\[\%self\%\]/",
       $_SERVER['SCRIPT_NAME'], $page);
   if($with_error == true)
      $page = preg_replace("/\[\%legend\%\]/",
          $error_legend, $page);
   else
      $page = preg_replace("/\[\%legend\%\]/",
          $standard_legend, $page);
   echo $page;
   exit();
}

#-----------------------------------------
# Funktion check_form_data():
# Formulareingaben prüfen
function check_form_data() {
   //echo "<pre>", var_dump($_POST), "</pre>";
   if(empty($_POST['Mail']) or empty($_POST['Text']))
  show_form_data(true);
   else
  mail_and_thanks();
}

#-----------------------------------------
# Funktion mail_and_thanks():
# Formulareingaben mailen und Dankeseite ausgeben
function mail_and_thanks() {
   global $mailto, $thanks_file;
   $headers = "From: {$_POST['Vorname']} {$_POST['Zuname']} ";
   $headers .= "<{$_POST['Mail']}>\r\n";
   $headers .= "Content-Type: text/plain; charset=ISO-8859-1\r\n";
   $headers .= "Content-Transfer-Encoding: 8bit";
   mail($mailto, $_POST['subject'], $_POST['Text'], $headers);
   $page = file_get_contents($thanks_file);
   echo $page;
   exit();
}
?>
```

In diesem Script führen wir eine Technik ein, mit der wir in den vorangegangenen Beispielen noch nicht gearbeitet haben: eigene Funktionen. Insgesamt sind im Script drei Funktionen notiert:

- `show_form_data()` zeigt das Formular an. Dazu liest die Funktion die Datei *feedback.htm* ein, ersetzt darin die beiden Platzhalter und gibt den HTML-Code am Ende aus. Die Funktion erwartet einen Parameter namens `$with_error`, der den Boolean-Wert `false` oder `true` haben sollte. Abhängig von diesem übergebenen Wert ersetzt die Funktion den Platzhalter im Legendentext der Feldgruppenüberschrift entweder mit »Hinweis« oder mit »Fehler«.
- `check_form_data()` überprüft eingegebene Formulardaten daraufhin, ob die beiden Formular-Pflichtfelder *Mail* und *Text* einen Inhalt haben. Wenn ja, wird die Funktion `mail_and_thanks()` aufgerufen. Sind nicht beide Felder gefüllt, dann wird die Funktion `show_form_data()` aufgerufen, und zwar so, dass sie »Fehler« mit ausgibt.
- `mail_and_thanks()` erzeugt aus den übergebenen Formulardaten eine E-Mail, sendet sie an den Seitenbetreiber und gibt abschließend die Danke-Seite aus, indem sie die *danke.htm* einliest und ihren Inhalt ausgibt.

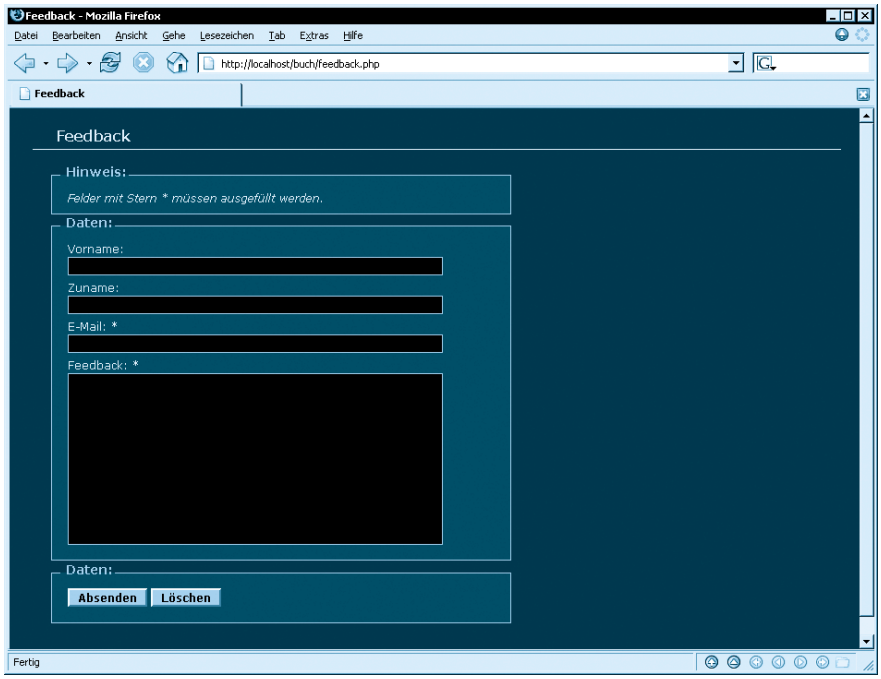


Abbildung 12.7: PHP-Script, hier mit Ausgabe des Formulars

Die gesamte Logik des Scripts erschließt sich allerdings nur beim Betrachten des Abschnitts »Entscheiden, was zu tun ist« im oberen Teil.

Dort wird abgefragt, ob eine Variable namens `$_POST['feedback']` existiert. Da im HTML-Formular beim `<form>`-Tag `method="post"` angegeben ist, wird das PHP-Script auf dem Webserver mit der HTTP-Request-Methode POST aufgerufen. Der Webserver stellt dem Script die Formulardaten in Form einer entsprechenden Umgebungsvariablen zur Verfügung. In PHP können Sie über die superglobale Variable `$_POST` darauf zugreifen. Die Parallelvariable `$_GET` haben wir ja bereits kennen gelernt. Bei `$_POST` handelt es sich wie bei `$_GET` um einen assoziativen Array. Feldbezeichner erscheinen darin als Schlüssel und Feldwerte als den Schlüsseln zugewiesene Werte.

Eine Variable `$_POST['feedback']` gibt es dann, wenn unser Formular ausgefüllt und versendet wurde, und zwar deshalb, weil es innerhalb des Formulars ein Feld mit dem Namen `feedback` gibt. Wir haben den Submit-Button so benannt:

```
<input type="submit" name="feedback" class="button"
       value="Absenden">
```

Beim Erstaufruf des Scripts liegen natürlich noch keine Formulardaten vor. In diesem Fall gelangt das Script in den else-Zweig der if-Abfrage nach `$_POST['feedback']`. Dort wird die Funktion `show_form_data()` aufgerufen. Als Parameter wird ihr der Boolean-Wert `false` übergeben. Die Funktion verrichtet ihren Dienst, d.h., sie liest die Datei ***feedback.htm*** ein, ersetzt darin den Platzhalter `[%self%]` durch den Inhalt der superglobalen Variablen `$_SERVER['SCRIPT_NAME']` (absoluter Webpfadname des Scripts auf dem Server), den Platzhalter `[%legend%]` wegen des übergebenen Parameters `false` auf »Hinweis«, gibt die Seite mit `echo` aus und beendet dann mit `exit()` das Script.

Die Seite mit dem Formular wird nun beim Anwender angezeigt und er hat Zeit, das Formular auszufüllen. Im einleitenden `<form>`-Tag steht nun wegen der Platzhalterersetzung: `action="/buch/feedback.php"` (vorausgesetzt, das PHP-Script wurde so benannt). Das bedeutet, nach Absenden des Formulars wird das Script erneut aufgerufen.

Bei diesem erneuten Aufruf liegen die eingegebenen Formulardaten im superglobalen Array `$_POST` vor. Das Script gelangt nun in den Ausführungszweig unterhalb der if-Abfrage `if(isset($_POST['feedback']))`. Dort wird `check_form_data()` aufgerufen. In dieser Funktion wird lediglich geprüft, ob die beiden Pflichtfelder mit den Namen `Mail` und `Text` leer (***empty***) sind:

```
if(empty($_POST['Mail']) or empty($_POST['Text']))
```

Ist das der Fall, wurde das Formular nicht vollständig ausgefüllt und es soll noch mal angezeigt werden. Deshalb wird in diesem Fall `show_form_data()` aufgerufen – diesmal jedoch mit dem Parameter `true`, was `show_form_data()` veranlasst, das Formular mit der Legendenüberschrift »Fehler« anzuzeigen. Der Anwender kann das Formular dann erneut ausfüllen.

Wurden dagegen beide Pflichtfelder ausgefüllt, gelangt das Script in den else-Zweig. Dort steht ein Aufruf der Funktion `mail_and_thanks()`.

Die Funktion `mail_and_thanks()` stellt zunächst in der Variablen `$header` ein paar wichtige Zeilen zusammen, die in den Kopfdaten der zu versendenden Mail stehen sollen – unter anderem das wichtige Feld `From:`, das den Absender der Mail enthält. Hier trägt die Funktion eine Zeichenkette ein, die sich aus Werten des Formulars ergibt: `{$_POST['Vorname']} {$_POST['Zuname']} <{$_POST['Mail']}>`. Das erzeugt eine Angabe wie etwa *Rainer Zufall <merk@wuerd.ig>*. Der Seitenanbieter erhält dann also die Mail gleich so, als wäre sie vom E-Mail-Programm des Anwenders abgesendet worden. Er kann selber ganz einfach auf die Mail antworten, indem er in seinem Mailprogramm die Funktion »Antworten« wählt.

Mit der PHP-Funktion `mail()` wird die E-Mail versendet. Die Funktion kann allerdings nur dann Mails versenden, wenn die *php.ini* entsprechend konfiguriert wurde. Dort finden Sie – per Voreinstellung auskommentiert – die Einträge *SMTP* und *sendmail_from* für PHP unter MS Windows und den Eintrag *sendmail_path* für Linux/Unix-Systeme. Auf Shell-Ebene eines Linux-Rechners können Sie den Pfad mit dem Kommando `which sendmail` ermitteln.

An `mail()` müssen mindestens drei Parameter übergeben werden: 1. die Mailadresse des Empfängers, 2. das Mail-Subject und 3. der Inhalt der Mail. Als vierten Parameter können zusätzliche Kopfdaten für die Mail übergeben werden und als fünften Parameter Aufrufparameter für das sendmail-Programm. In unserem Beispiel werden vier Parameter übergeben. Die Mailadresse des Seitenanbieters ist in der Variablen `$mailto` gespeichert, die zu Beginn des Scripts definiert wird. Als Mail-Subject wird `$_POST['subject']` übergeben. Das Feld mit dem Namen `subject` ist im HTML-Formular ein hidden-Feld. Als Inhalt der Mail wird `$_POST['Text']` übergeben, also der Inhalt aus dem textarea-Feld in HTML.

Abschließend liest die Funktion `mail_and_thanks()` den Inhalt der Danke-Seite mit `file_get_contents()` ein und gibt ihn mit `echo` aus.

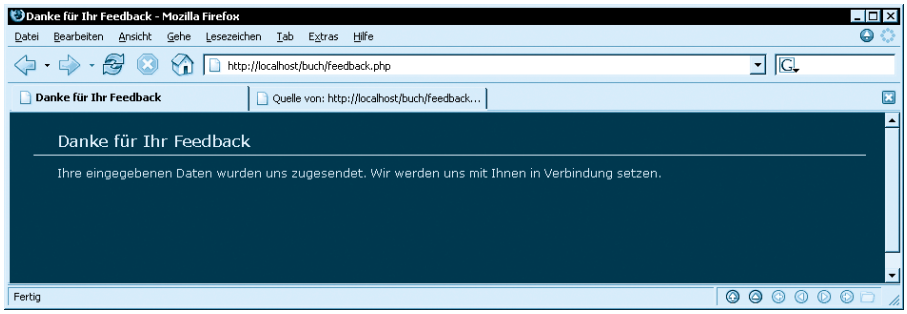


Abbildung 12.8: Immer noch das gleiche PHP-Script, hier mit der Ausgabe der Danke-Seite

Fazit: PHP vereinfacht die Formularauswertung

Wer schon einmal versucht hat, mit POST übergebene Formulardaten in einem Script direkt von der Standardeingabe zu lesen, wird es zu schätzen wissen, dass es in PHP genügt, die gewünschten Felder einfach aus dem superglobalen Array `$_POST` zu holen. Das Dekodieren der Daten, welche bei der Übertragung nach dem Schema www-form-url-encoded

kodiert werden, das Auseinanderdividieren von Feldnamen und Werten – all das hat PHP schon erledigt, bevor überhaupt ein Script gestartet ist, das Formulardaten auslesen möchte.

Auch das Versenden von E-Mails ist dank der praktischen Funktion `mail()` sehr bequem. Allerdings setzt die Funktion voraus, dass ein sendmail-Programm konfiguriert ist, das in der Lage ist, ausgehende Mails zu verarbeiten und via SMTP-Protokoll zu versenden. Auf öffentlichen Server-Rechnern mit Linux-System ist das kein großes Problem. Dort ist eigentlich immer eines der Programme ***sendmail*** oder ***qmail*** installiert und aktiv. Bei anderen Programmen als ***sendmail*** muss allerdings ein sendmail-Wrapper installiert werden, der PHP vortäuscht, es habe es mit ***sendmail*** selber zu tun. Unter MS Windows ist alles problematischer, weil Mails dort nicht direkt durch Aufruf eines Programms wie ***sendmail*** versendet werden können, sondern einen so genannten ***Mail Transfer Agent*** erfordern, der über eine Socket-Verbindung Versandaufforderungen für Mails entgegennehmen kann. Wer unbedingt die sendmail-Funktionalität auf seinem lokalen Windows-PC zum Austesten von Scripts haben möchte, kann unter *http://www.indigostar.com/sendmail.htm* ein Shareware-Programm herunterladen, das ***sendmail*** für Windows nachahmt.

12.2.5 Code-Erstellung und Code-Verteilung

Bislang haben wir nur einzelne PHP-Scripts kennen gelernt, die auch zugleich via URI im Browser aufrufbar waren. Bei größeren Projekten wird der Code jedoch in aller Regel auf mehrere, manchmal sogar sehr viele Scripts verteilt. PHP-Scripts können andere PHP-Scripts einbinden. Dabei müssen nicht alle Scripts innerhalb der Document Root liegen. Nur diejenigen, die im Browser direkt aufrubar sein sollen, müssen sich in einem Webverzeichnis befinden.

Die einzelnen Scripts übernehmen dabei typischerweise abgegrenzte Aufgaben. Ein Script etwa übernimmt die Kommunikation mit der Datenbank, ein zweites erledigt das Verarbeiten von Formulardaten, ein drittes das Behandeln aufgetretener Fehler und ein viertes das Zusammenfügen von HTML-Templates. Ein Script fungiert meistens als »Ober-Script«, das alles koordiniert.

Ein kleines Beispiel soll demonstrieren, wie sich Code aufgabenspezifisch auf mehrere Scripts verteilen lässt und welche Vorteile das bringt. Wir setzen dabei auf dem Script aus ***Abschnitt 12.2.3*** auf und verfolgen das Template-Konzept in etwas veränderter Form weiter. Insgesamt wird unser Projekt jetzt aus folgenden PHP-Scripts bestehen:

- ***linkweb_vars.php*** definiert globale Variablen für das gesamte Script-Ensemble.
- ***linkweb_templates.php*** ist für das Einbinden von Templates zuständig.
- ***linkweb_errors.php*** übernimmt die Ausgabe von Fehlermeldungen.
- ***linkweb.php*** ist das koordinierende Script, das auch im Browser aufgerufen wird.

Dazu kommen noch zwei Template-Dateien:

- *linkweb.tpl* ist das Template für normale Webseiten des Projekts.
- *linkweb_errors.tpl* ist das Template für Fehlermeldungsseiten.

Außerdem benötigen wir natürlich Datendateien. Dazu nehmen wir wieder die Dateien *0001.txt* bis *0015.txt* wie im Beispiel in ***Abschnitt 12.2.3***.

PHP-Script linkweb_vars.php: die Variablenzentrale

Listing 12.11: Script linkweb_vars.php

```
<?php
#-----------------------------------------
# Templates:
$templates = array();
$templates['standard'] = "linkweb.tpl";
$templates['error'] = "linkweb_errors.tpl";
#-----------------------------------------
# Inhaltsdateien:
$content_files = array();
$content_files['home'] = "0001.txt";
$content_files['impressum'] = "0007.txt";
$content_files['themen'] = "0003.txt";
$content_files['branchen'] = "0009.txt";
$content_files['auskunft'] = "0005.txt";
$content_files['geo'] = "0002.txt";
$content_files['wissen'] = "0004.txt";
$content_files['literatur'] = "0008.txt";
$content_files['lexika'] = "0006.txt";
$content_files['glossare'] = "0010.txt";
$content_files['wikis'] = "0014.txt";
$content_files['ftp'] = "0011.txt";
$content_files['wais'] = "0015.txt";
$content_files['newsgroups'] = "0012.txt";
$content_files['foren'] = "0013.txt";
#-----------------------------------------
# Fehler:
$errors = array();
$errors['no_page'] = "Fehler: Seitenname existiert nicht";
$errors['no_template_key'] = "Interner Fehler: Template-Schlüssel existiert
nicht";
$errors['no_error_key'] = "Interner Fehler: Fehler-Schlüssel existiert nicht";
#-----------------------------------------
# Diverses:
$standard_title = "untitled";
?>
```

Wie ersichtlich, werden in diesem Script ausschließlich Variablen definiert. Der Vorteil dieser Lösung ist, dass man bei mehreren Scripts nicht mehr überlegen muss, welche Variable in welchem Script eingeführt wird. Wenn beispielsweise neue Seiten hinzukommen, neue Templates, neue Fehlermeldungen usw., dann wird alles in dieser Script-Datei notiert.

PHP-Script linkweb_templates.php: Zusammenkleben von Templates und Daten

Listing 12.12: Script linkweb_templates.php

```
<?php
#-----------------------------------------
# Includes:
include_once("linkweb_vars.php");
include_once("linkweb_errors.php");

#-----------------------------------------
# Funktionen:

function template_use($template_key) {
   global $templates;
   if(! array_key_exists($template_key, $templates))
      error_show("no_template_key");
   else
      return(file_get_contents($templates[$template_key]));
}

function template_set_var($template_content, $name, $value) {
    $pattern = "/\[\%".$name."\%\]/";
    return(preg_replace($pattern, $value, $template_content));
}

function template_get_title($template_content) {
    global $standard_title;
    $pattern = "/<h1>(.*)<\/h1>/";
    preg_match($pattern, $template_content, $matches);
    if(isset($matches[1]))
       return($matches[1]);
    else
       return($standard_title);
}

function template_show($template_content) {
    echo $template_content;
    exit();
}
?>
```

In diesem Modul-Script begegnet uns erstmals die PHP-Funktion `include_once()`. Sie erlaubt das Einbinden anderer Dateien an der aktuellen Stelle im Code. Eingebunden werden die anderen Scripts *linkweb_vars.php* und *linkweb_errors.php*. Der Grund ist, dass das Script zum Template-Management auf Variablen zugreifen muss, die in *linkweb_vars.php* definiert wurden, und dass bei auftretenden Fehlern eine Funktion aus *linkweb_errrors.php* aufgerufen werden soll.

Neben `include_once()` gibt es auch die ältere Funktion `include()`. Wir ziehen `include_once()` jedoch vor, um Probleme mit Mehrfacheinbindungen zu vermeiden. Darauf kommen wir später noch zurück.

Es ist nicht zwingend nötig, die anderen Dateien wie in unserem Beispiel am Beginn eines Scripts einzubinden. Wichtig ist nur, dass ein anderes PHP-Script eingebunden wird, bevor auf Variablen oder Funktionen dieses Scripts zugegriffen wird. Im Sinne eines übersichtlichen Codes ist es jedoch zweckmäßig, die Inkludierungen am Beginn zu notieren, damit die Abhängigkeiten des Scripts transparent werden.

Das Script ***linkweb_templates.php*** besteht außer den include_once-Anweisungen am Beginn nur aus einer Reihe von Funktionen. Auch das ist typisch für modular aufgebaute Scripts: Der Code in den Modulen wird nicht direkt beim Einbinden ausgeführt, sondern steht in Routinen, die bei Bedarf aufgerufen werden. Folgende Funktionen sind in ***linkweb_templates.php*** vorhanden:

- `template_use()` lädt die Template-Datei eines gewünschten Templates und gibt deren kompletten Inhalt zurück. Ein Script, das ein Template laden möchte, muss sich darum also nicht selber kümmern, sondern ruft einfach diese Funktion mit dem Namen des gewünschten Templates als Parameter auf. Die Funktion ruft, falls kein Template mit dem übergebenen Namen existiert, die Funktion `error_show()` auf, die für eine ordentliche Fehlerausgabe sorgt. Diese Funktion befindet sich in dem eingebundenen Script ***linkweb_errors.php***.
- `template_set_var()` ersetzt in HTML-Code selbst definierte Variablen der Notationsform `[%Variablenname%]` durch einen gewünschten Wert. Auch das ist bereits aus ***Abschnitt 12.2.3*** bekannt. Hier wurde diese »Leistung« jedoch in eine dafür zuständige Funktion verbannt. Ein Script, das diese Leistung wünscht, ruft einfach die Funktion auf.
- `template_get_title()` sucht in HTML-Code nach der ersten h1-Überschrift und gibt deren Elementinhalt zurück. Der Rückgabewert lässt sich als Seitentitel interpretieren.
- `template_show()` gibt Inhalte aus. Gedacht ist die Funktion zur Ausgabe fertig zusammengeklebter Seiteninhalte. Deshalb enthält sie am Ende auch einen Aufruf von `exit()`, denn nach der Ausgabe an den Browser hat das Script seine Arbeit beendet.

Es fällt auf, dass alle Funktionsnamen mit `template_` beginnen. Konventionen dieser Art sind nirgendwo vorgeschrieben. Sie sind einfach Teil einer ordentlichen Programmierung.

PHP-Script linkweb_errors.php: Fehlerausgaben

Listing 12.13: Script linkweb_errors.php

```
<?php
#-----------------------------------------
# Includes:
include_once("linkweb_vars.php");
include_once("linkweb_templates.php");
```

```
#-----------------------------------------
# Funktionen:

function error_show($error_key) {
    global $templates, $errors;
    $page = template_use("error");
    if(array_key_exists($error_key, $errors))
       $page = template_set_var($page, "message",
               $errors[$error_key]);
    else
       $page = template_set_var($page, "message",
               $errors['no_error_key']);
    template_show($page);
}
?>
```

In diesem Script werden die zentralen Variablen und die Template-Verwaltung eingebunden. Auch dieses Script enthält ansonsten nur funktionsgebundenen Code, und zwar in einer einzigen Funktion namens `error_show()`. Diese Funktion ruft die in ***linkweb_templates.php*** stehende Funktion `template_use()` mit dem Template-Namen `error` auf. Dadurch wird `template_use()` veranlasst, die Template-Datei zu laden, deren Pfadname sie über den in ***linkweb_vars-php*** notierten Array `$templates` mit dem Schlüsselnamen `error` ermittelt. Das alles mag etwas verwirrend sein am Anfang, doch es ist wichtig, diese Zusammenhänge zu verstehen und zurückzuverfolgen. Denn das Verständnis dieser Querbezüge zwischen den Script-Modulen ist der Schlüssel für eine eigene erfolgreiche Programmierung im Stil der Profis.

Der Rückgabewert von `template_use()` ist der komplette Inhalt einer Template-Datei. Diese wird innerhalb von `error_show()` in der funktionslokalen Variablen `$page` gespeichert. Als nächstes muss die Variable `[%message%]` in der Template-Datei für Fehlermeldungen durch den gewünschten Fehlermeldungstext ersetzt werden. Dazu wird wieder eine Funktion aus ***linkweb_templates.php*** aufgerufen, nämlich `template_set_var()`. Die Funktion wird mit geeigneten Parametern versorgt. Da `show_error()` nicht sicher sein kann, ob der ihr übergebene Parameter, den sie als Schlüsselname einer Fehlermeldung im Array `$errors` interpretiert, dort tatsächlich vorkommt, wird eine if-Abfrage eingebaut, die das überprüft. Sollte keine Fehlermeldung zum übergebenen Wert existieren, wird eine Standardfehlermeldung ausgegeben.

Am Ende wird `template_show()` aufgerufen, wodurch die Fehlerseite an den Browser ausgegeben wird.

PHP-Script linkweb.php: das Hauptprogramm

Listing 12.14: Script linkweb.php

```
<?php
#-----------------------------------------
# Includes:
include_once("linkweb_templates.php");
```

```
include_once("linkweb_errors.php");

#-----------------------------------------
# Inhalt seitenabhängig einlesen:
if(isset($_GET['page']))
   $get_page = $_GET['page'];
else
   error_show('no_page');
$page = template_use("standard");
$page = template_set_var($page, "content",
        file_get_contents($content_files[$get_page]));
$title = template_get_title($page);
$page = template_set_var($page, "title", $title);
template_show($page);
?>
```

Dies ist das einzige Script, das im Browser aufgerufen wird. Auch hier werden zu Beginn die im weiteren Code benötigten Modul-Scripts eingebunden. Der Inhalt des »Hauptprogramms« steht im Gegensatz zu denen der Modul-Scripts nicht in Funktionen, sondern wird direkt ausgeführt. Die Aufgabe des Hauptprogramms besteht darin, eine normale Seite aus Templates, Datendateien usw. zusammenzukleben und auszugeben. Dazu muss der übergebene GET-Parameter `page` ausgewertet werden. Wurde kein solcher GET-Parameter beim URI-Aufruf des Scripts übergeben, wird `error_show()` aufgerufen, um einen entsprechenden Fehler auszugeben. Ansonsten wird ähnlich wie innerhalb von `error_show()` ein Template-Inhalt in einer Variablen `$page` gespeichert. Der Seiteninhalt aus der passenden Datendatei wird durch Aufruf von `template_set_var()` gesetzt. Ersetzt wird die Zeichenfolge `[%content%]` im Template-Code. Der Wert, durch den sie ersetzt wird, wird durch Aufruf der PHP-Funktion `file_get_contents()` ermittelt. Der Rückgabewert dieser Funktion ist nämlich der komplette Inhalt der eingelesenen Datei. Welche Datei eingelesen werden soll, entscheidet das Script, indem es in dem in ***linkweb_vars.php*** gespeicherten Array `$content_files` den Schlüssel mit dem im GET-String übergebenen Namen verwendet.

Der Titel wird noch ermittelt und in das title-Element als Elementinhalt eingesetzt. Am Ende wird die fertige Seite durch Aufruf von `template_show()` ausgegeben.

Einbindungen kreuz und quer

In unserem Beispiel ergeben sich folgende Abhängigkeiten:

- ***linkweb.php*** bindet ***linkweb_template.php*** und ***linkweb_errors.php*** ein.
- ***linkweb_template.php*** bindet ***linkweb_vars.php*** und ***linkweb_errors.php*** ein.
- ***linkweb_errors.php*** bindet ***linkweb_vars.php*** und ***linkweb_templates.php*** ein.

Durch das muntere Einbinden entsteht die Situation, dass manche Dateien gleich mehrfach eingebunden werden, und sogar gegenseitige Einbindungen kommen vor. In vielen Programmiersprachen führt dies zu Problemen, da deren Compiler das Einbinden gnadenlos so durchziehen wie angewiesen. Dadurch kommen dann Funktionen oder Variab-

len aus mehrfach eingebundenen Modulen im Gesamtquelltext doppelt oder mehrfach vor, was den Compiler zu Fehlermeldungen veranlasst und dazu, das Kompilieren zu verweigern. Das Ergebnis sind die zahlreichen *ifdefs* etwa in C, um die sich der Programmierer selber kümmern muss.

In PHP sorgt die im Beispiel überall verwendete Funktion `include_once()` dafür, dieses Problem zu vermeiden. Wurde ein Modul bereits eingebunden, bindet es der Compiler nicht noch einmal ein.

Fazit: Modularisierung von Code lohnt sich

Vielleicht ist Ihnen aufgefallen, dass die PHP-Quelltexte in den modulartig verteilten Scripts dieses Abschnitts deutlich abstrakter wirken als diejenigen aus den ersten Beispielen. Das liegt daran, dass fast alle Aufgaben nicht mehr »selbst« erledigt werden, sondern durch Aufruf entsprechender Funktionen. Das einheitliche Namenskonzept bei Variablennamen und Funktionsnamen tut ein übriges. Der Code wird dadurch andererseits eleganter und, was noch viel wichtiger ist, besser wartbar.

Mit diesem Modularisierungsbeispiel haben wir uns dem Programmierstil der professionellen PHP-Entwickler bereits angenähert. Was noch fehlt, ist die Verwendung von Objekten, Klassen und Instanzen. Darauf gehen die ***Abschnitte 12.3.7*** und ***12.3.8*** genauer ein.

12.3 Sprachkonzepte von PHP

Nach den ersten praktischen Eindrücken von PHP wird es nun Zeit, sich mit den Details der Sprache zu beschäftigen. Denn in der Praxis benötigen Sie eine genaue Kenntnis der Sprachsyntax und Sie müssen über den Geltungsbereich von Variablen, über Datentypen und viele andere Sprachbesonderheiten Bescheid wissen.

12.3.1 Anweisungen, Blöcke, Ausdrücke und Kommentare

Ein PHP-Script oder ein PHP-Script-Bereich besteht in einer Folge von Anweisungen. Anweisungen können auch in Blöcken stehen, die nur unter bestimmten Bedingungen ausgeführt werden, z.B. in einem Funktionsblock, der nur ausgeführt wird, wenn die Funktion aufgerufen wird, oder in einem Block, der von einer if-Abfrage abhängig ausgeführt wird.

Jede Anweisung wird mit einem Semikolon (`;`) beendet. Nur bei der letzten Anweisung eines Scripts oder Script-Bereichs darf das Semikolon entfallen.

Anweisungsblöcke werden in geschweifte Klammern `{` und `}` eingeschlossen. Bei Funktionen sind diese Klammern in jedem Fall Pflicht. Bei if-else-Konstrukten oder bei while- oder for-Schleifen dürfen die geschweiften Klammern dann entfallen, wenn nur eine abhängige Anweisung folgt.

Anweisungen können sehr unterschiedlich aussehen, z. B.:

```
$name = "Stefan";
include_once("andere.php");
echo "Server: ", $_SERVER['SERVER_NAME'];
$bytes = filesize("daten.txt");
$print_code = htmlspecialchars(file_get_contents("daten.html"));
```

Im ersten Fall wird einer Variablen ein Wert zugewiesen. Im zweiten Fall wird eine PHP-Funktion aufgerufen und einfach ausgeführt. Im dritten Fall wird ein so genanntes Sprachkonstrukt ausgeführt, nämlich `echo`. Übergeben wird eine kommagetrennte Aufzählung von Werten, die auszugeben sind. Im vierten Fall wird ein Funktionsaufruf einer Variablen zugewiesen. Diese Art von Anweisung ist immer dann sinnvoll, wenn Funktionen Werte ermitteln und zurückgeben. Der zurückgegebene Wert kann dann in der Variablen gespeichert werden. Auch die letzte Beispielanweisung gehört zu diesem Typ. Hier wird die PHP-Funktion `htmlspecialchars()` aufgerufen. Diese erwartet als Parameter eine Zeichenkette. Statt ihr eine feste Zeichenkette zu übergeben, können wir ihr jedoch auch eine Variable übergeben, in der eine Zeichenkette gespeichert ist, oder sogar, wie im Beispiel, den Aufruf einer anderen Funktion, die eine Zeichenkette zurückgibt.

Ausdrücke

In der Programmierpraxis ist vielfach auch von Ausdrücken die Rede. Bereits in Zusammenhang mit JavaScript wurde dies kurz behandelt. Ausdrücke sind Teile einer Anweisung, in denen ein Wert erzeugt wird. Dazu sind in aller Regel Operatoren erforderlich. Beispiele:

```
$name = $vorname." ".$zuname;
$x = 4 * (7 + 14);
```

In beiden Beispielen ist der Ausdruck das, was rechts vom Zuweisungsoperator, also vom Gleichheitszeichen steht. Im ersten Fall wird eine Zeichenkette aus mehreren anderen zusammengesetzt und im zweiten Fall wird eine Berechnung durchgeführt. Viele Anweisungen haben diesen Aufbau: Ein Ausdruck wird einer Variablen zugewiesen. Damit wird der vom Ausdruck erzeugte Wert gespeichert.

Ein weiteres Beispiel:

```
$x = 3 < 4;
echo $x;
```

Ausgegeben wird in diesem Beispiel 1, weil in `$x` die Bewertung des Ausdrucks `3 < 4` gespeichert wird. Da < ein Vergleichsoperator ist, ist das »Ergebnis« des Ausdrucks ein Wahrheitswert. Denn der Vergleich trifft entweder zu oder nicht. In unserem Beispiel trifft er zu, weil 3 tatsächlich kleiner ist als 4. In `$x` wird also »wahr« gespeichert. Die numerische Entsprechung für »wahr« ist in PHP wie in vielen anderen Programmiersprachen 1.

Kommentare

Zu PHP kommen auch viele, die schon in anderen Sprachen programmiert haben. Deshalb ist PHP sehr großzügig, was die Syntax von Kommentaren betrifft: Jeder darf seinen früheren »Stil« beibehalten, ob er nun von Perl herkommt oder von C.

Einzeilige Kommentare werden entweder durch ein Gatterzeichen (#) oder durch zwei Schrägstriche (//) eingeleitet. Steht dieses Kommentarsignal am Zeilenanfang, wird die gesamte Zeile auskommentiert. Steht sie an einer späteren Stelle, wird der Teil davor als ausführbarer Code betrachtet. Beispiel:

```
# ----------------------------------------
// Alles Kommentar
# ----------------------------------------
$x = "foo"; // Programmierer-Slang, Teil 1
$y = "bar"; # Programmierer-Slang, Teil 2
```

Mehrzeilige Kommentare können Sie durch `/*` einleiten und mit `*/` beenden. Beispiel:

```
/* ----------------------------------------
   Alles Kommentar
   ---------------------------------------- */
```

Kommentare können selbstverständlich auch PHP-Code enthalten. Dadurch können Sie beispielsweise momentan nicht benötigten oder noch unfertigen Code ausblenden.

12.3.2 Variablen und Datentypen

In Variablen können Sie Werte speichern, die Sie im weiteren Verlauf des Scripts benötigen. Wichtig zu wissen ist, dass Variablen in PHP – zumindest außerhalb vom Umfeld objektorientierter Programmierung – nicht wie in vielen anderen Programmiersprachen erst deklariert werden müssen, bevor mit ihnen gearbeitet werden kann. Es genügt, einer bislang unerwähnten Variablen einen Wert zuzuweisen. Damit existiert diese Variable.

Allen Variablen gemeinsam ist, dass sie mit einem Dollarzeichen (`$`) beginnen, wie z.B. `$name` oder `$x`.

Variablennamen

Bei der Vergabe von Variablennamen sind Sie weitgehend frei. Wegen des variablenidentifizierenden Dollarzeichens dürfen auch PHP-eigene Wörter als Variablennamen dienen, d.h., es sind auch Variablennamen wie `$if` oder `$return` erlaubt.

Groß- und Kleinschreibung werden unterschieden, d.h. `$Name` und `$name` sind innerhalb eines Geltungsbereichs zwei verschiedene Variablen.

Variablen müssen mit einem Buchstaben oder einem Unterstrich (_) beginnen. In der weiteren Folge dürfen auch Ziffern vorkommen. Auch Zeichen jenseits des ASCII-Zeichensatzes sind erlaubt, wie `$Nützlichkeitswert`. Die Länge von Variablennamen ist nicht begrenzt.

Datentypen

Variablen speichern Werte, also Daten. Nun können diese Daten ganz unterschiedlichen Typs sein. PHP unterscheidet folgende einfachen Datentypen:

- Zeichenketten (*Strings*)
- Ganzzahlen (*Integers*)
- Fließkommazahlen (*Floats*)
- Wahr/Falsch-Werte (*Boolean*)

Dass bei numerischen Werten zwischen Ganzzahlen und Fließkommazahlen unterschieden wird, liegt an der unterschiedlichen Speichertechnik solcher Zahlen im Rechner.

Neben diesen einfachen Datentypen gibt es noch weitere:

- *Arrays* (mehrere Werte in einer Variablen)
- *Objekte* (Datenstrukturen mit eigenen Variablen (Eigenschaften) und Funktionen (Methoden))
- *Ressource* (Zeiger auf eine externe Ressource, z.B. ein Datei-Handle oder eine Datenbankverbindung)
- *NULL* (Spezialwert, der bedeutet, dass eine Variable keinen Wert hat)

Zeichenketten

Zeichenketten werden z.B. bei Zuweisungen an Variablen in doppelte oder einfache hohe Anführungszeichen gesetzt. Beispiele:

```
$name = "Anton";
$name = 'Berta';
```

Der Unterschied besteht darin, dass PHP Inhalte in doppelten Anführungszeichen nochmals parst und darin notierte Variablen erkennt. Beispiel:

```
$part = "lupus est";
$sentence = "homo hominem $part";
echo $sentence;
```

Das Beispiel gibt `homo hominem lupus est` aus, weil die an `$sentence` zugewiesene Zeichenkette die zuvor definierte Variable `$part` enthält. Aber:

```
$man = "Anton";
$woman = "$mania";
echo $woman;
```

Dabei wird gar nichts ausgegeben, da zuvor keine Variable namens `$mania` mit einem Wert belegt wurde. Gemeint ist natürlich die Variable `$man`. Um PHP dies mitzuteilen, muss die Variable in geschweifte Klammern eingeschlossen werden:

```
$man = "Anton";
$woman = "{$man}ia";
echo $woman;
```

Jetzt wird `Antonia` ausgegeben, weil die zuvor definierte Variable `$man` erkannt wird. Alle komplexen Variablen, also Array- und Objektvariablen, müssen Sie bei Verwendung innerhalb einer Zeichenkette auf jeden Fall in geschweiften Klammern notieren. Beispiel:

```
$footer_text = "Server: {$_SERVER['SERVER_NAME']}";
```

Innerhalb einfacher Anführungszeichen werden keine Variablen geparst. Beispiel:

```
$part = "lupus est";
$sentence = 'homo hominem $part';
echo $sentence;
```

Dabei wird `homo hominem $part` ausgegeben, weil die enthaltene Variable nicht als solche erkannt wird.

Ein weiterer Unterschied zwischen einfachen und doppelten Anführungszeichen betrifft die Interpretation von Escape-Sequenzen. Auch diese werden nur bei doppelten interpretiert, nicht bei einfachen. Beispiel:

```
$word = "<pre>umge-\nbrochen</pre>";
echo $word;
```

Dies gibt aus:

```
umge-
brochen
```

Der Grund ist, dass `\n` als Escape-Zeichenfolge für »Zeilenumbruch« erkannt wird.

Dagegen jedoch:

```
$word = '<pre>umge-\nbrochen</pre>';
echo $word;
```

Damit wird ausgegeben:

```
umge-\nbrochen
```

Die Escape-Zeichenfolge `\n` wird also nicht erkannt. Weitere Escape-Zeichenfolgen sind z. B. `\r` (Wagenrücklauf) oder `\t` (Tabulator). Auch Zeichen, die innerhalb von Zeichenketten in doppelten Anführungszeichen Sonderbedeutung haben, müssen mit einem Backslash als

Escape-Sequenz notiert werden. Das betrifft das doppelte hohe Anführungszeichen, welches ansonsten die Zeichenkette beenden würde, das Dollarsymbol, welches ansonsten Variablen signalisieren würde, und den Backslash selbst. Beispiel:

```
$link = "<a href=\"/dollar1000.htm\">1000\$-Seite</a>";
$fancy = "ein \\\ ist eigentlich ein \\";
```

Gerade beim Zusammenbasteln von später auszugebendem HTML-Code kommen häufig Zeichenketten vor wie im Beispiel der Variablen `$link`. Wenn Sie dabei mit doppelten Anführungszeichen arbeiten, müssen Sie stets die Zeichen mit Sonderbedeutung durch Backslash maskieren.

Anders bei einfachen Anführungszeichen:

```
$link = '<a href="/dollar1000.htm">1000$-Seite</a>';
$fancy = 'ein \ ist immer ein \ ';
```

In diesem Fall müssen Sie keines der zuvor maskierten Zeichen maskieren. Aber aufgepasst: Zumindest das Zeichen ' selbst muss innerhalb einer so begrenzten Zeichenkette als \' maskiert werden. Deshalb auch das Leerzeichen am Ende der zweiten Zeichenkettenzuweisung im Beispiel. Die Anweisung `$fancy = 'ein \ ist immer ein \';` sieht zwar korrekt aus, würde aber eine PHP-Fehlermeldung erzeugen, da `\'` als Zeichen ' interpretiert wird, danach aber das Anweisungsabschlusszeichen ; folgt. Die Zeichenkette ist damit für PHP nicht korrekt abgeschlossen.

Zeichenketten können Variablen zugewiesen werden. Auch viele PHP-Funktionen erwarten Zeichenketten als Parameter. Und das Sprachkonstrukt `echo` kann Zeichenketten ausgeben. In allen Fällen gelten die hier behandelten Feinheiten.

Zeichenketten in PHP können übrigens »beliebig« lang sein. Nur der verfügbare Speicher stellt eine Grenze dar, eine andere Grenze gibt es nicht. Von daher ist es also unproblematisch, in einer Zeichenkettenvariablen auch den kompletten Inhalt einer größeren Datei zu speichern.

Jedes Zeichen einer Zeichenkette ist ein Byte, d.h., jedes Zeichen kann maximal 256 verschiedene Zustände haben. Deshalb unterstützt PHP selber keine Mehrbyte-Kodierung, wie es etwa Unicode erfordern würde.

Zahlen

PHP unterscheidet wie erwähnt und wie die meisten anderen Programmiersprachen intern zwischen Ganzzahlen und Fließkommazahlen. An der Oberfläche können Sie die Zahlen ganz normal oder in Form von speziellen Notationsformen notieren. Beispiele:

```
$a = 1234;
$b = +1234;
$c = -1234;
$d = 1.234;
$e = 01234;
```

```
$f = 0x1234;
$g = sprintf("0%d",1234);
$h = 1.234e+2;
$i = 1.234e-2;
```

Wenn einer Variablen numerische Werte zugewiesen werden sollen, müssen die Anführungszeichen weggelassen werden. Natürlich könnten Sie anstelle von `$a = 1234` auch `$a = "1234"` notieren: Dann würde allerdings eine Zeichenkette, bestehend aus vier Zeichen, gespeichert, und nicht die Zahl.

Wie an `$b` und `$c` in den Beispielen erkennbar, kann eine Zahl auch mit Vorzeichen notiert werden. Zahlen ohne Vorzeichen werden wie üblich als positive Zahlen interpretiert.

Bei Fließkommazahlen ist das Dezimalzeichen der Punkt. Wie genau eine Fließkommazahl mit vielen Stellen dargestellt wird, hängt von der Präzisionstiefe ab. Diese ist in der *php.ini* einstellbar.

Die Beispiele zu den Variablen `$e` und `$f` im obigen Beispiel sind keine Dezimalzahlen. Gibt man `$e` mit `echo` aus, wird 668 ausgegeben, weil die Ausgabe per Default Dezimalwerte ausgibt, 01234 aber eine Oktalzahl ist (die beim Umrechnen ins Dezimalsystem 668 ergibt). Oktalzahlen werden durch eine führende 0 indiziert. Notieren Sie also niemals als Dezimalzahl gemeinte Zahlen mit führender 0. Und wenn Sie es unbedingt wollen, dann so wie im Beispiel bei der Variablen `$g`. Dieser wird die Dezimalzahl 1234 zugewiesen, jedoch formatiert mit einer 0 davor. Zuständig für diesen Effekt ist die PHP-Funktion `sprintf()`.

Wenn der führenden 0 noch ein x oder X folgt, so wird die Zahl hexadezimal interpretiert. Würde man obige Variable `$f` mit `echo` ausgeben, so würde 4660 ausgegeben, weil dies der ins Dezimalsystem umgerechnete Wert ist, der mit `0x1234` hexadezimal ausgedrückt ist.

Die aus der Wissenschaft und Mathematik bekannten Exponentialschreibweisen sind wie in den Beispielen zu `$h` und `$i` zu sehen ebenfalls erlaubt. In `$h` ist 123.4 gespeichert, weil `e+2` das Fließkomma um zwei Stellen nach rechts verschiebt, und in `$i` ist 0.01234 gespeichert, weil `e-2` das Fließkomma um zwei Stellen nach links verschiebt.

Boolean-Werte

Solche Werte können `true` bzw. `1` oder `false` bzw. `0` sein. Beispiel:

```
$a = true;
$b = false;
if($a) echo "a ";
if($b) echo "b ";
if(!$a) echo "!a ";
if(!$b) echo "!b ";
```

Um einer Variablen einen booleschen Wert zuzuweisen, weisen Sie ihr am besten eines der dafür vorgesehenen Schlüsselwörter true (für »wahr«) oder false (für »nicht wahr«) zu, und zwar ohne Anführungszeichen.

Boolesche Werte sind hilfreich, um so genannte Flags zu speichern, also ja/nein-Zustände. Besonders gut geeignet sind sie für Bedingungen, wie sie in if-Abfragen oder Schleifenköpfen benötigt werden. Da die Bedingung ebenfalls nur wahr oder falsch sein kann, genügt es, einfach die boolesche Variable als Bedingung zu verwenden.

Das obige Beispiel gibt übrigens aus: `a !b`. Der Grund ist, dass nur die erste und die vierte echo-Anweisung ausgeführt wird, da nur in diesen Fällen die formulierte Bedingung wahr ist.

Arrays

Arrays sind ein spezieller Typ von Variablen, bei denen mehrere Werte unter einem Variablennamen gespeichert werden. Auf die einzelnen Werte kann dann entweder numerisch zugegriffen werden, wie z.B. durch `$namen[0]` für den ersten im Array `$namen` gespeicherten Wert, `$namen[1]` für den zweiten Wert usw. Oder es handelt sich um einen assoziativen Array, bei dem über Schlüsselnamen auf Werte zugegriffen werden kann, z.B. durch eine Syntax wie `$mitarbeiter["schmidt"]`. (Es dürfen auch einfache hohe Anführungszeichen sein, also `$mitarbeiter['schmidt']`).

Es gibt zwar in PHP ein Sprachkonstrukt namens `array()`, das aussieht wie eine Funktion. Doch eigentlich ist es nur eine von PHP erlaubte Schreibweise, um einen Array zu initialisieren. Eigentlich genügt es, einem Array einfach Werte zuzuweisen. Beispiel:

```
$namen[] = "Anton";
$namen[] = "Berta";
$namen[] = "Christa";
echo $namen[1];
```

Dies ist das Gleiche wie:

```
$namen = array("Anton", "Berta", "Christa");
```

Durch die eckigen Klammern am Ende des Namens wird eine Variable als Array-Variable erkannt. Um einem Array neue Elemente hinzuzufügen, genügt es, leere eckige Klammern zu notieren wie im Beispiel. Um ein Array-Element herauszugreifen, z.B. für eine Ausgabe von Daten, muss es adressiert werden. Dies kann durch einen 0-basierten Index geschehen, also durch Zahlen, beginnend bei 0 für das erste Element, 1 für das zweite, 2 für das dritte usw. Im obigen Beispiel wird `Berta` ausgegeben, weil das zweite Element angesprochen wird.

Für viele Daten sind jedoch assoziative Arrays interessanter, da diese den Direktzugriff auf bestimmte Elemente über Schlüsselnamen erlauben. Beispiele für die Definition assoziativer Arrays:

```
$mitarbeiter['name'] = "Anna Lyse";
$mitarbeiter['nummer'] = 23903;
$mitarbeiter['telefon'] = 7231;
$mitarbeiter['geburtstag'] = "1968-04-23";
```

Dies ist das Gleiche wie:

```
$mitarbeiter = array(
   'name' => "Anna Lyse",
   'nummer' => 23903,
   'telefon' => 7231,
   'geburtstag' => "1968-04-23"
);
```

Mit echo `$mitarbeiter['telefon']` kann dann beispielsweise die Telefonnummer ausgegeben werden.

Die Schlüsselnamen werden innerhalb der eckigen Klammern in einfachen oder doppelten hohen Anführungszeichen notiert. Bei Definitionen mit `array()` müssen Sie bei assoziativen Arrays den besonderen Zuweisungsoperator => beachten.

Die Beispiele zeigen auch, dass in einem Array keineswegs nur Daten gleichen Typs gespeichert werden dürfen. Jedes Element innerhalb desselben Arrays kann jedem in PHP verfügbaren Datentyp angehören. Auch verschachtelte Arrays sind auf diese Weise kein Problem. Im obigen Beispiel könnte z. B. der Mitarbeitergeburtstag nochmals ein Unter-Array sein. Direkt notiert:

```
$mitarbeiter['geburtstag']['jahr'] = 1968;
$mitarbeiter['geburtstag']['monat'] = 4;
$mitarbeiter['geburtstag']['tag'] = 23;
```

Oder im `array()`-Konstrukt als Ganzes:

```
$mitarbeiter = array(
   'name' => "Anna Lyse",
   'nummer' => 23903,
   'telefon' => 7231,
   'geburtstag' => array(
      'jahr' => 1968,
  'monat' => 4,
  'tag' => 23
   )
);
```

Durch `echo $mitarbeiter['geburtstag']['jahr']` würde in beiden Fällen der Wert 1968 ausgegeben.

Auf Daten verschachtelter Arrays wird also einfach durch Aneinanderreihung der eckigen Klammern zugegriffen.

Objektvariablen

Auf Objektvariablen gehen wir in den *Abschnitten 12.3.7* und *12.3.8* näher ein. Hier nur das typische Schema:

```
$variable = new Objektfunktion();
$variable->Objekteigenschaft = "Wert";
$variable->Objektmethode();
```

Eine Objektvariable (im Schema: `$variable`) wird definiert, indem ihr der Aufruf einer objekterzeugenden Funktion zugewiesen wird, der so genannten Konstruktorfunktion des Objekts. Vor dem Funktionsaufruf muss außerdem das Schlüsselwort `new` notiert werden. Anschließend kann über die Variable auf Eigenschaften (objekteigene Variablen) und Methoden (objekteigene Funktionen) des Objekts zugegriffen werden. Dabei muss der Pfeiloperator benutzt werden, gebildet aus Minuszeichen und Größer-als-Zeichen.

Automatische und explizite Typumwandlung

PHP gehört nicht zu den Programmiersprachen, bei denen beim Erzeugen einer Variable auch deren Typ klar festgelegt wird. Eine Variable kann ihren Typ durchaus ändern. Beispiel:

```
$a = "Heinrich";
echo "$a<br>";
$a = 12;
echo "$a<br>";
$a = true;
echo "$a<br>";
```

Die Variable `$a` speichert in diesem Beispiel zunächst eine Zeichenkette, dann eine Zahl und schließlich einen booleschen Wert. Im Hintergrund findet dabei eine automatische Typumwandlung statt. PHP versucht anhand des zugewiesenen Werts zu erkennen, zu welchem Datentyp die Variable gehört. Auch im folgenden Beispiel denkt PHP mit:

```
$a = "12";
$b = "13";
$c = $a + $b;
echo $c;
```

Ausgegeben wird 25, obwohl sowohl `$a` als auch `$b` als Zeichenketten definiert wurden, mit denen sich nicht rechnen lässt. Anhand des Ausdrucks, der durch den Additionsoperator entsteht, erkennt PHP jedoch die gewünschte Absicht und wandelt `$a` und `$b` implizit in Zahlen um.

Nicht immer geht die automatische Typenkonvertierung gut. Für diese Fälle können Sie einen gewünschten Datentyp direkt angeben. Beispiel:

```
$age = $_POST['age'];
if((int) $age < 18)
   echo "Kiddie";
```

In diesem Beispiel ist in `$age` ein unbekannter Wert gespeichert, weil das Script natürlich nicht raten kann, was ein Anwender in einem Formularfeld zur Altersangabe eingegeben hat. Um für die if-Abfrage sicherzustellen, dass der gespeicherte Wert numerisch interpretiert wird, wird vor der betroffenen Variable `(int)` notiert. Das erzwingt eine Interpretation des gespeicherten Werts als Integer, also Ganzzahl. Kann der Wert nicht als Zahl interpretiert werden, wird 0 ermittelt. Falls der erste Teil des gespeicherten Werts als Zahl erkannt werden kann, wird zumindest dieser ermittelt. Boolesche Werte werden je nach Wert in 1 oder 0 umgewandelt.

Genauso verhält es sich bei `(float)`, was eine explizite Typumwandlung in eine Fließkommazahl bewirkt.

Ebenfalls möglich ist die explizite Typumwandlung `(string)`. Dabei ist alles denkbar einfach: Was gespeichert ist, wird in eine Zeichenkette umgewandelt.

Geltungsbereich von Variablen

Beim Geltungsbereich von Variablen in PHP müssen vor allem Entwickler aufpassen, die von anderen Programmiersprachen herkommen. Denn eine wichtige Regel ist in PHP anders als in vielen anderen Sprachen: Variablen, die außerhalb einer Funktion definiert werden, also so genannte globale Variablen, sind nicht automatisch innerhalb von Funktionen bekannt. Lokale Variablen innerhalb von Funktionen gelten dagegen nur innerhalb der Funktion.

Falsch ist:

```
$x = 42;
function test() {
   echo $x;
}
test();
```

Richtig ist:

```
$x = 42;
function test() {
   global $x;
   echo $x;
}
test();
```

Im Beispiel wird `$x` außerhalb von Funktionen definiert. Innerhalb einer Funktion wie `test()` kann deshalb aber nicht automatisch darauf zugegriffen werden. Sie muss eigens importiert werden. Dazu dient das Sprachkonstrukt `global`, mit dem innerhalb von Funktionen globale Variablen in die Funktion importiert werden können. Das global-Konstrukt wird wie jede Anweisung mit Semikolon abgeschlossen.

Sie können auch mehrere Variablen auf einmal importieren. Trennen Sie diese einfach durch Kommata:

```
global $x, $y, $z;
```

Variablen, die innerhalb von Funktionen definiert werden, gelten automatisch nur innerhalb der Funktion.

Auch Variablen, die innerhalb anderer Anweisungsblöcke definiert werden, gelten nur dort. Beispiel:

```
if(count($_GET) > 0) {
   $x = count($_GET);
}
echo $x;
```

Die Ausgabe bewirkt nichts, weil es keine Variable `$x` mit definiertem Wert gibt außerhalb des Blocks, den die if-Abfrage bildet.

Für Variablen aus anderen PHP-Scriptdateien, die mittels `include()`, `include_once()`, `require()` oder `require_once()` eingebunden werden, gilt: Im eingebundenen Script global definierte Variablen sind auch im einbindenden Script verfügbar.

Alle aus dem aktuellen Script oder aus eingebundenen Scripts verfügbaren globalen Variablen sind übrigens in einem von PHP bereit gestelltenassoziativen Array verfügbar: dem Array `$GLOBALS`. Beispiel:

```
$x = 12;
echo $GLOBALS['x'];
```

Das ist das Gleiche wie:

```
$x = 12;
echo $x;
```

Das Besondere an dem Array `$GLOBALS` ist, dass auf ihn auch innerhalb von Funktionen und Blöcken zugegriffen werden kann (in diesem Fall entfällt also der Import mittels `global`).

Eine Zwischenform stellen Variablen von Objekten, also Objekteigenschaften, dar. Diese werden zwar innerhalb eines Blocks definiert, doch über eine Objektvariable (siehe weiter oben) kann auf sie zugegriffen werden.

Konstanten

Konstanten sind über Namen ansprechbare Werte, die jedoch nicht wie Variablen änderbar sind, sondern nur verwendbar. Beispiel:

```
define("AUTHOR", "Stefan Münz");
echo AUTHOR;
```

Konstanten werden über die PHP-Funktion `define()` definiert. Als erster Parameter wird der gewünschte Name der Konstanten übergeben und als zweiter Parameter der gewünschte Wert. Der Konstantenname kann dann einfach benutzt werden, um ihn z.B. auszugeben oder als Wert in einer Rechenoperation zu verwenden. Dabei wird anders als bei Variablen kein `$`-Zeichen vorangestellt. Weil Konstanten überhaupt nicht von PHP-Schlüsselwörtern oder Funktionsnamen unterscheidbar sind, gehört es zum guten Programmierstil, Konstantennamen ausschließlich aus Großbuchstaben bestehen zu lassen.

Variablen-Debugging

Während der Entwicklungsphase einer Anwendung geschieht es häufig, dass im Programmablauf nicht das passiert, was passieren soll. Eine häufige Ursache sind Variablen, die ihren Wert ändern, wobei irgendetwas nicht bedacht oder abgefangen wurde. Eine der wichtigsten Aufgaben von Debugging-Software ist es daher, den Weg von Variablen in einem Programmablauf zu verfolgen. Dazu ist es sinnvoll, an »verdächtigen« Stellen im Quelltext einfach mal die PHP-Funktion `var_dump()` aufzurufen. Beispiel:

```
$x = 12;
var_dump($x);
```

Das erzeugt folgende Ausgabe an den Browser:

```
int(12)
```

Es wird also der aktuelle Typ der Variable sowie ihr aktueller Wert ausgegeben. Besonders hilfreich ist `var_dump()`, wenn man ihm den Namen eines Arrays übergibt. Dann wird ein Listing mit allen Array-Elementen, ihrem Datentyp und ihrem aktuellen Wert ausgegeben.

Sollte die Ausgabe einmal ziemlich chaotisch aussehen, dann notieren Sie Folgendes:

```
echo "<pre>",var_dump($variable),"</pre>";
```

Durch die HTML-Auszeichnung als präformatierter Text wird der Variablendump in der Regel ordentlich dargestellt.

12.3.3 Superglobale Variablen

Neben den selbst definierten Variablen stellt auch PHP ein ganzes Arsenal an vordefinierten Variablen zur Verfügung. Diese werden auch als ***Superglobals*** oder superglobale Variablen bezeichnet. Die Bedeutung der Bezeichnung erhellt sich dann, wenn man sich die Regel zum Geltungsbereich von Variablen vergegenwärtigt: Globale Variablen in PHP sind eigentlich nur relativ global, weil sie ja innerhalb von Funktionen nur über Umwege verfügbar sind. Superglobale Variablen dagegen sind solche, die überall ohne Umwege verfügbar sind.

Die superglobalen Variablen werden von PHP in Form von Arrays zur Verfügung gestellt. Nachfolgende Tabelle listet die superglobalen Arrays auf:

Array-Name	Bedeutung
`$_COOKIE`	Cookies, die beim Aufruf der Seite an den Webserver übergeben wurden, können über diesen Array ausgelesen werden.
`$_ENV`	Umgebungsvariablen auf dem Server-Rechner. Diese können von System zu System schwanken.
`$_FILES`	Informationen über Dateien, die über ein HTML-Formular mit `<input type="file">` an den Server übertragen wurden.
`$_GET`	Daten, die über die GET-Methode an den Server beim Aufruf übertragen wurden, also im GET-String standen.
`$_POST`	Daten, die über die POST-Methode an den Server beim Aufruf übertragen wurden, also z.B. wenn Daten eines HTML-Formulars anliegen, das mit `method="post"` definiert wurde.
`$_REQUEST`	Aus den Arrays `$_GET`, `$_POST` und `$_COOKIE` zusammengesetzter Array.
`$_SERVER`	Umgebungsvariablen, die vom Webserver zur Verfügung gestellt werden, z.B. zur Server-Software, oder Informationen über den aufrufenden Client.
`$_SESSION`	Session-Variablen. Diese müssen im Rahmen einer Session-Programmierung selbst definiert werden.

Tabelle 12.6: Superglobale Arrays in PHP

Cookies auslesen

Im nachfolgenden Beispiel wird gezeigt, wie mithilfe von JavaScript, also clientseitig, ein Cookie gesetzt wird. Serverseitig wird das Cookie mit PHP ausgelesen:

Listing 12.15: Cookies setzen mit JavaScript und Auslesen mit PHP

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Cookies</title>
<script type="text/javascript">
var now = new Date();
var start = Math.floor(now.getTime() / 1000);
var duration = 0;

function do_count() {
   now = new Date();
   present = Math.floor(now.getTime() / 1000);
   duration = present - start;
   window.setTimeout("do_count()", 1000);
}
function stop_count() {
```

```
    document.cookie = "duration=" + duration;
}

window.setTimeout('do_count()', 1000);
</script>
</head>
<body onunload="stop_count()">
<h1>Cookie</h1>
<?php
    if(count($_COOKIE) > 0) {
        echo "<p>Letzte Aufenthaltsdauer: {$_COOKIE['duration']} ";
        echo "Sekunden. <a href=\"cookie.php\">Link</a></p>";
    }
    else {
        echo "<p>Warten Sie ein wenig und klicken Sie dann auf ";
        echo "den <a href=\"cookie.php\">Link</a></p>";
    }
?>
</body>
</html>
```

Die Datei muss als PHP-Datei abgespeichert werden, also z. B. als *cookie.php*.

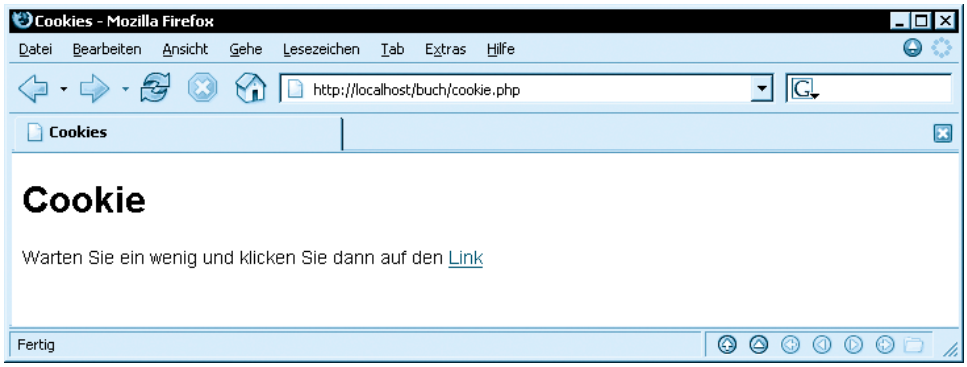


Abbildung 12.9: Seite beim Erstaufruf (noch kein Cookie gesetzt)

Beim Erstaufruf erscheint die Seite wie in Abbildung 12.9 gezeigt. JavaScript zählt während der Anzeigedauer die Sekunden mit. Der angezeigte Link ruft beim Anklicken die Seite selbst erneut auf. Bevor die Seite durch den Link verlassen wird (`onunload`), wird in JavaScript jedoch in der Funktion `stop_count()` ein Cookie gesetzt, das die bis dahin verstrichenen Sekunden der Aufenthaltsdauer speichert.

Cookies haben immer das Schema `Name=Wert`. Im Beispiel lautet der Bezeichnername `duration` und der Wert ist die Anzahl verstrichener Sekunden.

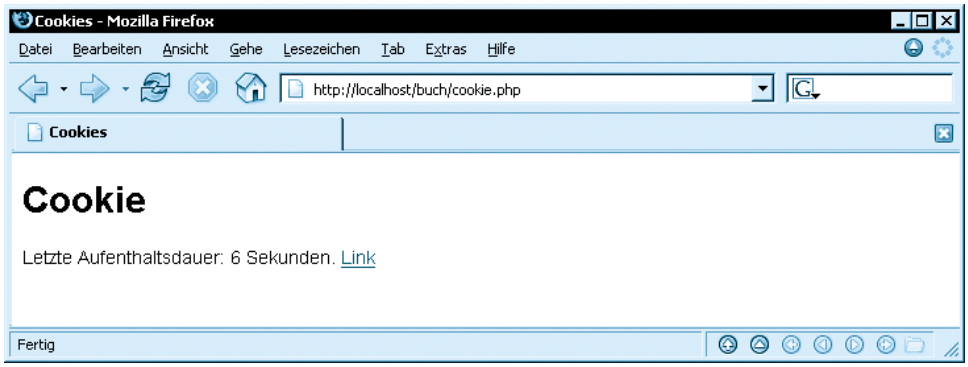


Abbildung 12.10: Seite beim Zweitaufruf (PHP kennt dank JavaScript die Aufenthaltsdauer)

Der angezeigte Text unterhalb der Überschrift wird durch echo-Anweisungen in PHP erzeugt. Ob der Text wie in Abbildung 12.9 oder wie in 12.10 angezeigt wird, entscheidet PHP durch die Abfrage `if(count($_COOKIE) > 0)`. Die Funktion `count()` zählt die Anzahl von Elementen in einem Array. Auf den superglobalen Array `$_COOKIE` angewendet, gibt sie Auskunft darüber, wie viele Cookies übertragen wurden. Eine andere sinnvolle Abfrage wäre `if(array_key_exists("duration", $_COOKIE))` gewesen. Damit wird direkt abgefragt, ob ein Cookie mit dem Namen `duration` übertragen wurde.

Mit `$_COOKIE['duration']` greift das PHP-Script dann auf den Wert des unter `duration` gespeicherten Cookies zu und gibt ihn aus.

Datei-Uploads ermitteln und verarbeiten

Der superglobale Array `$_FILES` ist die Schnittstelle für PHP, um auf Dateien zuzugreifen, die über ein HTML-Formular mittels `<input type="file">` an den Server übertragen wurden. Ein Beispiel:

Listing 12.16: HTML-Datei mit Datei-Upload-Möglichkeit

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Datei-Upload</title>
</head>
<body>
<h1>Datei-Upload</h1>
<form action="file-upload.php"
enctype="multipart/form-data" method="post">
<input type="file" name="uploadfile"><br>
<input type="submit" value="Absenden">
</form>
</body>
</html>
```

Listing 12.17: PHP-Ausgabe der übermittelten Dateiinformationen

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Datei-Upload</title>
</head>
<body>
<h1>Datei-Upload</h1>
<pre>
<?php
var_dump($_FILES);
?>
</pre>
</body>
</html>
```

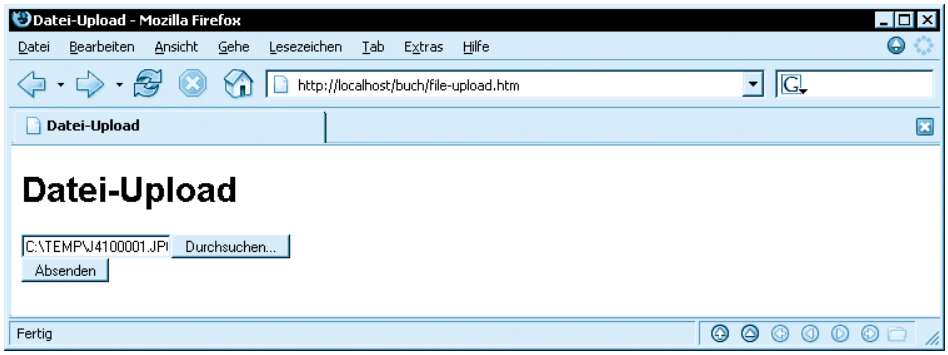


Abbildung 12.11: HTML-Datei mit Datei-Upload-Möglichkeit

Die PHP-Ausgabe mit der PHP-Funktion `var_dump()` zeigt, was dem Script an Daten zur hochgeladenen Datei zur Verfügung steht.

Allgemein zugegriffen werden kann auf die Informationen über `$_FILES['Feldname']`, wobei `Feldname` der Name ist, der im HTML-Formular bei `<input type="file" name=>` vergeben wurde. In unserem Beispiel kann also mit `$_FILES['uploadfile']` auf die Informationen der hochgeladenen Datei zugegriffen werden.

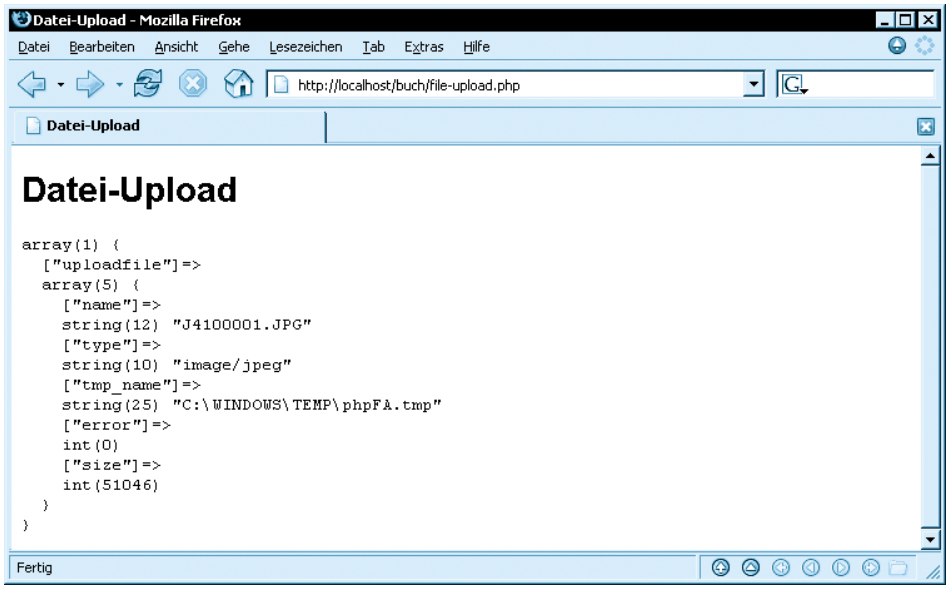


Abbildung 12.12: PHP-Ausgabe der Informationen zur übertragenen Datei

Folgende Informationen werden dem PHP-Script vermittelt:

- `['name']`: Der Originaldateiname, d.h. wie die Datei auf dem Rechner des Anwenders hieß.
- `['type']`: Der Mime-Type, sofern er ermittelt werden konnte oder angegeben wurde.
- `['tmp_name']`: Der absolute Pfadname, unter dem die Datei auf dem Server-Rechner vorübergehend gespeichert wurde.
- `['error']`: Die Anzahl von Fehlern, die bei der Übertragung aufgetreten sind.
- `['size']`: Die Dateigröße in Byte.

Das Script könnte die Datei im weiteren Verlauf also etwa so einlesen:

```
$fh = fopen ($_FILES['uploadfile']['tmp_name'], "r");
$contents = fread ($fh, filesize ($_FILES['uploadfile']['size']));
fclose ($fh);
```

12.3.4 Operatoren

Operatoren dienen dazu, Ausdrücke zu erzeugen, Beziehungen zu schaffen usw. Wie alle modernen Programmiersprachen unterscheidet PHP verschiedene Typen von Operatoren.

Zuweisungsoperator

Das Gleichheitszeichen (=) dient als Zuweisungsoperator. Beispiele:

```
$name = "Buch";
$seite = 23;
$gelesen = false;
```

Bei einer Zuweisung muss links vom Operator eine Variable stehen. Dieser wird der Wert zugewiesen, der rechts notiert ist oder aus dem rechts notierten Ausdruck ermittelt wird.

Vergleichsoperatoren

Vergleichsoperatoren dienen zum Erzeugen von Ausdrücken, deren Wert wahr oder falsch sein kann. Folgende Operatoren dieser Sorte stellt PHP bereit:

Vergleichsbeispiel	Bedeutung
`$a == $b`	Es wird ermittelt, ob `$a` und `$b` den gleichen Wert haben. Dabei wird gegebenenfalls auch eine automatische Typenkonvertierung durchgeführt: 23 und »23« würden als gleich ermittelt.
`$a === $b`	Es wird ermittelt, ob `$a` und `$b` den gleichen Wert haben. Dabei wird auch der Typ berücksichtigt und es wird keine automatische Typenkonvertierung durchgeführt: 23 und »23« würden als nicht gleich ermittelt.
`$a != $b`	Es wird ermittelt, ob `$a` und `$b` einen unterschiedlichen Wert haben. Der Ausdruck ist wahr, wenn die Werte verschieden sind, und falsch, wenn sie gleich sind. Dabei wird wie bei `==` gegebenenfalls eine automatische Typenkonvertierung durchgeführt.
`$a !== $b`	Es wird ermittelt, ob `$a` und `$b` einen unterschiedlichen Wert haben. Der Ausdruck ist wahr, wenn die Werte verschieden sind, und falsch, wenn sie gleich sind. Dabei wird wie bei `===` auch der Variablentyp berücksichtigt und es wird keine automatische Typenkonvertierung durchgeführt.
`$a <> $b`	Andere Schreibweise für `!=`.
`$a < $b`	Es wird ermittelt, ob `$a` kleiner als `$b` ist. Der Ausdruck ist wahr, wenn `$a` kleiner als `$b` ist, und falsch, wenn `$a` größer als oder gleich `$b` ist. Es wird gegebenenfalls eine automatische Typenkonvertierung durchgeführt.
`$a <= $b`	Es wird ermittelt, ob `$a` kleiner als oder gleich `$b` ist. Der Ausdruck ist wahr, wenn `$a` kleiner als oder gleich `$b` ist, und falsch, wenn `$a` größer als `$b` ist. Es wird gegebenenfalls eine automatische Typenkonvertierung durchgeführt.
`$a > $b`	Es wird ermittelt, ob `$a` größer als `$b` ist. Der Ausdruck ist wahr, wenn `$a` größer als `$b` ist, und falsch, wenn `$a` kleiner als oder gleich `$b` ist. Es wird gegebenenfalls eine automatische Typenkonvertierung durchgeführt.
`$a >= $b`	Es wird ermittelt, ob `$a` größer als oder gleich `$b` ist. Der Ausdruck ist wahr, wenn `$a` größer als oder gleich `$b` ist, und falsch, wenn `$a` kleiner als `$b` ist. Es wird gegebenenfalls eine automatische Typenkonvertierung durchgeführt.
`x ? a : b`	Es wird ermittelt, ob Ausdruck `x` wahr ist. Wenn ja, wird Anweisung `a` ausgeführt, wenn nein, wird Anweisung `b` ausgeführt.

Tabelle 12.7: Vergleichsoperatoren in PHP

Vergleichsoperatoren werden vor allem in Abfragebedingungen für if-else-Konstrukte sowie als Durchlaufbedingung in Schleifenköpfen benötigt.

Rechenoperatoren, Inkrementierung und Dekrementierung

Über die üblichen Operatoren +, –, * und / stehen die vier Grundrechenarten zur Verfügung. Daneben gibt es den bei allen auf C basierenden Programmiersprachen typischen Operator % für Modulodivision. Er ermittelt den Restwert einer ganzzahligen Division. Beispiele:

```
$mwst = 56 * 0.16;
$rest = 87 % 17;
```

Die erste Berechnung ist eine gewöhnliche Multiplikation. Bei der zweiten Berechnung steht in `$rest` am Ende der Wert 2, weil 85/17 ganzzahlig teilbar ist und 87 um 2 größer ist als 85.

Bei den Grundrechenarten gilt die übliche Punkt-vor-Strich-Regel. Um die Regel zu umgehen, kann mit Klammern gearbeitet werden. Klammern können beliebig verschachtelt werden:

```
$a = 3 * 7 + 4;
echo $a, " ";
$b = 3 * (7 + 4);
echo $b;
```

In `$a` wird 25 gespeichert, weil zunächst 3 * 7 errechnet wird und anschließend 4 addiert wird, während in `$b` 33 gespeichert wird, weil zuerst 4 + 7 errechnet wird und anschließend 3 mit dem Ergebnis aus der Klammer multipliziert wird.

Eng verwandt mit den arithmetischen Operatoren sind diejenigen zur Inkrementierung und Dekrementierung. Typisches Anwendungsbeispiel sind Schleifenköpfe von for-Schleifen:

```
$namen = array("Ina", "Nina", "Tina", "Lina");
for($i = 0; $i < count($namen); $i++)
   echo "<p>{$namen[$i]}</p>";
```

Der Inkrementierungs-Operator ist ++ und zählt den Wert einer Variablen um 1 hoch. Der Dekrementierungsoperator ist `--` und zählt den Wert einer Variablen um 1 herunter. Je nachdem, ob die Variable wie im Beispiel vor dem Operator steht oder dahinter, spricht man von Pre-Inkrementierung (`++$i`), Post-Inkrementierung (`$i++`), Pre-Dekrementierung (`--$i`) oder Post-Dekrementierung (`$i--`). Bei »Post« wird erst der aktuelle Variablenwert »verwendet« und anschließend wird er um 1 erhöht oder erniedrigt. Bei »Pre« dagegen wird der aktuelle Variablenwert zuerst erhöht bzw. erniedrigt und dann erst »verwendet«.

Logische Operatoren

Logische Operatoren dienen der Verknüpfung von Einzelausdrücken oder booleschen Werten, die wahr oder falsch sein können, zu komplexen Ausdrücken. Der Gesamtausdruck wird durch sie wahr oder falsch. Nachfolgende logische Operatoren werden von PHP zur Verfügung gestellt:

Verknüpfungsbeispiel	Bedeutung
`$a and $b`	Logische Und-Verknüpfung. Der Gesamtausdruck ist wahr, wenn sowohl `$a` als auch `$b` wahr sind. In allen anderen Fällen ist er falsch.
`$a && $b`	Andere Schreibweise mit gleicher Bedeutung wie `and`.
`$a or $b`	Logische Verknüpfung mit inklusivem Oder. Der Gesamtausdruck ist wahr, wenn einer von beiden Ausdrücken, die in `$a` und `$b` gespeichert sind, wahr ist oder wenn beide wahr sind. Nur wenn beide falsch sind, ist der Gesamtausdruck falsch.
`$a || $b`	Andere Schreibweise mit gleicher Bedeutung wie `or`.
`$a xor $b`	Logische Verknüpfung mit exklusivem Oder. Der Gesamtausdruck ist wahr, wenn entweder `$a` wahr ist und `$b` falsch oder wenn `$a` falsch ist und `$b` wahr. Wenn `$a` und `$b` beide wahr oder beide falsch sind, ist der Gesamtausdruck falsch.
`! $a`	Logische Verneinung. Der Gesamtausdruck ist wahr, wenn der in `$a` gespeicherte Ausdruck falsch ist, und er ist falsch, wenn der in `$a` gespeicherte Ausdruck wahr ist.

Tabelle 12.8: Logische Operatoren in PHP

In der Praxis kommen logische Operatoren meist dann zum Einsatz, wenn mehrere Vergleiche miteinander verknüpft werden sollen. Beispiel:

```
while($value < 1 or $value > 49)
   repeat();
```

In der Durchlaufbedingung für eine while-Schleife wird festgelegt, dass der Schleifenkörper dann ausgeführt wird, wenn `$value` kleiner als 1 oder größer als 49 ist. Es genügt, wenn einer der Vergleiche wahr ist.

Operator für Zeichenkettenverknüpfung

In neueren Programmier- und Script-Sprachen wird das Zusammenfügen von zwei oder mehreren Zeichenketten wegen der häufigen Anwendung nicht mehr über Funktionen gelöst, sondern es wird ein Operator bereitgestellt, der es erlaubt, Zeichenketten direkt zu verknüpfen. In PHP ist dies der Punkt-Operator. Beispiel:

```
$given_name = "Stefan ";
$family_name = "Münz";
$name = $given_name.$family_name;
echo $name;
```

In der dritten Zeile werden die in den beiden Zeilen davor definierten Variablen durch den Punkt-Operator verknüpft. Die Verknüpfung wird einer neuen Variablen zugewiesen. Darin ist am Ende `Stefan Münz` gespeichert.

Selbstverständlich dürfen auch mehr als zwei Zeichenketten auf einmal verknüpft werden und es dürfen Variablen mit literal notierten Zeichenketten vermischt werden. Beispiel:

```
$output = "<p><b>Server: </b>".$_SERVER['SERVER_NAME']."</p>";
echo $output;
```

In diesem Fall werden drei Einzelzeichenketten zusammengefügt und nur die mittlere davon ist eine Variable.

Spezielle Notationsformen zum Ändern von Werten

Häufig sind Anweisungen nötig wie »addiere zum bisherigen Wert von $x 10 dazu« oder »erweitere die bisher in $str gespeicherte Zeichenkette um 'abc'«. Das kann so aussehen:

```
$x = $x + 10;
$str = $str."abc"
```

PHP bietet ebenso wie viele andere Programmiersprachen eine verkürzte Darstellungsform dafür an:

```
$x += 10;
$str .= "abc"
```

Anwendbar ist diese Notationsform auf die Operatoren für die vier Grundrechenarten, auf den Operator für Modulodivision, auf den Verknüpfungsoperator für Zeichenketten und auf die nachfolgend kurz behandelten Bit-Operatoren.

Bit-Operatoren

Bit-Operatoren haben zwei Aufgaben: Sie können auf Bit-Ebene Werte durch Verschieben von Bits oder durch logischen Vergleich von Bits pro Stelle verändern. Dazu muss der Programmierer allerdings wissen, wie ein Wert im Arbeitsspeicher genau gespeichert ist – ein Wissen, von dem einen PHP ansonsten gerade abschirmt.

Die Zahl 100 sieht auf Bit-Ebene, also in 0-1-Schweibweise, so aus:

```
1100100
```

Nun speichert PHP Zahlen wie 100, also typische Integer-Werte, in 4 Byte großen Feldern. Da sich jedes Byte aus 8 Bit zusammensetzt, besteht das Feld also aus 32 Bit und würde bei der Zahl 100 wie folgt aussehen:

```
00000000000000000000000001100100
```

Nehmen wir nun eine zweite Zahl, nämlich 1000. Diese sieht im 4-Byte-Feld so aus:

```
00000000000000000000001111101000
```

Nun können wir Bit-Operatoren auf die beiden Variablen anwenden:

```
$a = 100;
$b = 1000;
$c = $a & $b;
```

In `$c` steht am Ende der Wert 96. Das hat folgenden Grund. Der Bit-Operator `&` setzt an allen Bit-Stellen im 32-Bit-Feld, in denen sowohl in $a als auch in $b ein Bit gesetzt ist, ein Bit, also eine 1. In allen anderen Fällen setzt er kein Bit, also 0.

Die nachfolgende »Rechnung« zeigt, was genau geschieht:

```
00000000000000000000000001100100 &
00000000000000000000000001111101000
--------------------------------
00000000000000000000000001100000
```

Der Dezimalwert zu `1100000` ist 96.

Ähnlich funktionieren alle Bit-Operatoren:

Vergleichsbeispiel	Bedeutung
`$a & $b`	Bitweise Und-Verknüpfung. Nur an Stellen, wo in `$a` und `$b` Bits gesetzt sind (1), wird im Ergebnis ein Bit gesetzt (1). An allen übrigen Stellen wird kein Bit gesetzt (0).
`$a | $b`	Bitweise Verknüpfung mit inklusivem Oder. An Stellen, wo in `$a` oder in `$b` oder in `$a` und `$b` Bits gesetzt sind (1), wird im Ergebnis ein Bit gesetzt (1). An den übrigen Stellen wird kein Bit gesetzt (0).
`$a ^ $b`	Bitweise Verknüpfung mit exklusivem Oder. Nur an Stellen, wo in `$a` ein Bit gesetzt ist (1), aber nicht in `$b` (0), oder wo in `$a` kein Bit gesetzt ist (0), aber in `$b` (1), wird im Ergebnis ein Bit gesetzt (1). An allen übrigen Stellen wird kein Bit gesetzt (0).
`~ $a`	Bit-Umkehrung. An Stellen, wo in `$a` ein Bit gesetzt ist (1), wird im Ergebnis kein Bit gesetzt (0). An Stellen, wo in `$a` kein Bit gesetzt ist (0), wird im Ergebnis ein Bit gesetzt (1).
`$a << $b`	Bitweise Linksverschiebung. Alle Bits von `$a` werden um so viele Stellen nach links verschoben wie in `$b` angegeben, z.B. `$a << 3`. Rechts wird mit 0-Bits aufgefüllt. Bei dieser Operation bewirkt jede Stelle eine Multiplikation mit 2.
`$a >> $b`	Bitweise Rechtsverschiebung. Alle Bits von `$a` werden um so viele Stellen nach rechts verschoben wie in `$b` angegeben, z.B. `$a >> 3`. Links wird mit 0-Bits aufgefüllt. Bei dieser Operation bewirkt jede Stelle eine Division durch 2.

Tabelle 12.9: Bit-Operatoren in PHP

12.3.5 Kontrollstrukturen

In PHP stehen folgende Kontrollstrukturen zur Verfügung:

- Bedingte Ausführung von Anweisungen und Verzweigungen mit `if`, `elseif` und `else`.
- Fallunterscheidungen mit `switch/case`.
- Abarbeitungsschleifen mit `for` und `foreach`.
- Kopfschleifen mit `while` und Fußschleifen mit `do ... while`.

Bedingte Ausführung und Verzweigung mit if, elseif und else

Eine oder mehrere Anweisungen können abhängig davon ausgeführt werden, ob eine Bedingung erfüllt ist. Beispiel:

```
if($_GET['page'] == 'home') {
   $page = file_get_contents('home.dat');
   replace_variables('home', $page);
   echo $page;
}
```

Der Block bedingt auszuführender Anweisungen wird durch das Schlüsselwort `if` eingeleitet. In Klammern wird dahinter eine Bedingung formuliert, die wahr oder falsch sein kann. Typisch dafür ist die Verwendung von Vergleichsoperatoren, da diese ein Ergebnis vom Typ »wahr« oder »falsch« liefern. Auch die Verknüpfung von Ausdrücken mit logischen Operatoren kommt häufig vor, da diese ebenfalls Wahr-Falsch-Werte liefern.

Daneben liefern aber auch viele PHP-Funktionen `true` oder `false` als Ergebnis zurück. Ein Beispiel:

```
$name = "Rainer Zufall";
if(isset($name))
   echo $name;
```

Die PHP-Funktion `isset()` ermittelt, ob eine Variable existiert oder nicht. Wenn ja, wird `true` zurückgegeben, wenn nein, dann `false`. Innerhalb der Klammern von `if()` als Bedingung notiert, bewirkt der Funktionsaufruf, dass die bedingt auszuführenden Anweisungen dann ausgeführt werden, wenn `isset()` den Wert `true` zurückliefert.

Wenn mehrere Bedingungen abhängig von `if` ausgeführt werden sollen, müssen diese als Block in geschweifte Klammern eingeschlossen werden, wie im ersten der obigen Beispiele. Wenn nur eine Anweisung bedingt ausgeführt werden soll, können die geschweiften Klammern entfallen.

Mit `else` kann ein Alternativzweig für die Fälle definiert werden, in denen die if-Bedingung nicht erfüllt ist. Beispiel:

```
if(isset($name))
   echo $name;
else
   echo "Hier könnte Ihr Name stehen!";
```

Auch bei else gilt: Sollen mehr als eine Anweisung abhängig davon ausgeführt werden, müssen diese in geschweifte Klammern eingeschlossen werden.

Für einfache Fallunterscheidungen können Sie auch noch ein oder mehrere Zweige des Typs `elseif` zwischen `if` und `else` setzen. Beispiel:

```
if($_GET['page'] == 'home') {
   $page = file_get_contents('home.dat');
```

```
    replace_variables('home', $page);
}
elseif($_GET['page'] == 'after_login') {
    check_login();
    $page = file_get_contents('logged_in.dat');
    replace_variables('logged_in', $page);
}
else
    $page = file_get_contents('login.dat');
echo $page;
```

Bei `elseif` muss ebenso wie bei `if` eine Bedingung formuliert werden, und zwar eine andere als bei `if`. Wenn allerdings wie im Beispiel sowohl bei der if-Bedingung als auch bei einer oder mehreren elseif-Bedingungen immer wieder abgefragt wird, ob eine bestimmte Variable einen bestimmten Wert hat, dann bieten sich auch Fallunterscheidungen an, wie nachfolgend beschrieben.

Fallunterscheidung mit switch/case

Nicht selten besteht das »Herzstück« einer größeren PHP-Anwendung letztlich aus einer mehr oder weniger umfangreichen switch/case-Konstruktion. Denn wenn ein Script beispielsweise als Container dient, um via GET-Parameter bestimmte Seiten auszugeben, aber bei jeder Seite andere Aufgaben erfüllen muss, dann ist das ein typischer Fall für eine Fallunterscheidung. Um also beim Beispiel zu if-elseif-else zu bleiben – dies könnte auch so programmiert werden:

```
switch($_GET['page']) {
    case 'home':
        $page = file_get_contents('home.dat');
        replace_variables('home', $page);
        break;
    case 'after_login':
        check_login();
        $page = file_get_contents('logged_in.dat');
        replace_variables('logged_in', $page);
        break;
    default:
        $page = file_get_contents('login.dat');
        break;
}
echo $page;
```

Das Konstrukt wird eingeleitet mit `switch($Variablenname)`. Für jeden Wert, der unterschieden werden soll, wird ein `case Wert:` notiert. Zeichenkettenwerte werden dabei wie üblich in einfache oder doppelte hohe Anführungszeichen gesetzt, Zahlenwerte nicht. Im Anschluss daran können beliebig viele Anweisungen notiert werden, die ausgeführt werden, wenn die Fallunterscheidungsvariable diesen Wert hat. Dabei sind keine geschweiften Klammern nötig. Jedoch muss der gesamte switch-Block in geschweifte Klammern gesetzt werden.

Am Ende sollte noch an Stelle eines `case ...:` ein `default:` notiert werden. Dies entspricht einem else-Zweig und ermöglicht es, alle Fälle zu behandeln, die durch die explizit unterschiedenen Fälle nicht abgedeckt werden.

Von großer Bedeutung ist bei Fallunterscheidungen mit `switch/case` auch die Anweisung `break;`. Wird sie nicht notiert, dann werden alle nachfolgenden Fälle ebenfalls ausgeführt. In manchen Fällen kann das sogar gewünscht sein, meistens jedoch nicht. Wenn also von dem gesamten switch-Konstrukt maximal ein Fall ausgeführt wird, dann muss am Ende jedes Falls ein `break;` notiert werden.

Abarbeitungsschleifen mit for und foreach

Den typischen Aufbau einer for-Schleife kennen Sie bereits aus JavaSript. Das nachfolgende Beispiel zeigt, wie for-Schleifen zur Erstellung einer HTML-Tabelle genutzt werden können:

```
$html = '<table border="1">';
for($i = 1; $i <= 10; $i++) {
   $html .= '<tr>';
   for($j = 1; $j <= 10; $j++)
      $html .= '<td>'.($i * $j).'</td>';
   $html .= '</tr>';
}
$html .= '</table>';
echo $html;
```

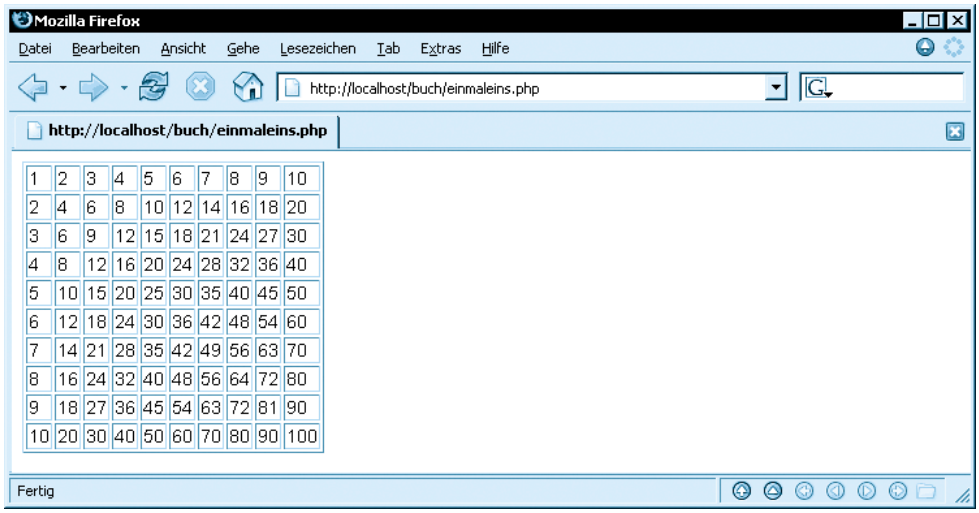


Abbildung 12.13: Anwendungsfall für for-Schleifen: Tabellen mit berechenbaren Inhalten

Das Beispiel erzeugt mittels einer verschachtelten for-Schleife eine HTML-Tabelle mit dem kleinen Einmaleins. Dabei wird die Variable `$html` in einer äußeren und dann noch in einer inneren for-Schleife sukzessive um entsprechende Inhalte erweitert. Am Ende genügt es, die Variable auszugeben.

Eine for-Schleife beginnt mit dem Schlüsselwort `for`. In den Klammern dahinter stehen typischerweise drei Anweisungen. Die erste initialisiert eine Variable mit einem Wert, die dritte verändert den Wert der Variablen und die mittlere formuliert eine Schleifendurchlaufbedingung. Die erste und die dritte Anweisung müssen so geartet sein, dass durch das Verändern der Variablen mit der dritten Anweisung irgendwann ein Wert erreicht ist, der nicht mehr die Durchlaufbedingung erfüllt. Dann ist die for-Schleife beendet.

Es hat sich unter vielen Programmierern eingebürgert, die Zählvariable einer for-Schleife `$i` zu benennen. Sollte wie im Beispiel eine innere for-Schleife benötigt werden, die eine andere Variable benutzen muss, kann der nächste Buchstabe im Alphabet verwendet werden.

Im Beispiel ist `for($i = 1; $i <= 10; $i++)` also so zu lesen:

`$i` wird auf 1 gesetzt. Ist `$i` kleiner oder gleich 10? Ja! Also führe den Schleifenkörper aus. Zähle `$i` außerdem hoch. `$i` wird dadurch 2. Ist `$i` kleiner oder gleich 10? Ja! Also führe den Schleifenkörper aus. Zähle außerdem hoch usw. Irgendwann hat $i den Wert 11 und die Schleifendurchlaufbedingung ist nicht mehr erfüllt.

Den Schleifenkörper bilden diejenigen Anweisungen, die bei jedem Schleifendurchgang ausgeführt werden sollen. Bei mehr als einer Anweisung müssen die Anweisungen in geschweifte Klammern eingeschlossen werden.

Neben der klassischen for-Schleife bietet PHP auch eine foreach-Schleife an. Sie bietet sich speziell an, um alle Elemente eines Arrays zu durchlaufen (***Traversion***) und zwar dann, wenn man innerhalb des Schleifenkörpers keinen Zugriff auf eine Zählervariable wie `$i` benötigt. Nachfolgendes Beispiel zeigt einen Ausschnitt aus einer HTML-Datei:

```
<h1>Städte</h1>
<h2>Aachen</h2>
<p>bla</p>
<p>bla</p>
<p>bla</p>
<p>bla</p>
<h2>Augsburg</h2>
<p>bla</p>
<p>bla</p>
<p>bla</p>
<h2>Bamberg</h2>
<p>bla</p>
<p>bla</p>
<h2>Berlin</h2>
<p>bla</p>
```

Wenn die Aufgabe gestellt ist, aus diesen Zeilen die in h2-Überschriften notierten Städtenamen zu extrahieren und in einer Aufzählungsliste auszugeben, so bietet sich eine foreach-Schleife an:

```
$lines = file("staedte.html");
echo "<ul>\n";
```

```
foreach($lines as $line) {
   if(preg_match("/<h2>(.*)<\/h2>/", $line, $matches))
      echo "<li>{$matches[1]}</li>\n";
}
echo "</ul>";
```

Zunächst wird mit der PHP-Funktion `file()` die HTML-Datei eingelesen – in unserem Beispiel heißt sie ***staedte.html***. Gespeichert wird das Ergebnis in `$lines`. Da `file()` eine Datei zeilenweise einliest, ist `$lines` am Ende ein Array, wobei in jedem Element eine Dateizeile steht.

In der foreach-Schleife werden alle Zeilen der Reihe nach abgearbeitet. Bei jeder Zeile wird gefragt, ob auf sie das Muster »<h2>Irgendwas</h2>« passt. Wenn ja, steht hinterher in der mit übergebenen Variablen `$matches`, aus der die Suchfunktion `preg_match()` einen Array macht, im zweiten Element (also in `$matches[1]`) der im regulären Ausdruck geklammerte Inhalt, also das, was zwischen <h2> und </h2> steht. Passt eine Zeile auf das Suchmuster, dann wird ein Listenpunkt der Aufzählung ausgegeben.

Der foreach-Schleifenkopf hat folgenden typischen Aufbau:

```
foreach($Arrayvariable as $Elementvariable)
```

Damit wird `$Arrayvariable` so oft durchlaufen, wie es Elemente im Array gibt. Innerhalb des Schleifenkörpers kann auf das jeweils aktuelle Element mit `$Elementvariable` zugegriffen werden.

Bei assoziativen Arrays ist auch folgende Variante möglich:

```
foreach($Arrayvariable as $Name => $Wert)
```

Damit kann dann innerhalb des Schleifenkörpers sowohl auf den Schlüsselnamen als auch auf den zugehörigen Wert eines Array-Elements zugegriffen werden. Beispiel:

```
foreach($_POST as $name => $value)
   echo "<b>$name:</b> $value</b><br>";
```

Das Beispiel gibt alle im superglobalen Array `$_POST` gespeicherten Elemente mit Feldnamen und Wert aus.

Kopfschleifen mit while und Fußschleifen mit do ... while

while-Schleifen benötigen im Schleifenkopf einen Ausdruck, der wahr oder falsch sein kann, genauso wie if-Bedingungen. Die Schleife wird dann so oft durchlaufen, wie der Ausdruck im Schleifenkopf wahr ist. Innerhalb des Schleifenkörpers muss dafür gesorgt werden, dass sich Zustände ändern, damit die Durchlaufbedingung im Schleifenkopf irgendwann nicht mehr erfüllt ist und die Schleife verlassen wird. Ansonsten entsteht eine Endlosschleife.

Schleifen mit `while` eignen sich dann, wenn nicht ermittelbar ist, wie oft die Schleife durchlaufen wird. Ein Beispiel:

```
$now = time();
$then = $now + 5;
$count = 0;
while($now < $then) {
  $file = fopen ("http://www.google.com/", "r");
  $count += 1;
  $now = time();
}
echo "$count mal Google geladen innerhalb 5 Sekunden";
```

Mit der PHP-Funktion `time()` wird ein Unix-Zeitstempel erzeugt, und zwar die aktuell vergangenen Sekunden seit dem 1. Januar 1970. Der Wert wird in `$now` gespeichert. In `$then` wird ein um 5 höherer Wert gespeichert, in dem zu `$now` 5 hinzuaddiert wird.

Unsere while-Schleife lassen wir nun 5 Sekunden lang laufen. Erreicht wird das, indem die Schleifendurchlaufbedingung mit `$now < $then` formuliert wird, also »solange der Wert von `$now` kleiner ist als der von `$then`«. Innerhalb des Schleifenkörpers wird in der letzten Anweisung jedes Mal wieder `$now = time()` ausgeführt, d.h., jedes Mal wird `$now` wieder mit einem neuen Unix-Zeitstempel aktualisiert. Dadurch ist `$now` natürlich nach irgendeiner Anzahl von Schleifendurchläufen gleich oder größer `$then`, was zum Abbruch der Schleife führt.

Innerhalb der Schleife wird die Startseite von *google.com* geladen. Auch hier begegnet uns übrigens wieder die »Leichtigkeit« von PHP: mit der gleichen Funktion (`fopen()`), die zum Öffnen lokaler Dateien dient, lassen sich auch beliebige Webseiten einlesen. Der Programmierer muss keine umständlichen Socketverbindungen erstellen, HTTP-GET-Kommandos erzeugen oder Sonstiges. Das alles erledigt PHP für ihn.

Da das Laden übers Internet natürlich einen Moment lang dauert, wird unsere while-Schleife innerhalb von 5 Sekunden nicht allzu oft durchlaufen.

Wenn die Schleifendurchlaufbedingung schon gleich beim ersten Prüfen nicht erfüllt ist, wird der Schleifenkörper kein einziges Mal ausgeführt. Da es jedoch Fälle gibt, in denen er wenigstens einmal ausgeführt werden soll, egal ob die Schleifendurchlaufbedingung zutrifft oder nicht, gibt es auch fußgesteuerte Schleifen mit `do ... while`:

```
$Zahl = 10;
do {
  echo $Zahl;
  $Zahl += 1;
}
while($Zahl < 10);
```

In diesem Beispiel würde die Schleife, wenn die Durchlaufbedingung für `while` oberhalb des Schleifenkörpers notiert wäre, kein einziges Mal ausgeführt, weil `$Zahl` schon höher ist als die Schleifendurchlaufbedingung erlaubt. Durch das Konstrukt über `do ... while` wird der Schleifenkörper jedoch zuerst ausgeführt, ehe die Durchlaufbedingung geprüft wird. Die Prüfung entscheidet dann über den nächsten Schleifendurchlauf.

12.3.6 Funktionen, Parameter und Return-Werte

PHP basiert letztlich auch auf C und C ist die Sprache der Funktionen. Das klassische C-Ideal besteht in einer Programmierung, die vorwiegend aus Funktionen besteht, von denen jede eine bestimmte, generische Aufgabe erfüllt und daher wiederverwendbar ist. Funktionen erwarten meistens einen oder mehrere Parameter als Input und liefern einen Return-Wert (Rückgabewert) als Output. Im Prinzip also soll eine typische Funktion ein kleines EVA-Programm innerhalb des Gesamtprogramms sein (EVA = Eingabe, Verarbeitung, Ausgabe – das Paradigma der klassischen Datenverarbeitung aus den Zeiten bildschirmloser, magnetbandgefütterter Rechner).

Nun stellt PHP schon sehr viele Funktionen bereit. Dennoch bieten sich bei der Programmierung genügend Gelegenheiten, eigene Funktionen zu definieren. Es genügt schon, wenn ein Set aus Anweisungen an mehreren Stellen im Script-Code wiederholt werden muss. In diesem Fall bietet es sich an, das Anweisungs-Set in eine Funktion zu schreiben und diese bei Bedarf aufzurufen. Die Funktion hat dann die Aufgabe eines Unterprogramms, einer Sub-Routine.

Schema

Eine typische Funktion hat folgenden Aufbau:

```
function Funktionsname($Parameter) {
   # Anweisungen;
  return($Wert)
}
```

Eingeleitet wird eine Funktionsdefinition durch das Schlüsselwort `function`. Dahinter folgt, durch Leerzeichen getrennt, ein frei wählbarer Funktionsname. Der Name kann wie ein Variablenname aus Buchstaben, Ziffern und Unterstrich bestehen, wobei der Name mit einem Buchstaben oder Unterstrich beginnen muss. Groß- und Kleinschreibung von Buchstaben werden ***nicht*** unterschieden, d.h., es ist egal, ob Sie eine Funktion mit `test()` oder `TEST()` aufrufen. Funktionsnamen müssen script-weit eindeutig sein. Wenn Sie beispielsweise andere Scripts über `include_once()` oder ähnliche Funktionen einbinden, dann dürfen in den anderen Scripts keine gleichnamigen Funktionen vorkommen. Nur Methoden verschiedener Objekte dürfen gleichnamig sein.

Der gesamte Funktionskörper muss in geschweifte Klammern eingeschlossen werden, auch dann, wenn die Funktion aus nur einer Anweisung besteht.

Falls die Funktion an die aufrufende Anweisung einen Wert zurückgeben will, kann sie dazu die PHP-Funktion `return()` verwenden. Als Parameter wird dieser Funktion übergeben, was zurückgegeben werden soll. Meistens ist das eine Variable, in der ein ermittelter Wert steht. Es kann jedoch auch eine literale Zeichenkette oder eine aus literalen Teilen und Variablen zusammengesetzte Zeichenkette sein. Wenn mehrere Werte zurückgegeben werden sollen, können diese in einem Array gesammelt werden. Zurückgegeben wird dann die Array-Variable. Durch `return()` wird die Funktion sofort verlassen.

Parameter und lokale Variablen

Variablen, die innerhalb eines Funktionskörpers definiert werden, gelten nur innerhalb dieser Funktion und sind außerhalb davon nicht bekannt. Innerhalb einer Funktion dürfen also durchaus Variablennamen vorkommen, die auch außerhalb davon vorkommen. Die Variablen außerhalb der Funktion werden durch diejenigen innerhalb der Funktion nicht berührt. Beispiel:

```
$i = 0;
function set_i() {
   $i = 23;
}
set_i();
echo $i;
```

Das Beispiel gibt 0 aus, weil die Funktion `set_i()`, die in ihrem Funktionskörper einer Variablen `$i` einen Wert zuweist, keinen Einfluss auf die Variable `$i` hat, die zuvor global mit 0 initialisiert wurde.

Auch Parameternamen, die bei einer Funktion definiert werden, sind lokale Variablen der Funktion.

Funktionsaufruf und Rückgabewert

Nachfolgendes Beispiel zeigt einen Komplettzusammenhang zwischen Funktionsaufruf, Funktion, Parametern und Rückgabewert (Return-Wert).

```
$mylink = set_link("http://www.example.org/", "Beispiel-Link");
echo $mylink;

function set_link($uri, $text) {
   $str = "<a href=\"$uri\">$text</a>";
   return($str);
}
```

Das Beispiel zeigt zunächst, dass ein Funktionsaufruf durchaus vor der Funktion selbst notiert werden darf. Der Grund dafür ist, dass PHP ein Script vor der Ausführung ohnehin erst einmal on the fly kompiliert. Dabei werden intern auch alle Funktionen und Variablen registriert.

Im Beispiel wird eine Funktion namens `set_link()` aufgerufen. Ihr werden in den runden Klammern hinter dem Funktionsnamen zwei Zeichenkettenparameter übergeben. Mehrere Parameter werden durch Kommata getrennt. Auch wenn keine Parameter übergeben werden, müssen die runden Klammern notiert werden.

Da die Funktion `set_link()` einen Rückgabewert erzeugt, kann sie wie im Beispiel einer Variablen zugewiesen werden. In der Variablen `$mylink` steht im Beispiel am Ende das, was die Funktion `set_link()` zurückgibt. Funktionen, die einen Wert zurückgeben, können aber auch direkt in Prozessanweisungen oder sogar als Parameter an andere Funktionen übergeben werden. So wäre es im obigen Beispiel sogar kürzer, gleich zu notieren:

```
echo set_link("http://www.example.org/", "Beispiel-Link");
```

Angenommen, es gibt eine zweite Funktion namens `set_link_style()`, die als ersten Parameter einen kompletten HTML-Hyperlink erwartet und als zweiten Parameter den Namen einer CSS-Klasse, die dem Link zugewiesen werden soll. Dann könnten Sie im Beispiel notieren:

```
$mylink = set_link_style(set_link("http://www.example.org/", "Beispiel-Link"),
"extern_link");
```

Als erster Parameter wird beim Aufruf von `set_link_style()` ein Aufruf von `set_link()` übergeben. Der PHP-Interpreter weiß, dass er in diesem Fall zuerst `set_link()` ausführen muss und den davon zurückgegebenen Wert in den Aufruf von `set_link_style()` einsetzen muss.

Default-Werte bei Parametern

Eine PHP-spezifische Besonderheit muss erwähnt werden. Bei Parameterdefinitionen im Funktionskopf können auch Defaultwerte angegeben werden. Ein Beispiel:

```
function line_numbered_file($file, $color="red") {
    $lines = file($file);
    $chars = strlen((string) count($lines));
    for($i = 1; $i <= count($lines); $i++) {
       $line = str_replace("%", "%%", $lines[$i]);
       $format = "<span style=\"color:".
                 $color."\">%".$chars."d</span> ".$line;
       $numbered_lines[] = sprintf($format, $i);
    }
    return($numbered_lines);
}
$file_lines = line_numbered_file("/etc/php.ini");
echo "<pre>";
foreach($file_lines as $line)
  echo $line;
echo "</pre>";
```

Die im Beispiel definierte Funktion `line_numbered_file()` versieht die Zeilen einer Textdatei mit führenden Zeilennummern. Dabei werden die Zeilennummern in HTML andersfarbig formatiert. Welche Textfarbe die Zeilennummern erhalten sollen, wird mit dem zweiten Parameter `$color` übergeben. Sollte dieser Parameter beim Funktionsaufruf fehlen, verwendet PHP den Default-Wert `red`. Grund dafür ist die Notation `$color="red"` in der Parameterdefinitionsliste in den runden Klammern der Funktion.

Normalerweise gibt PHP eine Warnung aus, wenn ein Funktionsaufruf eine Funktion nicht mit all den Parametern versorgt, die sie erwartet. Wenn jedoch bei der Funktionsdefinition mit der beschriebenen Syntax Defaultwerte für Parameter definiert werden, dann dürfen diese Parameter beim Aufruf auch weggelassen werden.

Auf den Inhalt der Funktion `line_numbered_file()` gehen wir an dieser Stelle nicht näher ein, weil uns das zu weit vom Thema abbringen würde.

Variable Anzahl von Parametern

Eine weitere Möglichkeit besteht darin, bei der Funktionsdefinition überhaupt keine Parameter zu definieren. Dennoch können der Funktion beim Aufruf Parameter übergeben werden, und zwar beliebig viele. PHP stellt einige Funktionen bereit, mit deren Hilfe eine Funktion auf ihr übergebene Parameter zugreifen kann. Ein Beispiel:

```
$a = 17;
$b = 64;
$c = 23;

function sum() {
   $x = 0;
   $params = func_get_args();
   foreach($params as $param)
      $x += $param;
   return($x);
}

echo sum($a, $b, $c);
```

Die Funktion `sum()` im Beispiel ist in der Lage, die Summe einer beliebigen Anzahl übergebener Zahlenwerte zu ermitteln und zurückzugeben. Dabei erwartet sie gar keine Parameter. Falls sie ohne Parameter aufgerufen würde, würde sie einfach 0 zurückgeben.

Mit der PHP-Funktion `func_get_args()` kann eine Funktion alle ihr übergebenen Parameter als Array ermitteln. In unserer Beispielfunktion wird der Array in `$params` gespeichert. In einer foreach-Schleife arbeitet die Funktion den Array `$params` ab und erhöht `$x` um den jeweiligen Wert, in der Annahme, dass lauter numerische Werte übergeben wurden.

Die echo-Ausgabe am Ende des Beispiels gibt 104 aus, weil der Funktion `sum()` beim Aufruf die drei zu Beginn initialisierten Variablen `$a`, `$b` und `$c` übergeben werden.

Zeiger (Referenzen) statt Werte übergeben

In Script-Sprachen wie PHP wird nicht so ausufernd mit so genannten Zeigern (Pointern) gearbeitet wie etwa in C. Dennoch bietet auch PHP die Möglichkeit an, einer Funktion anstelle eines Werts nur die Speicheradresse zu übergeben, an der der Wert beginnt. Sinnvoll ist dies beispielsweise, wenn als Parameter eine 1 Megabyte große Zeichenkette übergeben wird. Anstelle eines Megabyte werden nur ein paar Byte übergeben, eben die Arbeitsspeicheradresse der Variablen mit dem großen Inhalt. Damit PHP weiß, dass ein übergebener Wert nicht wirklich ein Wert ist, sondern nur eine Arbeitsspeicheradresse, muss er kenntlich gemacht werden. Ein Beispiel:

```
$satz = "Ein langer Satz mit vielen Wörtern";
function count_words(&$str) {
```

```
    return(count(explode(" ",$str)));
}
echo count_words($satz);
```

Die Zeigertechnik wird bei der Parameterdefinition der Funktionsdefinition angewendet. Im Beispiel bewirkt `&$str`, dass die Funktion beim Aufruf keine Kopie des übergebenen Werts erhält, sondern dessen Arbeitsspeicheradresse. Das kaufmännische Und (`&`) vor dem Parameternamen genügt, um den Parameter als Referenz bzw. Zeiger zu deklarieren.

Beim Aufruf der Funktion muss (im Gegensatz etwa zu C) nichts beachtet werden. Im Beispiel wird der Funktion einfach ein Variablenname übergeben.

Wichtig zu wissen ist jedoch, dass Funktionen, die eine als Referenz übergebene Variable ändern, nicht die Kopie ändern, sondern das Original. In manchen Fällen kann dies sogar explizit gewünscht sein, in anderen dagegen nicht.

12.3.7 Klassen und Objekte in PHP 4

In PHP 4.x ist die Möglichkeit objektorientierter Programmierung zwar im Prinzip angelegt, doch fehlen noch entscheidende Features wie Regelungen zur Sichtbarkeit von Variablen. Dennoch lohnt es sich bei größeren Projekten, gleich mit objektorientierter Programmierung zu beginnen.

Grundbegriffe

Eine ***Klasse*** ist in der objektorientierten Programmierung das, was in der Textverarbeitung eine Dokumentvorlage ist. So, wie in einer Dokumentvorlage Seitenformate, Absatz- und Zeichenformate sowie weitere Vorgaben definiert werden, werden in einer Klasse Datenstrukturen und Funktionen definiert.

Ein ***Objekt*** ist dann ein konkretes Dokument. So, wie ein Dokument aus einer Dokumentvorlage erstellt werden kann, kann ein Objekt aus einer Klasse erzeugt werden. Andere Bezeichnungen für »Objekt« sind ***Instanz*** und ***Exemplar***.

Eine ***Objekteigenschaft*** oder einfach ***Eigenschaft*** ist eine Variable, die in der Klasse definiert wurde, aus der das Objekt erzeugt wurde.

Eine ***Objektmethode*** oder einfach ***Methode*** ist eine Funktion, die in der Klasse definiert wurde, aus der das Objekt erzeugt wurde.

Klassen und Objekte anlegen

Beginnen wir mit einem ganz einfachen Beispiel, um das Prinzip zu demonstrieren:

```
class MiniClass {
   var $test = 23;
}

$mini_object = new MiniClass;
echo $mini_object->test;
```

In diesem Beispiel wird eine Klasse mit dem Namen `MiniClass` definiert. Eine Klassendefinition beginnt mit dem Schlüsselwort `class`. Dahinter folgt, durch Leerzeichen getrennt, der gewünschte Klassenname. Bei der Wahl des Klassennamens sind Sie frei. Klassennamen müssen jedoch innerhalb eines gesamten Script-Kompilats eindeutig sein. Ansonsten gelten die gleichen Regeln wie bei Variablen und Funktionsnamen. Es hat sich jedoch eingebürgert, Klassennamen mit Großbuchstaben beginnen zu lassen.

Der gesamte Inhalt einer Klasse muss in geschweifte Klammern eingeschlossen werden, genauso wie bei einer Funktion.

Innerhalb einer Klasse darf nichts anderes notiert sein als Variablendefinitionen oder Funktionen. In unserem Minimalbeispiel ist lediglich eine einzige Variablendefinition notiert. Variablendefinitionen innerhalb von Klassen werden durch das Schlüsselwort `var` eingeleitet. Ansonsten gelten die gleichen Regeln wie bei normalen Variablen. Variablen können mit einem Wert initialisiert werden wie im Beispiel. Zwingend erforderlich ist das jedoch nicht. Wenn eine Variable zunächst keinen Wert haben soll, notieren Sie einfach `var $name;` – also direkt hinter dem Variablennamen das Semikolon zum Abschließen der Anweisung.

Wenn Sie Klassenvariablen mit einem Wert initialisieren wollen wie im Beispiel, darf dies nur ein konstanter Wert sein, also etwa eine Zahl oder eine Zeichenkette, nicht aber ein ermittelter Wert, wie er z.B. durch Aufruf von PHP-Funktionen oder anderen, eigenen Funktionen zustande käme. Falls Sie den Initialisierungswert der Variablen doch ermitteln lassen wollen, benötigen Sie eine Konstruktorfunktion. Dazu weiter unten mehr.

In unserem Minimalbeispiel wird unterhalb der Klasse eine Variable namens `$mini_object` definiert. Als Wert wird ihr `new MiniClass` zugewiesen. Mit dem Schlüsselwort `new` signalisieren Sie, dass Sie aus dieser Variable eine Objektvariable machen möchten. Dahinter geben Sie den Klassennamen an, aus dem ein Objekt erzeugt werden soll. In unserem Beispiel wird `$mini_object` zu einer Objektvariable für die Klasse `MiniClass`.

Die Variablendefinition `$mini_object = new MiniClass` ist also nichts anderes als das Erzeugen eines Objekts aus der Klasse `MiniClass`. Das Objekt ist in der Variablen `$mini_object` gespeichert. Über diese Objektvariable ist der Zugriff auf die Eigenschaften (Variablen) und Methoden (Funktionen) der Klasse möglich.

Der Zugriff erfolgt über den »Zeige-Operator« `->` (Minuszeichen und Größer-als-Zeichen direkt hintereinander notiert). In unserem Minibeispiel wurde in der Klasse `MiniClass` eine Variable `$test` definiert. Aus Sicht der Objektvariablen `$mini_object` ist das eine Objekteigenschaft. Durch die Syntax `$mini_object->test` können Sie auf den Wert der Eigenschaft zugreifen. Beachten Sie dabei, dass vor dem Namen der Eigenschaft, hier also `test`, kein $-Zeichen notiert werden darf!

Das Beispiel gibt 23 aus, weil es einfach `$mini_object->test` ausgibt und darin die in der Klassendefinition festgelegte Vorbelegung gespeichert ist. Die Objekteigenschaft kann jedoch wie jede andere Variable auch geändert werden.

Klassen mit Konstruktortfunktion

Eine Klasse darf wie schon erwähnt nichts anderes enthalten als Variablendefinitionen mit `var` und Funktionen. Wenn innerhalb der Klasse eine Funktion definiert wird, die exakt den gleichen Namen hat wie die Klasse selbst, dann gilt diese Funktion als so genannte *Konstruktorfunktion*. Das Besondere an der Konstruktorfunktion ist, dass sie beim Anlegen eines neuen Objekts mit `$variable = new $Klassenname` automatisch mit ausgeführt wird. Ein Beispiel:

```
class MiniClass {
   var $test;
   function MiniClass() {
      $this->test = ini_get("max_execution_time");
   }
}
$mini_object = new MiniClass;
echo $mini_object->test;
```

Die Klasse `MiniClass` enthält nach wie vor eine Variablendefinition, wobei die Variable `$test` jedoch jetzt nicht mehr initialisiert wird. Sie soll zwar initialisiert werden, aber mit einem ermittelten Wert, was bei der Definition mit `var` innerhalb der Klasse nicht erlaubt ist. Die Initialisierung holen wir deshalb in einer Konstruktorfunktion nach. Diese ist im Beispiel die Funktion `MiniClass()`, weil sie exakt genauso heißt wie die Klasse, innerhalb deren sie definiert wird.

Innerhalb der Konstruktorfunktion wird die Variable `$test` initialisiert, und zwar zum Spaß mit einem Aufruf der PHP-Funktion `ini_set()`. Diese holt Werte aus der *php.ini*. In unserem Fall ermittelt sie, wie lange das Script maximal laufen darf, bevor es gewaltsam beendet wird.

Wichtig zu kennen ist die Syntax: Wenn innerhalb einer Klasse auf Klassenvariablen zugegriffen werden soll, also auf Variablen, die außerhalb von Funktionen mit `var` definiert wurden, dann muss das mit `$this->variablenname` geschehen.

Das Beispiel-Script gibt am Ende den Wert aus der *php.ini* aus, der dort bei der Direktive *max_execution_time* zugewiesen ist.

Konstruktorfunktion mit Parametern

Eine Konstruktorfunktion, so haben wir schon gelernt, ist nicht zwingend erforderlich. Sie ist beispielsweise dann nötig, wenn Initialisierungswerte von Klassenvariablen durch Funktionsaufrufe oder Ähnliches ermittelt werden sollen. Ein anderer wichtiger Grund, eine Konstruktorfunktion zu verwenden, wäre der Wunsch, Parameter an eine Klasse zu übergeben. Auch dazu ein Beispiel:

```
class RGBColor {
   var $R = 0;
   var $G = 0;
   var $B = 0;
```

```
    function RGBColor($r, $g, $b) {
        if($r >= 0 and $r <= 255)
           $this->R = $r;
        if($g >= 0 and $g <= 255)
           $this->G = $g;
        if($b >= 0 and $b <= 255)
           $this->B = $b;
    }
}
$my_blue = new RGBColor(40,80,220);
echo $my_blue->B;
```

In diesem Beispiel wird eine Klasse namens `RGBColor` definiert, in der ein RGB-Wert gespeichert werden kann. Sie enthält neben den drei Klassenvariablen `$R`, `$G` und `$B` für jeden Farbanteil eine gleichnamige Funktion `RGBColor`, die dadurch ihre Konstruktorfunktion ist. Diese Funktion erwartet drei Parameter (für die drei Farbanteile).

Beim Erzeugen des Objekts wird die Klasse dann wie eine Funktion aufgerufen. Nur durch das Schlüsselwort `new` davor ist noch erkennbar, dass eine Klasse und keine Funktion aufgerufen wird.

Die Konstruktorfunktion bekommt die beim Klassenaufruf übergebenen Parameter übergeben. In unserem Beispiel prüft sie, ob sich die Werte im erlaubten Bereich zwischen 0 und 255 befinden. Wenn ja, weist sie den Klassenvariablen die übergebenen Werte zu.

Abgeleitete und erweiternde Klassen

Angenommen, Sie haben folgende Klasse:

```
class Person {
   var $given_name;
   var $family_name;
   var $birthday_dd;
   var $birthday_mm;
   var $birthday_yyyy;
}
```

Sie haben möchten nun außerdem eine Klasse für Kontaktdaten anlegen. Dann bietet Ihnen das Konzept der objektorientierten Programmierung an, die existierende Klasse `Person` zu kopieren und zu erweitern. Beispiel:

```
class Contact extends Person {
   var $street;
   var $zip_code;
   var $location;
   var $phone_number;
   var $mobile_number;
   var $mail_address;
}
```

Bei der Definition einer Klasse können Sie hinter dem Klassennamen das Schlüsselwort `extends` und dahinter den Namen einer anderen existierenden Klasse angeben. Dadurch binden Sie alle dort definierten Klassenvariablen (Eigenschaften) und Funktionen (Methoden) in die neu anzulegende Klasse mit ein. Die neue Klasse kann dann zusätzliche Variablen und Funktionen definieren.

Im Beispiel wird eine Klasse `Contact` definiert, welche die Klasse `Person` erweitert. Innerhalb von `Contact` kann nun z.B. innerhalb einer Methode auf `$this->given_name` zugegriffen werden, da alle Definitionen aus `$Person` in `$Contact` übernommen wurden. Ebenso verhält es sich bei Objekten, die aus `Contact` erzeugt werden. Ein Beispiel:

```
$julia = new Contact;
$julia->given_name = "Julia";
echo $julia->given_name;
```

Wichtig zu wissen ist noch, wie es sich bei abgeleiteten Klassen mit Konstruktorfunktionen verhält. Hat die abgeleitete Klasse eine eigene Konstruktorfunktion, so wird diese beim Erzeugen eines Objekts aufgerufen, die der Elternklasse dagegen nicht. Gibt es in der abgeleiteten Klasse dagegen keine Konstruktorfunktion, so wird, falls die Elternklasse eine Konstruktorfunktion besitzt, diese aufgerufen. Vor allem Letzteres müssen Sie als Programmierer wissen, denn wenn die Konstruktorfunktion der Elternklasse Parameter erwartet, müssen Sie die abgeleitete Klasse beim Erzeugen eines Objekts mit den Parametern aufrufen, die von der Konstruktorfunktion der Elternklasse erwartet werden.

Direktaufruf von Methoden

Mithilfe des Operators `::` ist es möglich, Methoden aus Klassen direkt aufzurufen, ohne ein Objekt der Klasse zu erzeugen. Beispiel:

```
class Member {
   var $name = "Max Mustermann";
   function show_name() {
      echo "bla";
   }
}
Member::show_name();
```

Mit der Syntax `Klassenname::Methodenname()` kann eine Methode aus einer Klasse direkt ausgeführt werden. Allerdings mit einer Einschränkung: Die Methode darf nicht mit `$this->Variablenname` auf Klassenvariablen Bezug nehmen, da die Klassenvariablen ohne Objektinstanz nicht existieren. Das obige Beispiel gibt `bla` aus. Würde in `show_name()` jedoch stehen: `echo $this->name`, dann würde beim Aufruf über `Member::show_name()` gar nichts ausgegeben, da in diesem Aufrufkontext keine Klassenvariable `$name` existiert.

Praxisbeispiel: Template-Klasse

In einem kleinen Praxisbeispiel greifen wir auf das Beispiel aus ***Abschnitt 12.2.5*** zurück. Dort haben wir bereits gesehen, wie sich PHP-Code bei größeren Projekten sinnvoll auf

mehrere Scripts verteilen lässt. Dies wollen wir nun auch in Zusammenhang mit Klassen und Objekten demonstrieren.

Das Beispiel besteht aus folgenden PHP-Scripts:

- *linkweb.php:* das zentrale Script, das auch im Browser aufgerufen wird
- *class.template.php:* Script mit der Klasse »Template« für HTML-Templates
- *class.page.php:* Script mit Klasse »Page« zum Zusammenkleben einer Seite aus Template und Inhalt
- *class.error.php:* Script mit Klasse »Error« zur Ausgabe von Fehlermeldungen

Die übrigen Dateien für Templates, Inhaltsdateien und CSS-Datei bleiben wie gehabt. Zunächst der Quelltext des zentralen Scripts:

Listing 12.18: Datei linkweb.php

```
<?php
include_once("class.template.php");
include_once("class.page.php");
$main_template = new Template;
$main_page = new Page;

$content_files = array();
$content_files['home'] = "0001.txt";
$content_files['impressum'] = "0007.txt";
$content_files['themen'] = "0003.txt";
$content_files['branchen'] = "0009.txt";
$content_files['auskunft'] = "0005.txt";
$content_files['geo'] = "0002.txt";
$content_files['wissen'] = "0004.txt";
$content_files['literatur'] = "0008.txt";
$content_files['lexika'] = "0006.txt";
$content_files['glossare'] = "0010.txt";
$content_files['wikis'] = "0014.txt";
$content_files['ftp'] = "0011.txt";
$content_files['wais'] = "0015.txt";
$content_files['newsgroups'] = "0012.txt";
$content_files['foren'] = "0013.txt";

if(isset($_GET['page']))
   $get_page = $_GET['page'];
else
   $get_page = 'home';
$main_template->template_file = "linkweb.tpl";
$main_page->page_content = $main_template->get_template();
$main_page->set_var('content',
            file_get_contents($content_files[$get_page]));
$main_page->set_var('title', $main_page->get_title());
$main_page->show_page();
exit();
?>
```

Dieses Script enthält selber keine Klasse, sondern regelt nur den Gesamtablauf. Die Definition des Arrays `$content_files` haben wir der Einfachheit halber mit in das zentrale Script übernommen – bei größeren Projekten empfiehlt sich aber, wie früher erwähnt, ein eigenes PHP-Script für globale Variablendefinitionen.

Die eigentliche Arbeit des Gesamt-Scripts wird jedoch von Script-Dateien erledigt, die im Wesentlichen aus Klassen bestehen. Im Zentralscript werden oben die Scripts ***class.template.php*** und ***class.page.php*** eingebunden. Die Dateinamen sind natürlich frei wählbar. Es ist jedoch guter Stil, PHP-Scripts, die eigentlich nur eine Klasse enthalten, im Dateinamen entsprechend zu bezeichnen.

Unterhalb der include-Anweisungen sind denn auch gleich die Definitionen der benötigten Objektvariablen notiert. In `$main_template` wird ein Objekt der Klasse `Template` gespeichert und in `$main_page` ein Objekt der Klasse `Page`.

Der Script-Ablauf im Zentral-Script besteht wie schon im früheren Beispiel darin, zunächst den übergebenen GET-Parameter `page` abzufragen, um zu ermitteln, welche Seite angezeigt werden soll.

Dann wird mit den beiden erzeugten Objekten gearbeitet. Zunächst wird der Eigenschaft `template_file` des Template-Objekts die gewünschte Template-Datei zugewiesen. Über die Methode `get_template()` wird die Template-Datei eingelesen. Der Rückgabewert der Methode ist die eingelesene Datei. Dieser wird gleich der Eigenschaft `page_content` des Page-Objekts zugewiesen. So lesen wir in einer einzigen Anweisung das HTML-Template ein und kopieren es in den zu erstellenden Seiteninhalt.

Nun müssen die aus dem Template kopierten Platzhalter des Typs `[%name%]` durch seitenabhängige Inhalte ersetzt werden. Dies besorgt die Methode `set_var()` des Page-Objekts. Ihr werden zwei Parameter übergeben: der Platzhaltername, bei `[%name%]` also `'name'`, und der gewünschte Wert, durch den der Platzhalter ersetzt werden soll. Das Zentral-Script muss die Platzhalter `[%content%]` und `[%title%]` ersetzen. Bei `[%content%]` setzt es seitenabhängig die passende Datei aus dem Array `$content_files` ein und bei `[%title%]` kann es den Inhalt durch Aufruf der Methode `get_title` des Page-Objekts ermitteln. Die Methode `get_title` sucht im bislang vorhandenen HTML-Code der Seite nach einer h1-Überschrift, ermittelt deren Inhalt und gibt ihn zurück.

Am Ende muss das Zentral-Script nur noch die Methode `show_page()` des Page-Objekts aufrufen, um seine Arbeit abzuschließen und die gewünschte Seite auszugeben.

Das gesamte Script-Ensemble wird natürlich erst verständlich durch die Quelltexte der Dateien mit den Klassen. Zunächst die Template-Klasse:

Listing 12.19: Script class.template.php mit Klasse »Template«

```
<?php
#-----------------------------------------
# Includes:
include_once("class.error.php");
$template_error = new Error;
```

```
#-----------------------------------------
# Klasse Template:

class Template {
   var $template_file;
   var $template_content;

   function get_template() {
      global $template_error;
      if(empty($this->template_file))
         $template_error->handle_error
         ('Keine Template-Datei angegegeben!');
      if(! file_exists($this->template_file))
         $template_error->handle_error
         ('Template-Datei nicht gefunden!');
      $this->template_content =
             file_get_contents($this->template_file);
      return($this->template_content);
   }
}
?>
```

Die Script-Datei zeigt den typischen Aufbau einer Script-Datei, die lediglich eine Klasse enthält. Im globalen Bereich vor der Klassendefinition werden andere Scripts eingebunden, die mehrfach benötigt werden. In unserem Fall wird ***class.error.php*** benötigt. Da dies auch eine Klasse ist, muss natürlich auch ein Objekt daraus erzeugt werden, das in der Objektvariablen `$template_error` gespeichert wird.

Umgekehrt werden wir in ***class.error.php*** sehen, dass dort ein Template zur Ausgabe der Fehlerseite benötigt wird und dass dazu ein Template-Objekt in der Variablen `$error_template` erzeugt wird. Eine solche Stringenz sowohl bei Dateinamen als auch bei Variablennamen ist bei größeren Projekten dringend zu empfehlen, da ansonsten leicht Probleme mit doppelt vorkommenden Variablen vorkommen können und die Gesamtübersicht einfach leichter verloren geht.

Die Klasse `Template` wird durch `class Template { ... }` definiert. Der Inhalt besteht aus zwei Klassenvariablen (Objekteigenschaften) namens `$template_file` und `$template_content`. In der einen muss ein Template-Objekt den Pfadnamen der gewünschten Template-Datei ablegen und in der anderen wird nach Aufruf der einzigen Klassenfunktion (Objektmethode) `get_template()` der Inhalt der Template-Datei gespeichert.

Eine Konstruktorfunktion wird in dieser Klasse nicht benötigt.

Als Nächstes der Quelltext der Error-Klasse:

Listing 12.20: Script class.error.php mit Klasse »Error«

```
<?php
#-----------------------------------------
# Includes:
include_once("class.template.php");
include_once("class.page.php");
```

```
$error_template = new Template;
$error_page = new Page;

#----------------------------------------
# Klasse Error:

class Error {
   var $error_template = "error.tpl";

   function handle_error($message) {
      global $error_template;
      $error_template->template_file = $this->error_template;
      $error_page->page_content =
           $error_template->get_template();
      $error_page->set_var('message', $message);
      $error_page->show_page();
      exit();
   }
}
?>
```

Das Script mit der Error-Klasse benötigt die Klassen `Template` und `Page`, bindet dazu die nötigen Script-Dateien ein und erzeugt die erforderlichen Objekte in den Objektvariablen `$error_template` und `$error_page`.

Die Error-Klasse selbst besteht aus einer Klassenvariable für den Pfadnamen der Error-Template-Datei sowie aus der Funktion `handle_error()`. Deren Aufgabe ist eine ähnliche wie die des Zentral-Scripts. Sie muss die Error-Template-Datei einlesen und in die Objekteigenschaft `page_content` des erzeugten Page-Objekts kopieren. Dann muss sie noch mit der set_var-Methode des Page-Objekts den Platzhalter `[%message%]` durch die Fehlermeldung ersetzen, die beim Aufruf von `handle_error()` übergeben werden muss, und zuletzt kann sie die Fehlerseite ausgeben und den Scriptlauf beenden.

Nun noch der Quelltext der Page-Klasse:

Listing 12.21: Script class.error.php mit Klasse »Page«

```
<?php
#----------------------------------------
# Klasse Page:

class Page {
   var $page_content;
   var $standard_title = "untitled";

   function set_var($name, $value) {
      if(empty($this->page_content))
         return;
      else {
         $this->page_content =
                str_replace("[%$name%]",
                $value, $this->page_content);
```

```
                return($this->page_content);
            }

        }

        function show_page() {
            if(empty($this->page_content))
                return;
            else
                echo $this->page_content;
        }

        function get_title() {
            if(empty($this->page_content))
                return;
            else {
                $pattern = "/<h1>(.*)<\/h1>/";
                preg_match($pattern, $this->page_content, $matches);
                if(isset($matches[1]))
                    return($matches[1]);
                else
                    return($this->standard_title);
            }
        }
    }
    ?>
```

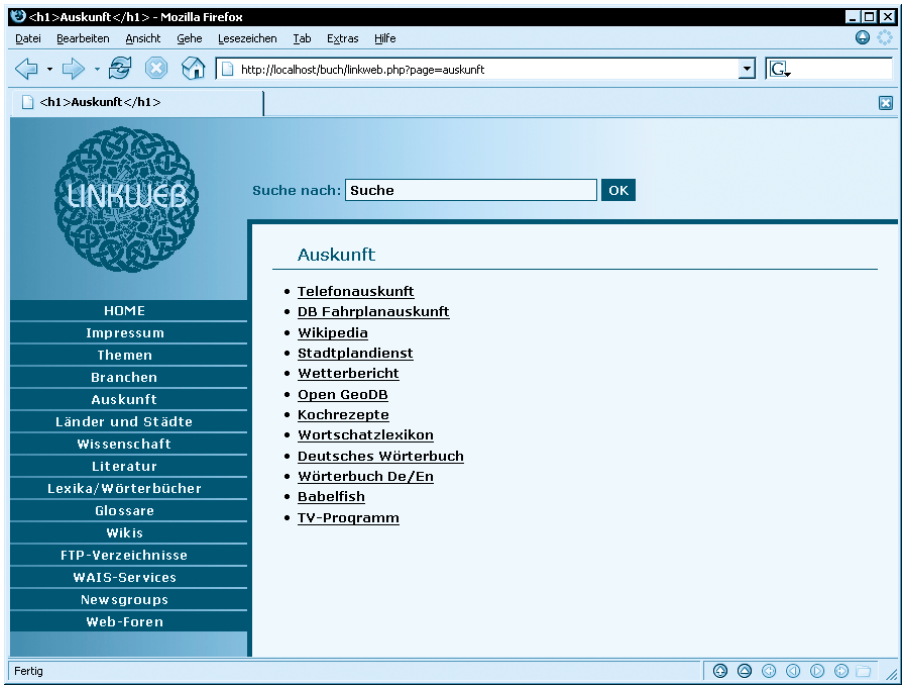


Abbildung 12.14: Beispielprojekt »Linkweb«, diesmal mit Klassen und Objekten realisiert

Die Page-Klasse benötigt keine anderen Scripts. Die Klasse selbst besteht aus den Klassenvariablen `$page_content` und `$standard_title`. In `$page_content` wird der komplette HTML-Code einer auszugebenden Seite gespeichert. In diese Variable muss zunächst der durch Einlesen des Templates gewonnene Inhalt kopiert werden, bevor die Methoden des Page-Objekts sinnvoll anwendbar sind. Die Initialisierung von `$standard_title` soll nur einen Defaultwert für den Titel vorgeben, falls die Funktion `get_title()` keine h1-Überschrift ermitteln kann, aus der sie den Titel gewinnen kann.

Die Funktionen der Page-Klasse sind bereits aus den Aufrufen als Objektmethoden bekannt: `set_var()` erlaubt es, Platzhalter des Schemas `[%name%]` durch Inhalte zu ersetzen. Die Methode aktualisiert dabei die Klassenvariable `$page_content`, gibt deren Wert aber zusätzlich auch an eine aufrufende Anweisung zurück. `show_page()` gibt eine fertig erzeugte Seite aus und `get_title()` sucht nach einer h1-Überschrift und gibt deren Inhalt zurück.

12.3.8 Objektorientierung in PHP 5

Basis von PHP 4.x ist die so genannte Zend-Engine, das Herzstück des PHP-Interpreters. PHP 5.x verdankt seinen ganzzahligen Versionssprung der Tatsache, dass darin eine Zend-Engine 2 steckt. Ein Hauptmerkmal der Zend-Engine ist, dass sie ein PHP-Script nicht einfach Zeile für Zeile einliest und ausführt, sondern dass sie erst mal vom aufgerufenen PHP-Script ausgehend alles kompiliert, was dazu gehört, d.h., das Script selber, alle von ihm eingebundenen Scripts und alle von eingebundenen Scripts eingebundene Scripts werden zu einem Quelltext-Ensemble vereinigt, Ganzes syntaktisch überprüft und bei Erfolg zu Objektcode kompiliert.

Erst dadurch wurde überhaupt eine Abkehr von rein prozeduraler Programmierung in PHP möglich. Wie wir im vorherigen Abschnitt gesehen haben, ist die Implementierung nicht-prozeduraler, sondern objektorientierter Programmierung in PHP 4.x zwar durchaus möglich. Doch es fehlen dort noch wichtige Features, über die andere, heute führende OOP-Sprachen wie C++ oder Java längst verfügen (OOP = *objektorientierte Programmierung*).

PHP 5.x bietet PHP dem Programmierer alle Wege an, angefangen von einzelnen kleinen PHP-Bereichen in ansonsten statischem HTML-Code, über normale prozedurale Scripts bis hin zu streng objektorientierter Arbeitsweise.

Wichtige neue Features der Objektorientierung in PHP 5.x sind:

- *Datenkapselung (Sichtbarkeit):* Bei der Implementierung in PHP 4.x kann ein Objekt alle Klassenvariablen als Eigenschaften lesen und ändern und es kann alle in der Klasse definierten Funktionen als Methoden ausführen. Dies ist jedoch nicht immer wünschenswert. Manche Klassenvariablen und Funktionen sollen nur internen Zwecken dienen. Dazu führt PHP 5.x die aus anderen OOP-Sprachen bekannten Schlüsselwörter `public`, `private` und `protected` ein (Beispiel siehe weiter unten).
- *Objekt-Handles:* Bei größeren Projekten kann es vorkommen, dass ein Objekt unter mehreren Objektvariablen ansprechbar sein soll. In PHP 4.x lassen sich Objektvari-

ablen zwar einer anderen Variable zuweisen, doch dadurch entsteht lediglich eine Kopie und beide Kopien verwalteten letztlich unterschiedliche Objekte. In PHP 5.x bewirkt eine Zuweisung wie `$objvar2 = $objvar`, dass in `$objvar2` das gleiche Objekt gespeichert ist wie in `$objvar` und dass beide Variablen nur unterschiedliche Handles für den Zugriff auf das gleiche Objekt sind.

- ***Statische Klassenvariablen und -funktionen:*** Jedes Objekt einer Klasse ist normalerweise eine Kopie der Klasse. Alle Klassenvariablen kann das Objekt als seine objekteigenen Objekteigenschaften verwalten. Es kann jedoch wünschenswert sein, dass eine Klassenvariable einen Wert zentral verwaltet. Beliebtes Beispiel sind laufende Nummern, die beim Erzeugen neuer Objekte der Klasse jeweils hochgezählt werden, etwa als Basis für eindeutige Datensatznummern. Statische Variablen existieren deshalb nicht wie üblich als Kopie in jedem Objekt, sondern sind nur einmal verfügbar, nämlich in der Klasse selbst. Definiert werden sie durch das Schlüsselwort `static`. Die entsprechende Möglichkeit ist in PHP5.x neu implementiert worden und in PHP 4.x noch nicht verfügbar.

- ***Finale Klassen und Methoden:*** Im Abschnitt über Objektorientierung in PHP 4.x haben wir bereits das Konzept der abgeleiteten Klassen kennen gelernt. Wenn solche abgeleiteten Klassen Funktionen (Methoden) enthalten, die den gleichen Namen haben wie diejenigen in der Elternklasse, dann wird dies als ***Überschreiben*** der Funktion der Elternklasse bezeichnet. In vielen Fällen kann dies sogar explizit gewünscht sein, in anderen jedoch nicht. Um eine Methode aus einer Elternklasse davor zu schützen, von einer gleichnamigen Methode in einer Kindklasse überschrieben zu werden, kann die Methode seit PHP 5.x mit dem Schlüsselwort `final` definiert werden, was so viel wie Überschreibschutz bedeutet. Auch ganze Klassen lassen sich davor schützen, dass abgeleitete Klassen von ihnen erzeugt werden. Für diese Fälle ist das Schlüsselwort `static` auf die Definition der gesamten Klasse anwendbar.

- ***Abstrakte Klassen:*** Solche Klassen sind sozusagen das Gegenteil von finalen Klassen. Ihre Aufgabe ist nicht, zu verhindern, dass andere Klassen davon abgeleitet werden können, sondern dass Objekte von ihr abgeleitet werden können. Abstrakte Klassen dienen ausschließlich dazu, dass andere Klassen von ihr abgeleitet werden können. In einem komplexen Objektentwurfsmodell können solche Basisklassen eine wichtige Rolle spielen.

Neben der Objektorientierung führt PHP 5.x auch auf anderen Ebenen Funktionalitäten ein, die sich in anderen Programmiersprachen bewährt haben. Darunter etwa den try-catch-throw-Mechanismus. Hier soll uns jedoch nur die Objektorientierung interessieren. Nachfolgend einige Beispiele, welche die neuen Features bei der Objektorientierung verdeutlichen sollen.

Beispiel: Datenkapselung

Angenommen, für eine webbasierte Bibliotheksverwaltung mit PHP 4.x wird unter anderem folgende wichtige Klasse definiert:

```
class Book {
   var $isbn;
   var $title;
   var $sub_title;
   var $author;
   var $editor;
   var $publisher;
   var $publishing_year;
   var $bib_code;
   var $borrow_count;
   var $borrow_days;
   var $borrowed_flag;
   var $borrowed_since;

   function get_book_data_by_isbn() {
      # Code ...
   }
   function insert_book_data() {
      # Code ...
   }
}
```

Der Quelltext stellt natürlich nur einen Ausschnitt aus der Klasse dar. Erkennbar ist, dass bei den Klassenvariablen neben denen für Daten, die zur Identifizierung eines Buchs typisch sind, auch solche für die Ausleihe vorgesehen sind. `$borrow_count` soll speichern, wie oft ein Buch bereits ausgeliehen wurde, `$borrowed_days` die Gesamtzahl der Tage, die das Buch ausgeliehen war, `$borrowed_flag` einen Boolean-Wert, der angibt, ob das Buch momentan ausgeliehen ist, und im Fall, dass es gerade ausgeliehen ist, soll `$borrowed_since` das Datum speichern, an dem die aktuelle Ausleihe begonnen hat.

Wenn nun ein Objekt dieser Klasse erzeugt wird, so sind alle Klassenvariablen darin als Objekteigenschaft lesbar und änderbar. Nehmen wir an, die im Beispiel angedeutete Methode `get_book_data_by_isdn()` ist eine der Methoden, die Buchdaten aus einer Datenbank holen und die Objekteigenschaften mit gefundenen Daten versorgen kann. Dann wäre es kein Problem, über die Objektvariable, nennen wir sie `$search_book`, zur Laufzeit des Scripts Eigenschaften wie `$isbn` oder `$borrow_count` einfach zu ändern.

Solange ein einziger Programmierer an dem Projekt arbeitet und weiß, was er tut, stellt das kein Problem dar. Sollen jedoch viele Entwickler mit dieser Klasse arbeiten, besteht durchaus die Gefahr, dass unbeabsichtigt oder beabsichtigt ermittelte Objektdaten z.B. vor der Ausgabe oder Weiterverarbeitung geändert, sprich manipuliert werden.

Um Klassenvariablen davor zu schützen, können sie wie im folgenden, abgewandelten Beispiel auf Basis von PHP 5.x definiert werden:

```
class Book {
   private $isbn;
   private $title;
   private $sub_title;
   private $author;
```

```
    private $editor;
    private $publisher;
    private $publishing_year;
    private $bib_code;
    private $borrow_count;
    private $borrow_days;
    private $borrowed_flag;
    private $borrowed_since;

    function get_book_data_by_isbn() {
        # Code ...
    }
    function insert_book_data() {
        # Code ...
    }
    function get_title() {
        return($this->title);
    }
    function get_borrow_count() {
        return($this->borrow_count);
    }
    protected function delete_data() {
        # Code ...
    }
    # usw.
}
```

Anstelle des Schlüsselworts `var` vor dem Namen der Klassenvariable wird in diesem Beispiel stets `private` notiert. Damit wird erreicht, dass die Klassenvariable in Objekten, die aus dieser Klasse erzeugt werden, nicht als Objekteigenschaft zur Verfügung steht. Sie dient lediglich intern der Speicherung von Daten. Ein Objekt kann diese Daten also weder lesen noch ändern. Nun muss in einem solchen Fall natürlich die Möglichkeit eingerichtet werden, zumindest lesend an die Daten zu gelangen. Dazu können die typischen so genannten ***get-Funktionen*** (oder ***Gettern***) als Methoden definiert werden.

Im Beispiel sind `get_title()` und `get_borrow_count()` als Vertreter dieser Art notiert. Solche get-Funktionen tun nichts weiter, als eine nach außen geschützte Variable auszulesen und ihren Wert zurückzugeben. Auf diese Weise ist einer Objektvariablen das Auslesen von Objektdaten über eine get-Funktion möglich, nicht aber das Ändern.

Analog zu den get-Funktionen kommen häufig auch ***set-Funktionen*** (***Setter***) zum Einsatz, die das Ändern von Objektdaten auf Basis eines als Parameter übergebenen Inputs erlauben. Die Funktion hat dabei jedoch die Möglichkeit, kontrollierend einzugreifen und Daten nur dann zu ändern, wenn der übergebene Wert einem bestimmten Format entspricht, einen bestimmten Wertebereich einhält oder einer Plausibilitätsprüfung stand hält.

Mit `private` definieren Sie Klassenvariablen, auf die nur innerhalb der Klasse selbst, also in Funktionen innerhalb der Klasse zugegriffen werden kann.

Mit `protected`, an gleicher Stelle notiert, definieren Sie Klassenvariablen, auf die zusätzlich auch in abgeleiteten Klassen zugegriffen werden kann. Wenn sie nämlich eine Klasse von einer anderen ableiten, stehen Klassenvariablen, die in der Elternklasse mit `private` definiert wurden, in der abgeleiteten Kindklasse nicht mehr zur Verfügung.

Mit `public`, ebenfalls an gleicher Stelle notiert, definieren Sie Klassenvariablen, die als normale Objekteigenschaften zur Verfügung stehen und von jedem Objekt gelesen und durch einfache Wertzuweisung geändert werden können.

Das Schlüsselwort `var`, das in PHP 4.x an gleicher Stelle notiert werden muss, ist in PHP 5.x ***nicht mehr erlaubt***!

Auch Funktionen von Klassen können sichtbar oder nicht sichtbar sein. Im Beispiel wird eine Funktion `delete_data()` als `protected` definiert. Angenommen, diese Funktion löscht alle aktuellen Werte von Klassenvariablen. In Book-Objekten soll diese Funktion nicht als Methode verfügbar sein, wohl aber in abgeleiteten Klassen – daher die Definition als `protected`.

Beispiel: Objekt-Handles

Verfolgen wir das Bibliotheksbeispiel weiter und nehmen an, ein Teil des aktuellen Scriptablaufs möchte ein Book-Objekt nutzen, um Daten eines neu erworbenen Buchs in die Datenbank zu schreiben und gleich anschließend zu protokollieren, von welchem angemeldeten Bibliotheksmitarbeiter die Daten zu welchem Zeitpunkt gespeichert wurden. Das könnte im Prinzip so aussehen:

```
# 1. Aktion - Datensatz erzeugen und schreiben:
$book = new Book();
$book->set_title($_POST['title']);
$book->set_sub_title($_POST['sub_title']);
$book->set_author($_POST['author_name']);
# usw.
$book->set_bib_code($_POST['bib_code']);
$book->insert_book_data();

# 2. Aktion - Mitloggen:
$log = new LogEntry();
$log->set_name_value('action', 'insert_book_data');
$log->set_name_value('user', $_SESSION['user_name']);
$log->set_name_value('timestamp', time());
$log->set_name_value('book_title', $book->get_title());
$log->set_name_value('book_sub_title', $book->get_sub_title());
# usw.
$log->set_name_value('bib_code', $book->get_bib_code());
```

Der zweite Teil der Gesamtaktion (das Protokollieren oder Mitloggen) hat ganz andere Aufgaben als der erste Teil (das Erzeugen eines neuen Datensatzes in der Datenbank). Dennoch wird im zweiten Teil noch mit der Objektvariablen aus dem ersten Teil gearbeitet, also mit `$book`, um die darin gespeicherten Daten zu ermitteln und an das LogEntry-Objekt zu übergeben. Nun ist das, so wie im Beispiel notiert, nicht weiter dramatisch. In

einem großen Webprojekt könnte es jedoch aufgrund von vereinbarten Konventionen nötig sein, für jede Aktion eigene Objektvariablen zu verwenden. In PHP 4.x wäre das auf das obige Beispiel bezogen möglich mit einer Anweisung wie:

```
$log_book = $book;
```

Dann würde von `$book` eine Kopie in `$log_book` erzeugt. Beide Variablen und damit beide Objekte haben ansonsten weiter nichts miteinander zu tun und können im weiteren Scriptverlauf ganz unterschiedliche Eigenschaftswerte erzeugen. In PHP 5.x ist dies anders gelöst. Die gleiche Anweisung wie oben bewirkt hier, dass in `$log_book` eine interne Referenz auf `$book` erzeugt wird. Oder noch genauer ausgedrückt: Sowohl `$book` als auch `$log_book` sind nichts weiter als so genannte Handles für ein Objekt, das aus der Klasse Book erzeugt wurde. Handles gibt und gab es immer schon in anderen Zusammenhängen, z.B. die Datei-Handles. Ein Beispiel aus diesem Bereich kann das Wesen von Handles gut erhellen:

```
$fh = fopen("temp.txt", "r");
$fh2 = $fh;
$content = fread($fh2, filesize("temp.txt"));
fclose($fh);
echo $content;
```

In diesem Beispiel ist `$fh` ein File-Handle (daher auch der Variablenname). Glückt das Öffnen der Datei mit `fopen()`, so ist `$fh` ein Zeiger auf die Datei, über den der Zugriff auf die Datei möglich ist. Anschließend wird `$fh` einer zweiten Variablen `$fh2` zugewiesen. Dadurch ist `$fh2` einfach ein zweiter Zeiger auf die gleiche Datei wie `$fh`. Zur Demonstration wird `$fh2` benutzt, um den Inhalt der Datei mit `fread` auszulesen und in einer Variablen zu speichern. Diese wird am Ende ausgegeben. Es gibt also eine Ressource – die Datei – und zwei gleichberechtigte Handles darauf.

Ebenso verhält es sich in PHP 5.x, wenn eine Objektvariable erzeugt wird. Diese ist im Grunde ein Handle für ein Objekt. Wird sie einer weiteren Variablen zugewiesen, so sind beide gleichberechtigte Handles auf das gleiche Objekt. Ändert die erste Objektvariable eine Eigenschaft des Objekts, erhält die zweite Variable beim Auslesen dieser Eigenschaft den von der ersten Variablen geänderten Wert.

Der zweite Teil in unserem Bibliotheksbeispiel könnte also etwa wie folgt notiert werden:

```
# 2. Aktion - Mitloggen:
$log_book = $book;
$log = new LogEntry();
$log->set_name_value('action', 'insert_book_data');
$log->set_name_value('user', $_SESSION['user_name']);
$log->set_name_value('timestamp', time());
$log->set_name_value('book_title', $log_book->get_title());
$log->set_name_value('book_sub_title', $log_book->get_sub_title());
# usw.
$log->set_name_value('bib_code', $log_book->get_bib_code());
```

In PHP 5.x ist dabei sichergestellt, dass in `$log_book` die gleichen aktuellen Objektdaten verfügbar sind wie in `$book`.

Beispiel: Statische Klassenvariablen und Methoden

Statische Klassenvariablen und in Klassen definierte Funktionen sind in einem Objekt dieser Klasse nicht als Eigenschaften und Methoden verfügbar. Der Zugriff auf solche Variablen und Funktionen ist nur über den bereits vorgestellten Operator `::` möglich. Im nachfolgenden Beispiel erhält unsere Klasse `Book` einen objektunabhängigen Zähler, der Auskunft darüber geben kann, wie viele Objekte der Klasse gerade existieren:

```
class Book {
   private $isbn;
   private $title;
   # usw.
   private static $book_objects;
   # usw.

   function __construct() {
      self::$book_objects += 1;
   }

   function __destruct() {
      self::$book_objects -= 1;
}

   static function get_object_count() {
      return self::$book_objects;
   }
}
```

In diesem Beispiel wird die Variable `$book_objects` als nur intern aufrufbare statische Klassenvariable definiert. Das Schlüsselwort `static` muss hinter dem für die Sichtbarkeit stehenden Wert und vor dem Variablennamen notiert werden. Angenommen, es wird nun ein Objekt dieser Klasse erzeugt:

Ferner sind zwei Funktionen namens `__construct()` und `__destruct()` notiert. Dies sind in PHP 5.x reservierte Namen für Konstruktor- und Destruktorfunktionen. In der Konstruktorfunktion, die ja bei jedem neu erzeugten Objekt der Klasse aufgerufen wird, wird die statische Klassenvariable mithilfe von `self::$book_objects` um 1 erhöht. Umgekehrt wird in der Destruktorfunktion der Zähler wieder um 1 verringert. Mehr zu diesen Konstruktor- und Destruktorfunktionen weiter unten bei »Weitere Unterschiede«.

Statische Variablen dürfen wie schon erwähnt nur mit `::` angesprochen werden, nicht über den Pfeiloperator `->`. Innerhalb der Klasse selbst ist das möglich mit dem Schlüsselwort `self`, also mit `self::$Variablenname`. Ist eine statische Klassenvariable als public definiert, kann von außen jederzeit mit `Klassenname::$Variablenname` darauf zugegriffen werden.

In unserem Beispiel gibt es ferner eine Getter-Funktion `get_object_count()`. Diese gibt den aktuellen Zählerstand der statischen Variablen `$book_objects` zurück. Ein Direktzugriff von außen ist ja wegen der private-Deklaration der Variablen ausgeschlossen. So wird sichergestellt, dass die Variable zwar mit der Getter-Funktion gelesen, aber nicht von außen geändert werden kann.

Die Getter-Funktion `get_object_count()` ist selbst eine statische Funktion, weil ihr das Schlüsselwort `static` vorangestellt ist. Das heißt, sie ist in einer Objektvariablen keine verfügbare Methode des Book-Objekts. Auch hier kann nur über den Operator `::`, dafür aber von überall aus auf die Funktion zugegriffen werden, und zwar wie in dieser Beispielanweisung:

```
$book_objects_counter = Book::get_object_count();
```

Beispiel: Finale Klassen und Methoden

Finale Klassen und Methoden dienen vor allem dem Überschreibschutz durch abgeleitete Klassen. Beispiel:

```
class Book {
   private $isbn;
   private $title;
   # usw.

   function get_book_data_by_isbn() {
       # Code ...
   }
   function insert_book_data() {
       # Code ...
   }
   final function get_title() {
       return($this->title);
   }
   # usw.
}
```

In diesem Beispiel wird die Getter-Funktion `get_title()` als `final` definiert. Da sie nur den aktuellen Wert einer Klassenvariablen zurückgibt, ist es wenig sinnvoll, sie in einer von `Book` abgeleiteten Klasse mit einer gleichnamigen Funktion, die anderen Code enthält, zu überschreiben. Daher ist eine Definition solcher Funktionen als final durchaus sinnvoll.

Das Schlüsselwort `final` kann auch vor `class` notiert werden, wie `final class Book { ... }`. In diesem Fall erlaubt die Klasse `Book` überhaupt keine Klassenableitungen mehr mit `extends`.

Beispiel: Abstrakte Klassen

Abstrakte Klassen sind als Vorlagen für abgeleitete Klassen gedacht. Ein Beispiel:

```
abstract class Book {
   protected $isbn;
   protected $title;
   protected $sub_title;
   protected $author;
   protected $editor;
   protected $publisher;
   protected $publishing_year;

   function get_title() {
       return($this->title);
   }
   # usw.
}

class BibBook extends Book {
   private $bib_code;
   private $borrow_count;
   private $borrow_days;
   private $borrowed_flag;
   private $borrowed_since;

   function get_bib_code() {
       return($this->bib_code);
   }
}

$book = new BibBook();
```

Durch Voranstellen des Schlüsselworts `abstract` wird im Beispiel die Klasse `Book` für das Erzeugen von Objekten gesperrt. Die Klasse enthält nur allgemeine Daten und Funktionen zu Büchern. Die Klasse `BibBook` erweitert die Klasse `Book` um Daten und Funktionen einer Bibliotheksverwaltung. Von dieser Klasse kann, wie unten im Beispiel gezeigt, ein Objekt erzeugt werden.

Weitere Unterschiede

PHP 5.x bietet eine Reihe von reservierten Funktionsnamen zur Verwendung innerhalb von Klassen an. Erkennbar sind diese Funktionen daran, dass sie mit zwei Unterstrichen beginnen. Sie werden auch als ***magische Funktionen*** bezeichnet. Die wichtigsten davon sind:

- `__construct()`: Funktionen mit diesem Namen werden als Konstruktorfunktion behandelt, d.h., die Funktion wird automatisch ausgeführt, wenn ein neues Objekt dieser Klasse erzeugt wird. PHP 5.x erlaubt es nach wie vor, Konstruktorfunktionen dadurch zu identifizieren, dass sie den gleichen Namen haben wie die Klasse selbst. Die neue Syntax soll es vermutlich ermöglichen, den Klassennamen zu ändern, ohne

innerhalb der Klasse auch noch den Namen der Konstruktorfunktion ändern zu müssen.

- `__destruct()`: Funktionen mit diesem Namen werden als Destruktorfunktion behandelt. Die Funktion wird automatisch ausgeführt, wenn eine Objektvariable mit dem PHP-Sprachkonstrukt `unset($objvar)` gelöscht wird.
- `__clone()`: PHP 5.x behandelt Objektvariablen wie beschrieben als Objekt-Handles. Um jedoch die Möglichkeit zu haben, Objekte tatsächlich zu kopieren, so wie es in PHP 4.x Standard ist, können Sie diese Funktion aufrufen. Beispiel: wenn in `$book` eine Objektvariable gespeichert ist, dann wird durch `$log_book = $book` in `$log_book` nur ein Object-Handle von `$book` gespeichert, aber kein neues Objekt der Klasse. Durch eine Anweisung wie `$log_book = $book->__clone()` wird jedoch ein neues Objekt in `$log_book` erzeugt, das die aktuellen Objektdaten aus `$book` in Kopie übernimmt. Die Funktion `__clone()` muss innerhalb einer Klasse nicht notiert werden.

Weitere neue Konzepte von PHP 5.x sind:

- `instanceof` ist ein Operator, der überprüft, ob eine Objektvariable ein Objekt einer bestimmten Klasse speichert. Der Operator eignet sich zur Verwendung in Bedingungen. Beispiel:

```
if($book instanceof Book)
   # tu was
```

  Die Bedingung prüft, ob in `$book` ein Objekt der Klasse `Book` gespeichert ist.

- `try`, `catch` und `throw` sind drei zusammengehörige Sprachkonstrukte, die vor allem für Debugging-Zwecke interessant sind. Beispiel:

```
try {
   if(!fopen("logging.txt", "r"))
      throw new exception("Fehler: logging.txt");
}
catch (exception $logging) {
   var_dump($logging);
}
```

  Das Paar `try` und `catch` gehört so ähnlich zusammen wie `if` und `else`. Mit `try` wird eine Bedingung oder der Erfolg einer Anweisung geprüft. Wenn darin etwas fehlschlägt oder falsch ist, wie im Beispiel das fehlschlagende Öffnen einer Datei mit der PHP-Funktion `fopen()`, dann kann dieser Fehler durch `throw` an ein »Ereignisobjekt«, hier `exception` benannt, übergeben werden. Dabei kann auch eine eigene Fehlermeldung übergeben werden. Im catch-Block lässt sich der Fehler dann auswerten. Beliebige Reaktionen sind möglich.

Fazit

PHP 5.x bringt vor allem im Bereich der Objektorientierung Neuerungen, die jedoch keine wirklichen Neuerungen sind, sondern lediglich Angleichungen an OOP-Standards, die von Sprachen wie C++ oder Java gesetzt werden. PHP versucht damit, Programmierer aus

dem Bereich der professionellen Softwareentwicklung anzusprechen. Ob das gelingt und ob sich Entscheider von Großprojekten davon beispielsweise dazu verleiten lassen, auf PHP statt etwa auf JSP zu setzen, muss die Zeit zeigen. Mit den neuen Features versucht PHP jedenfalls einen Spagat zwischen seiner ursprünglichen Konzeption als leicht erlernbare Brot- und Buttersprache und den Anforderungen an eine Programmiersprache für gehobene Ansprüche. Welche dieser Richtungen PHP in weiterer Zukunft eher bevorzugen sollte, ist eines der am meisten und heftigsten diskutierten Themen in der PHP-Welt.

12.3.9 Reguläre Ausdrücke

Reguläre Ausdrücke (*regular expressions*) sind kein spezielles Feature von PHP, sondern eine aus der Unix-Welt stammende, kanonisierte Form, wie Daten danach befragt werden können, ob sie auf bestimmte Suchmuster (*Patterns*) »passen«. Die Syntax regulärer Ausdrücke legt dabei fest, wie solche Suchmuster formuliert werden. Obwohl sie nur aus einem begrenzten Set von Regeln und Sonderzeichen besteht, ist diese Syntax so mächtig, dass sich praktisch beliebige und beliebig komplexe Suchmuster damit formulieren lassen. Nachteil ist die mühsame Lesbarkeit regulärer Ausdrücke, die vielen Einsteigern Probleme beim Verständnis bereitet.

Eine ganze Reihe von PHP-Funktionen erwarten allerdings reguläre Ausdrücke als Parameter. Von daher gehören Kenntnisse in deren Syntax zum Handwerkszeug jedes PHP-Programmierers.

Weiterführende Links:

Regular Expressions:
http://www.regular-expressions.info/
Regex Tutorial:
http://www.delorie.com/gnu/docs/regex/regex_toc.html
Schritt für Schritt zu eigenen regulären Ausdrücken:
http://www.regenechsen.de/regex_allg_de/regex_allg.php

Erstes Beispiel

Ein typischer Anwendungsfall für reguläre Ausdrücke in der Praxis der PHP-Programmierung ist das Überprüfen, ob ein übergebener Formularfeldinhalt einem bestimmten, erwarteten Muster entspricht. Angenommen, es wurde in `$_POST['birthday']` ein Geburtsdatum übergeben. Die Anwendung wünscht, dass dieses im Format dd.mm.jjjj vorliegt. Im PHP-Script könnte Folgendes stehen:

```
if(preg_match("/^\d\d\.\d\d\.\d\d\d\d$/", $_POST['birthday']))
  echo "Eingabe korrekt";
else
  echo "Eingabe verkehrt";
```

Die PHP-Funktion `preg_match()` durchsucht eine Zeichenkette nach einem regulären Ausdruck. Im ersten Parameter wird der gewünschte reguläre Ausdruck übergeben und im zweiten Parameter die zu durchsuchende Zeichenkette, in unserem Beispiel also

`$_POST['birthday']`. Die Funktion kann noch weitere Parameter übergeben bekommen, doch das ist an dieser Stelle nicht von Bedeutung. So wie hier aufgerufen, gibt `preg_match()` den Wert `1` (`true`) zurück, wenn das Suchmuster gefunden werden konnte, und `0` (`false`), wenn es nicht gefunden wurde. Daher eignet sich `preg_match()` hervorragend, um es als Bedingung in eine if-Abfrage einzubauen, so wie im Beispiel.

Reguläre Ausdrücke werden in Begrenzerzeichen eingeschlossen. Per Default sind das einfache Schrägstriche. Deshalb sind beim übergebenen ersten Parameter das erste und das letzte Zeichen ein `/`. Alles, was dazwischen steht, wird als Suchmuster interpretiert.

Unser Beispielausdruck verlangt, dass die zu durchsuchende Zeichenkette gleich am Anfang (`^`) eine Ziffer (`\d`) enthält, gefolgt von einer weiteren Ziffer. Dann muss ein Punkt (`\.`) folgen. Dann müssen wieder zwei Ziffern folgen, dann wieder ein Punkt und zuletzt, vor dem Ende der Zeichenkette (`$`) noch mal vier Ziffern.

Verwirrend sind bei regulären Ausdrücken vor allem die vielen Backslash-Zeichen (`\`). Diese dienen dazu, nachfolgende Zeichen mit Sonderbedeutung zu versehen, oder um Zeichen davor zu bewahren, als solche mit Sonderbedeutung interpretiert zu werden. Im Fall von `\d` weist der Backslash an, dass nicht nach dem Buchstaben »d« gesucht werden soll, sondern nach einer Ziffer zwischen 0 und 9, wofür es die abgekürzte Schreibweise `\d` gibt. Im Fall von `\.` ist das Gegenteil davon der Fall. Normalerweise hat der Punkt bei regulären Ausdrücken besondere Bedeutung (er steht für ein beliebiges Zeichen). Wenn aber nach dem Punkt als Zeichen gesucht werden soll, muss dies durch `\.` kenntlich gemacht werden.

Unser Beispiel ist sehr restriktiv. Es bewirkt, dass nur Formulareingaben wie 23.11.1972 oder 02.07.1969 als korrekt betrachtet werden. Nicht einmal Eingaben wie 2.7.1969 wären zulässig. Um dem Anwender wenigstens die Freiheit zu lassen, einstellige Tages- und Monatsangaben benutzen zu können, können wir den regulären Ausdruck abwandeln:

```
if(preg_match("/^\d{1,2}.\d{1,2}\.\d\d\d\d$/", $_POST['birthday']))
  echo "Eingabe korrekt";
else
  echo "Eingabe verkehrt";
```

Nun ist bei den Teilen für Monatstag und Monatsname nur noch einmal `\d` notiert. Dahinter steht jeweils in geschweiften Klammern `{1,2}`. Das bedeutet in der Syntax regulärer Ausdrücke: Voranstehendes Zeichen darf ein- bis zweimal in Folge vorkommen. Auf `\d` angewendet, bedeutet es: An dieser Stelle müssen ein oder zwei Ziffern vorkommen.

Diese kleine Variante lässt schon erahnen, dass reguläre Ausdrücke sehr fein justierbar sind.

PHP-Funktionen mit regulären Ausdrücken

Nachfolgende Tabelle listet PHP-Funktionen auf, in denen reguläre Ausdrücke als Parameter erwartet werden:

Funktion	Bedeutung
`preg_grep()`	Durchsucht einen Array und gibt einen neuen Array aus allen Array-Elementen zurück, die auf ein Suchmuster passen. Sehr praktisch beispielsweise, um in einer mit `file(...)` zeilenweise eingelesenen Datei alle Zeilen zu extrahieren, die einem Suchmuster genügen.
`preg_match()`	Durchsucht eine Zeichenkette nach einem Suchmuster und gibt 1 zurück, wenn das Suchmuster gefunden wurde, und 0, wenn es nicht gefunden wurde.
`preg_match_all()`	Durchsucht eine Zeichenkette nach einem Suchmuster und gibt die Anzahl gefundener Stellen zurück, auf die das Suchmuster passte. Dies kann auch 0 sein.
`preg_quote()`	Maskiert eine übergebene Zeichenkette so, dass sie zur Verwendung als regulärer Ausdruck taugt. Praktisch um eine dynamisch ermittelte Zeichenkette von PHP zur Verwendung in einem regulären Ausdruck vorbereiten zu lassen.
`preg_replace()`	Durchsucht eine Zeichenkette oder einen Array nach einem Suchmuster und ersetzt jedes passende Suchmuster durch eine anzugebende Ersetzungszeichenkette. Letztere darf dabei auch Referenzen auf Klammern des Suchmusters enthalten. Gibt die geänderte Zeichenkette oder den geänderten Array zurück.
`preg_replace_callback()`	Im Unterschied zu `preg_replace()` kann hierbei anstelle einer Ersetzungszeichenkette der Name einer Funktion übergeben werden. Der Funktion wird ein Array aller Stellen übergeben, auf die das Suchmuster passt. Die Funktion sollte eine Ersetzungszeichenkette zurückgeben.
`preg_split()`	Zerlegt eine Zeichenkette an Hand einer Trennzeichenregel in einen Array. Im Gegensatz zu den Funktionen `split()` oder `explode()` kann anstelle einer fixen Trennzeichenfolge ein regulärer Ausdruck übergeben werden.

Tabelle 12.10: PHP-Funktionen mit regulären Ausdrücken

Erkennbar ist, dass reguläre Ausdrücke bei vielen Alltagsaufgaben Anwendung finden, also beim Überprüfen, ob Daten ein korrektes Format haben, oder beim Ersetzen von Inhalten durch andere.

Reguläre Ausdrücke für einzelne Zeichen

Reguläre Ausdrücke sind zeichenweise zu lesen und werden auch zeichenweise ausgewertet. Ein einzelnes Zeichen ist also die Grundeinheit regulärer Ausdrücke und wird auch als *Atom* bezeichnet. Das klingt trivial, doch ein Zeichen kann durchaus durch eine ganze Reihe von Zeichen referenziert werden.

Nachfolgende Tabelle listet typische Referenzierungsmöglichkeiten für einzelne Zeichen auf. Die beispielhaften regulären Ausdrücke suchen der Einfachheit halber nur nach dem jeweils einen Zeichen. Jedes dieser so notierten Zeichen kann aber natürlich auch in einer Zeichenkette vorkommen.

Regulärer Ausdruck	Bedeutung
`/a/`	Passt, wenn mindestens ein `a` gefunden wird. Zeichen wie normale Buchstaben werden ganz normal notiert.
`/[aA]/`	Passt, wenn mindestens ein `a` oder ein `A` gefunden wird. Eine Auswahl aus mehreren möglichen Zeichen wird durch eckige Klammern markiert.
`/[A-Z]/`	Passt, wenn mindestens ein Großbuchstabe zwischen A und Z gefunden wird. Zeichenbereiche werden durch eckige Klammern markiert. Erstes und letztes Zeichen im Bereich werden durch – verbunden.
`/[a-zA-Z]/`	Passt, wenn mindestens ein Großbuchstabe oder Kleinbuchstabe gefunden wird. Mehrere Zeichenbereiche können innerhalb einer eckigen Klammer ebenfalls notiert werden.
`/[0-7]/`	Passt, wenn mindestens eine Ziffer zwischen 0 und 7 vorkommt.
`/\d/`	Passt, wenn mindestens eine Ziffer (0 bis 9) vorkommt.
`/\D/`	Passt, wenn mindestens ein Zeichen keine Ziffer (0 bis 9) ist.
`/\[/`	Passt, wenn mindestens ein Zeichen eine eckige Klammer ist. Da eckige Klammern Zeichenauswahl oder Zeichenbereiche bestimmen, also Sonderbedeutung in regulären Ausdrücken haben, müssen sie mit Backslash (\) maskiert werden.
`/\w/`	Passt, wenn mindestens ein Zeichen ein Buchstabe, eine Ziffer oder ein Unterstrich ist.
`/\W/`	Passt, wenn mindestens ein Zeichen weder Buchstabe noch Ziffer noch Unterstrich ist.
`/\r/`	Passt, wenn mindestens ein Steuerzeichen für Wagenrücklauf gefunden wird.
`/\n/`	Passt, wenn mindestens ein Steuerzeichen für Zeilenvorschub gefunden wird.
`/\t/`	Passt, wenn mindestens ein Steuerzeichen für Tabulator gefunden wird.
`/\f/`	Passt, wenn mindestens ein Steuerzeichen für Seitenvorschub gefunden wird.
`/\s/`	Passt, wenn mindestens eines der Leerraumzeichen \r, \n oder \t gefunden wird.
`/\S/`	Passt, wenn mindestens ein Zeichen kein Leerraumzeichen \r, \n oder \t ist.
`/[^a-zA-Z]/`	Passt, wenn mindestens ein Zeichen kein Buchstabe ist. Mit ^ innerhalb einer eckigen Klammer wird ein dahinter notiertes Zeichen, eine Auswahl mehrerer Zeichen oder ein Zeichenbereich verneint.

Tabelle 12.11: Notation einzelner Zeichen in regulären Ausdrücken

Reguläre Ausdrücke für Zeichenketten

In den meisten Fällen wird ein Suchmuster natürlich nicht aus einzelnen Zeichen, sondern aus einer ganzen Zeichenkette gebildet. Folgende Möglichkeiten bestehen dabei:

- Sie können nach einer bestimmten Zeichenkette suchen.
- Sie können mithilfe von *Platzhalterzeichen* suchen, wobei es Platzhalterzeichen für einzelne Zeichen wie auch für ganze Teilzeichenketten gibt.
- Sie können die Position bestimmen, ab der gesucht werden soll. Dabei ist es auch möglich zu bestimmen, dass ein Suchmuster die gesamte Zeichenkette abdecken soll.

Nachfolgende Tabelle enthält Beispiele für solche Suchmuster:

Regulärer Ausdruck	Bedeutung
`/aus/`	Passt, wenn die Zeichenkette `aus` gefunden wird.
`/^aus/`	Passt, wenn die Zeichenkette `aus` ***am Anfang*** der zu durchsuchenden Zeichenkette gefunden wird.
`/aus$/`	Passt, wenn die Zeichenkette `aus` ***am Ende*** der zu durchsuchenden Zeichenkette gefunden wird.
`/^aus$/`	Passt, wenn die zu durchsuchende Zeichenkette ***nichts anderes als die Zeichenkette*** `aus` enthält.
`/aus?/`	Passt, wenn `aus` gefunden wird, aber auch, wenn `au` oder `auf` gefunden werden. Das `?` bedeutet: Das Zeichen davor kann vorkommen, muss es aber nicht, und außerdem kann auch ein anderes Zeichen an seiner Stelle vorkommen.
`/a./`	Passt, wenn `ab` gefunden wird, aber auch, wenn `an` oder `a7` gefunden wird. Der Punkt `.` bedeutet: ein beliebiges Zeichen an dieser Stelle (außer `\n`).
`/a+/`	Passt, wenn `a` gefunden wird, aber auch, wenn `aa` oder `aaaaaa` gefunden wird. Das `+` bedeutet: beliebig viele Zeichen des Zeichens davor in Folge, aber mindestens das eine Zeichen.
`/a*/`	Passt, wenn `a` gefunden wird, aber auch, wenn `aa` oder `aaaaaa` oder einfach `b` gefunden wird (aber nicht `bb`). Das `*` bedeutet: beliebig viele Zeichen des Zeichens davor in Folge oder genau ein anderes Zeichen an seiner Stelle.
`/aus.+/`	Passt, wenn `aus` gefunden wird, gefolgt von einer beliebigen Zeichenfolge mit mindestens einem Zeichen.
`/aus.*/`	Passt, wenn `aus` gefunden wird, gefolgt von einer beliebigen Zeichenfolge oder auch von keiner Zeichenfolge.
`/x{10,20}/`	Passt, wenn 10 bis 20 `x` in Folge gefunden werden.
`/x{10,}/`	Passt, wenn 10 oder mehr `x` in Folge gefunden werden.
`/x.{2}y/`	Passt, wenn `x`, gefolgt von zwei beliebigen Zeichen außer `\n`, gefolgt von `y`, gefunden wird.
`/Hans\b/`	Passt, wenn `Hans` gefunden wird, aber ***nicht, wenn Hansel*** gefunden wird. `\b` markiert eine Wortgrenze.
`/\baus/`	Passt, wenn `aus` oder `ausnahmslos` gefunden wird, aber ***nicht, wenn Haus*** gefunden wird.
`/\baus\b/`	Passt, wenn `aus` als alleinstehendes Wort gefunden wird.
`/\baus\B/`	Passt, wenn `ausnahmslos` gefunden wird, aber ***nicht, wenn aus*** gefunden wird. `\B` bedeutet: keine Wortgrenze an dieser Stelle.

Tabelle 12.12: Beispiele für reguläre Ausdrücke mit Zeichenketten

Zeichenmaskierung in regulären Ausdrücken

Diverse Zeichen wie etwa der Punkt, die eckigen Klammern, Plus- und Sternzeichen haben wie gezeigt in regulären Ausdrücken besondere Bedeutung. Um in einem regulären Ausdruck ein solches Zeichen als Zeichen zu suchen, muss es maskiert werden. Zur Mas-

kierung dient einheitlich der Backslash (\). Natürlich muss auch der Backslash selbst maskiert werden, wenn nach ihm gesucht werden soll:

Zeichen	Maskierung	Beispiel
/	\/	/http:\/\//
.	\.	/http:\/\/www\.example\.org\//
+	\+	/\d+\+\d+/
*	*	/\d+*\d+/
?	\?	/Frage oder nicht\?/
^	\^	/ein \^ über dem Kopf/
$	\$	/Preis (US-Dollar): \d*\$/
|	|	/a|b/
( )	\(\)	/function .+\(.+\)/
[ ]	\[\]	/array\[0\]/
{ }	\{ \}	/\{.+\}/
\	\\	/c:\\/

Tabelle 12.13: Maskierungspflichtige Zeichen, wenn nach diesen Zeichen gesucht werden soll

Das Gierigkeitsprinzip

Gierige Suchmuster können Sie durch ein Fragezeichen (?) an der richtigen Stelle ausbremsen.

Angenommen, Sie haben folgenden PHP-Code:

```
$uri = "http://www.google.de/
search?hl=de&q=regul%C3%A4re+Ausdr%C3%BCcke&btnG=Google-Suche&meta=";
preg_match("/^.+\?.+=[^&].+/", $uri, $matches);
echo "{$matches[0]}";
```

Mit der PHP-Funktion `preg_match()` wird hier die Variable `$uri` durchsucht, die eine HTTP-Adresse mit GET-String speichert, wobei der GET-String aus mehreren Parametern besteht.

Gesucht wird nach folgendem Muster: ein Fragezeichen (`\?`), gefolgt von einer beliebigen Teilzeichenkette mit mindestens einem Zeichen (`.+`), gefolgt von einem Gleichheitszeichen (`=`), gefolgt wieder von einer Teilzeichenkette mit mindestens einem Zeichen und ansonsten beliebig vielen Zeichen, die kein &-Zeichen sind (`[^&]+`).

Erwartet werden könnte, dass damit das Fragezeichen zwischen URI-Dateipfad und GET-String und genau der erste GET-Parameter und sein Wert gefunden werden, also:

```
?hl=de
```

Tatsächlich gefunden wird aber Folgendes:

```
?hl=de&q=regul%C3%A4re+Ausdr%C3%BCcke&btnG=Google-Suche&meta=
```

Es wird also nicht das erste auftretende & als »Stoppzeichen« genommen, sondern das letzte innerhalb der Gesamtzeichenkette.

Dieses Verhalten von regulären Ausdrücken wird als *gierig* bezeichnet. Gefunden wird immer so viel wie möglich. Doch nicht immer ist das gewünscht. Damit das gefunden wird, was oben erwartet wurde, muss der reguläre Ausdruck wie folgt erweitert werden:

```
/\?.+?=[^&].+?/
```

Gefunden wird jetzt:

```
?hl=de
```

Grund für die Genügsamkeit sind die Fragezeichen hinter den beiden Pluszeichen. Durch ein Fragezeichen hinter einer der Kombinationen `.*` oder `.+` wird das betreffende Suchmuster genügsam und reicht nur bis zum nächsten gefundenen Stoppzeichen. Allerdings mussten wir aus diesem Grund in unserem Beispiel `[^&]+` zu `[^&].+` erweitern (also den Punkt einfügen), sonst wäre die Rechnung mit der Genügsamkeit nicht aufgegangen.

Alternative Teilausdrücke

Alternative Teilausdrücke können Sie durch den Senkrechtstrich (`|`) festlegen. Ein Beispiel:

```
$str = "Herr Walter, Frau Dorn, Herr Abel";
preg_match_all("/Herr|Frau/", $str, $matches);
echo count($matches[0]);
```

In diesem Beispiel werden in `$str` alle Vorkommnisse von `Herr` oder `Frau` gesucht. Der reguläre Ausdruck sucht nach beiden Wörtern, getrennt durch ein `|`. Auf die gleiche Weise könnten auch noch weitere Alternativen angehängt werden.

An die dafür geeignete Funktion `preg_match_all()` wird als dritter Parameter eine Variable namens `$matches` übergeben. Daraus macht `preg_match_all()` einen verschachtelten Array. Sein erstes Element ist ein weiterer Array, in dem alle gefundenen Stellen in je einem Element gespeichert werden. Durch Ausgabe von `count($matches[0])` wird die Länge dieses inneren Arrays ermittelt. Das ausgegebene Ergebnis teilt damit die Anzahl der Fundstellen mit. Im Beispiel beträgt sie 3.

Teilausdrücke merken und referenzieren

Vor allem beim Suchen und Ersetzen kommt es oft vor, dass man Teile des Suchmusters als Variablen in die Ersetzungszeichenkette übernehmen will. Dies wird in regulären Ausdrücken dadurch möglich, dass gewünschte Teile des Suchmusters, aus denen später Variablen werden sollen, in runde Klammern `(` und `)` eingeschlossen werden.

Ein Beispiel mit der PHP-Funktion `preg_replace()` soll dies demonstrieren. In Foren, Gästebüchern oder ähnlichen Plattformen, wo Anwender etwas beitragen können, möchten Anbieter ihren Anwendern das Formatieren von Text ermöglichen. Allerdings wollen sie vermeiden, dass Anwender direkt beliebigen HTML-Code eingeben, weil dadurch zu viel

Missbrauch möglich wäre (fiese JavaScript-Passagen, automatische Umleitungen, Referenzen auf Grafiken auf anderen Servern, hinter denen sich ein Script verbirgt, und allerlei mehr). Für solche Anbieter hat sich mittlerweile mit dem so genannten BBCode ein Quasi-Standard einer primitiven Pseudo-Auszeichnungssprache durchgesetzt. BBCode wurde von einem Bulletin-Board-System eingeführt und hat sich vielfach bewährt. Ein Beispieltext:

```
[b]Hallo[/b] und [u]guten Morgen[/u]
Ich bin noch etwas [i]schief drauf![/i]
[color=red]Dafür steht mir der Sinn nach Rot[/color]
```

Wurde ein solcher Inhalt in einem mehrzeiligen Textfeld eines Formulars eingegeben, könnte er dem PHP in einer Variable wie `$_POST['message_text']` zur Verfügung stehen. Eine PHP-Funktion zur Umsetzung dieses BBCodes in HTML-Markup könnte folgendermaßen aussehen:

```
function BB_to_HTML($str) {
   $str = preg_replace("/\[b\](.*?)\[\/b\]/","<b>$1</b>",$str);
   $str = preg_replace("/\[u\](.*?)\[\/u\]/","<u>$1</u>",$str);
   $str = preg_replace("/\[i\](.*?)\[\/i\]/","<i>$1</i>",$str);
   $str = preg_replace("/\[color=(.*?)\](.*?)\[\/color\]/",
                  "<span style=\"color:$1\">$2</span>",$str);
   return($str);
}
```

Die Funktion `BB_to_HTML()` sucht eine ihr übergebene Zeichenkette nach BB-Code ab und wandelt ihn in HTML-Markup um. Beispielsweise wird `[b]...[/b]` in `<b>...</b>` umgewandelt. Die entsprechende Code-Zeile lautet:

```
$str = preg_replace("/\[b\](.*?)\[\/b\]/","<b>$1</b>",$str);
```

Die PHP-Funktion `preg_replace()` erwartet als ersten Parameter einen regulären Ausdruck für das, wonach gesucht werden soll. Der zweite Parameter bestimmt, durch was Suchtreffer ersetzt werden sollen. Der dritte Parameter übergibt die Zeichenkette, in der gesucht und ersetzt werden soll.

Gesucht wird nach `[b]irgendwas[/b]`, ausgedrückt durch `/\[b\].*?\[\/b\]/`. Das Fragezeichen verhindert Gierigkeit für den Fall, dass mehrere fett ausgezeichnete Passagen im Text vorkommen. Doch eines fehlt noch: Den Inhalt zwischen `[b]` und `[/b]` wollen wir uns merken, um ihn beim Ersetzen durch `<b>irgendwas</b>` wieder einzusetzen. Dazu dienen die runden Klammern: `/\[b\](.*?)\[\/b\]/`. Der Inhalt zwischen den Auszeichnungen wird damit »vermerkt«. Ansonsten haben die runden Klammern keinen Einfluss auf die Syntax des regulären Ausdrucks.

Im zweiten Parameter von `preg_replace()` dürfen Referenzen auf gemerkte Inhalte notiert werden. Den ersten mit `(...)` gemerkten Inhalt eines regulären Ausdrucks können Sie mit der Variablen `$1` referenzieren, den zweiten mit `$2` usw. Im Beispiel lautet die Ersetzungszeichenkette daher `<b>$1</b>`.

Im letzten preg_replace()-Aufruf der Funktion wird mit zwei Merkvariablen gearbeitet:

```
$str = preg_replace("/\[color=(.*?)\](.*?)\[\/color\]/",
                    "<span style=\"color:$1\">$2</span>",$str);
```

In diesem Fall muss sich das Script sowohl den Farbwert als auch den Inhalt merken. Das Ganze wird dann in ein CSS-formatiertes span-Element übertragen. Diesmal wird der Elementinhalt mit `$2` referenziert, weil `$1` bereits den Farbwert referenziert.

Modifikatoren für reguläre Ausdrücke

Einige generelle Optionen für die Bewertung eines regulären Ausdrucks werden außerhalb des Ausdrucks selbst notiert, und zwar hinter dem abschließenden `/`. Im Fachjargon werden sie als ***Modifikatoren*** bezeichnet. Ein Beispiel:

```
if(preg_match("/stefan/i", $_POST['name']))
   # tu was
```

Durch das i hinter dem regulären Ausdruck, welches so viel bedeutet wie ignore case, ist es egal, ob in der zu durchsuchenden Zeichenkette `Stefan`, `stefan` oder `STEFAN` vorkommt. Das `i` verhindert einfach die Beachtung von Groß- und Kleinschreibung.

Weitere Modifikatoren sind:

- `m`: Es wird über mehrere Zeilen hinweg gesucht.
- `s`: Auch das Zeilenumbruchzeichen wird mit dem Punkt (.) erfasst.
- `x`: Leerraumzeichen im Suchausdruck sollen ignoriert werden.

12.4 PHP-Funktionsüberblick

Wie eingangs schon erwähnt, besteht eine besondere Stärke von PHP in der Vielzahl an vordefinierten Funktionen. Je nach Version und Verfügbarkeit sind es auf jeden Fall mehrere hundert. Sie alle systematisch zu behandeln, würde den Rahmen dieses Buches sprengen. Im Zweifelsfall und zur Suche ist die Standard-Funktionsreferenz von PHP, die auch in deutscher Sprache unter *http://de.php.net/manual/de/funcref.php* existiert, immer noch die erste Wahl. Wir werden im Rahmen dieses Abschnitts ein Repertoire an Funktionen vorstellen, das auf jeden Fall zum souveränen Umgang mit PHP befähigt. Weitere, MySQL-spezifische Funktionen werden wir dann noch im MySQL-Kapitel kennen lernen.

12.4.1 Daten lesen und auswerten

Beim Einsatz als Webprogrammiersprache fallen in PHP folgende typischen Aufgaben beim Lesen und Auswerten von Daten an:

- Daten, die mittels `POST` oder `GET` übergeben wurden und dem Script als Input zur Verarbeitung dienen, müssen ausgelesen und überprüft werden.

- Das Script muss gegebenenfalls Dateien auf dem Server öffnen und Inhalte daraus lesen können, z.B. weil Dateien durchsucht werden sollen oder weil Inhalte, die in Dateien stehen, in einer auszugebenden Webseite angezeigt werden sollen.

Webseiten-Input via GET und POST auslesen und auswerten

Daten, die via GET-String oder in einem POST-Request an den Webserver übertragen wurden, stellt PHP einem Script bequem in den superglobalen Arrays `$_GET` und `$_POST` zur Verfügung. Insofern sind möglicherweise auch die Funktionen für Arrays (siehe ***Abschnitt 12.4.3***) interessant.

Angenommen, in einem HTML-Formular ist folgendes Feld notiert:

```
<input type="text" name="Vorname" class="text" style="width:400px">
```

Im einleitenden `<form>`-Tag sei `method="post"` notiert. Dann kann ein PHP-Script, das durch Absenden des Formulars aufgerufen wird, über `$_POST['Vorname']` den im Feld eingegebenen Wert auslesen.

Oder angenommen, ein Script wurde über folgenden URI aufgerufen:

```
http://localhost/index.php?show=72&user=Miki089&view=normal
```

Dann kann das so aufgerufene Script beispielsweise über `$_GET['show']` den Wert des Parameters `show` (im Beispiel 72) ermitteln.

Nehmen wir nun jedoch folgende HTML-Passage an:

```
Ich habe Interesse an:
<input type="checkbox" name="Interesse" value="Sport"> Sport
<input type="checkbox" name="Interesse" value="Musik"> Musik
<input type="checkbox" name="Interesse" value="Essen"> Essen
<input type="checkbox" name="Interesse" value="Kino"> Kino
```

Es ist durchaus zweckmäßig, diesen Checkboxen den gleichen Namen zu verpassen, da sie ja eine logische Gruppe bilden. Das gleiche Problem würde auch bei select-Listen mit dem Attribut `multiple` für Mehrfachauswahl auftreten. Angenommen, der Anwender kreuzt »Musik« und »Kino« an. In `$_POST['Interesse']` bekommt das Script jedoch nur den Wert `Kino` übergeben. Mit einem Trick in HTML können Sie dieses Problem beheben:

```
Ich habe Interesse an:<br>
<input type="checkbox" name="Interesse[]" value="Sport"> Sport
<input type="checkbox" name="Interesse[]" value="Musik"> Musik >
<input type="checkbox" name="Interesse[]" value="Essen"> Essen
<input type="checkbox" name="Interesse[]" value="Kino"> Kino
```

Notieren Sie hinter dem Namen der Felder jeweils ein paar eckige Klammern. Der HTML-Parser akzeptiert das (es ist sogar valides HTML) und PHP wird damit signalisiert, dass es mehrere Felder mit diesem Namen als Array speichern soll. Nun würde das Script in `$_POST['Interesse']` einen Array mit den angekreuzten Werten vorfinden.

Oftmals weiß ein Script jedoch nicht, ob Daten eines bestimmten GET-Parameters oder eines Formularfelds übergeben wurden. Dazu kann es die PHP-Funktion `isset()` aufrufen. Ein typisches Aufrufbeispiel:

```
if(isset($_GET['user']))
  echo "<p>Username angegeben</p>";
else
  echo "<p>Kein Username angegeben</p>";
```

Funktion:	`isset()`
Aufgabe:	Prüft, ob eine Variable existiert.
Parameter:	1. Variablenname.
Rückgabewert:	`true`, wenn Variable existiert, `false`, wenn nicht.

Tabelle 12.14: Steckbrief Funktion »isset()«

Da sich ja, wie im Beispiel »Interesse an...« beschrieben, hinter einem in `$_POST` oder `$_GET` gespeicherten Array-Element selbst ein Array verbergen kann, ist es möglicherweise sinnvoll, zunächst einmal abzufragen, ob eine solche Variable ein Array ist. Beispiel:

```
if(is_array($_POST['Interesse']))
  echo "<p>ist ein Array</p>";
else
  echo "<p>ist kein Array</p>";
```

Funktion:	`is_array()`
Aufgabe:	Prüft, ob eine Variable ein Array ist.
Parameter:	1. Variablenname.
Rückgabewert:	`true`, wenn Variable ein Array ist, `false`, wenn nicht.

Tabelle 12.15: Steckbrief Funktion »is_array()«

Wenn Formulardaten an das Script übergeben wurden, lässt sich mit einer Funktion wie `isset()` oder `is_array()` jedoch ***nicht*** ermitteln, ob Felder leer sind. Von `isset()` werden grundsätzlich alle übertragenen Formularfelder als existent gemeldet, weil sie im Array `$_POST` oder `$_GET` existieren. Um zu überprüfen, ob ein Feld keinen Inhalt hat, können Sie stattdessen die Funktion `empty()` verwenden. Typisches Beispiel:

```
if(empty($_POST['Vorname']))
  # Fehlerbehandlung, falls Vorname angegeben werden muss
else
  # normale weitere Datenverarbeitung
```

Funktion:	`empty()`
Aufgabe:	Prüft, ob eine Variable leer ist. Das kann eine nicht initialisierte Variable oder eine Variable mit leerer Zeichenkette sein.
Parameter:	1. Variablenname.
Rückgabewert:	`true`, wenn Variable ein Array ist, `false`, wenn nicht.

Tabelle 12.16: Steckbrief Funktion »empty()«

Mittels `GET` oder `POST` übergebene Einzelwerte stehen stets als Zeichenketten zur Verfügung, auch wenn numerische Werte eingegeben wurden. Um einen erwarteten numerischen Wert in eine numerische Variable zu überführen, bietet sich eine Typenkonvertierung an. Dazu stehen in PHP diverse Einzelfunktionen bereit. Es genügt jedoch, sich eine zu merken: die Funktion `settype()`. Typisches Aufrufbeispiel:

```
settype($_POST['Alter'], "integer");
```

Damit wird der Formularwert, der in `$_POST['Alter']` gespeichert ist, in eine Ganzzahl (Integer-Wert) umgewandelt.

Funktion:	`settype()`
Aufgabe:	Führt eine Typenkonvertierung für eine Variable durch.
Parameter:	1. Variablenname. 2. Gewünschter Datentyp: `integer`, `float` oder `double`, `string`, `array`, `object`, `boolean`, `null`.
Rückgabewert:	`true`, wenn die Konvertierung erfolgreich war, `false`, wenn nicht.

Tabelle 12.17: Steckbrief Funktion »settype()«

Wenn nun aber ein Anwender in einem Feld zur Altersangabe Unsinn wie `abc` eingegeben hat, dann bewirkt `settype()`, dass in der entsprechenden Variablen hinterher der Wert 0 steht. Falls Sie vorher schon überprüfen wollen, ob ein Wert aus einem Formularfeld überhaupt numerisch ist, müssen Sie das Feld mithilfe von regulären Ausdrücken untersuchen. Die Funktion `preg_match()` ist hierzu das richtige Werkzeug. Ein Beispiel zur Überprüfung auf Ganzzahlwert:

```
if(preg_match("/^\d+$/", $_POST['Alter']))
  # Fehlerbehandlung, da Alter keine positive Ganzzahl
else
  # normale weitere Datenverarbeitung
```

Funktion:	`preg_match()`
Aufgabe:	Sucht in einer Zeichenkette nach einem regulären Ausdruck.

Tabelle 12.18: Steckbrief Funktion »preg_match()«

Parameter:	1. Regulärer Ausdruck. 2. Zu durchsuchende Zeichenkette. 3. (optional) Arrayvariable für ersten Suchtreffer und gemerkte geklammerte Inhalte. 4. (optional) Flag PREG_OFFSET_CAPTURE, das auch die Position des ersten Treffers in der Zeichenkette in der Arrayvariablen mit speichert. 5. (optional) Position, ab der innerhalb der Zeichenkette mit der Suche begonnen werden soll. Default ist 0 für »ab erstem Zeichen«.
Rückgabewert:	`1`, wenn die Suche erfolgreich war, `0`, wenn nicht.

Tabelle 12.18: Steckbrief Funktion »preg_match()« (Fortsetzung)

Andere interessante reguläre Ausdrücke zur Formatüberprüfung sind:

- `"/^[-+]?\d+$/":` Prüfung auf Ganzzahl mit eventuellem Vorzeichen.
- `"/^[-+]?[0-9]*\.[0-9]+|[0-9]+$/":` Prüfung auf Float-Zahl. Vorzeichen können möglich sein. Dezimalzeichen muss ein Punkt sein, Ganzzahlen werden jedoch auch akzeptiert.
- `"/^[-+]?[0-9]*\.?[0-9]+[eE][-+]?[0-9]+?$/":` Prüfung auf Zahl in Exponentialschreibweise. Vorzeichen können möglich sein. Das Exponentialzeichen kann »e« oder »E« sein. Dezimalzeichen ist der Punkt. Ganzzahlen werden jedoch auch akzeptiert.
- `"/^\d{1,2}.\d{1,2}\.\d\d\d\d$/":` Prüfung auf Datumseingabe in der Form `TT.MM.JJJJ`. Monatstage und Monate dürfen auch einstellig sein.

Bei Zeichenketten aus textarea-Feldern kann auch die Länge eine Rolle spielen. Diese lässt sich mit `strlen()` ermitteln. Ein Beispiel:

```
if(strlen($_POST['Mssage']) > 10000)
  # Fehlerbehandlung, da die Nachricht mehr als 10000 Byte hat
else
  # normale weitere Datenverarbeitung
```

Funktion:	`strlen()`
Aufgabe:	Ermittelt die Länge einer Zeichnkette.
Parameter:	1. Die Zeichenkette.
Rückgabewert:	Zurückgegeben wird die Anzahl Zeichen als Zahl.

Tabelle 12.19: Steckbrief Funktion »strlen()«

Machen Sie sich die Mühe, mittels `GET` oder `POST` übertragene Daten genauestens zu analysieren. Das gilt ganz besonders, wenn aufgrund solcher Daten sensible Vorgänge angestoßen werden, wie Bezahlvorgänge, Ausführen von Scripts oder Kommandos auf dem Server-Rechner usw.

Dateien öffnen und aus Dateien lesen

Wenn Sie mit PHP auf dem Server-Rechner Dateien lesen oder Daten in Dateien schreiben wollen, können Sie in PHP zwischen bequemen Komfortfunktionen und klassischer »Zu-Fuß-Programmierung« wählen. Dabei gilt die Regel: Alles, was sich durch die Komfortfunktionen erledigen lässt, sollte auch darüber erledigt werden, da es geprüfte, bewährte Funktionen sind. Nur wenn Sie gute Gründe haben, sollten Sie selber mit Dateizeigern und ähnlichem Werkzeug hantieren.

Bevor Sie eine Datei einlesen, empfiehlt es sich zu überprüfen, ob diese auch wirklich existiert. So können Sie Laufzeitfehler sauber abfangen. Dazu gibt es die Funktion `file_exists()`. Ein Beispiel:

```
if(file_exists("/var/www/data/contents/c0253.txt"))
  # Datei einlesen
else
  # Fehlerbehandlung: Datei nicht gefunden
```

Funktion:	`file_exists()`
Aufgabe:	Ermittelt, ob eine Datei oder ein Verzeichnis vorhanden ist oder nicht. Nur anwendbar auf Dateien/Verzeichnisse des lokalen Dateisystems, nicht auf URIs.
Parameter:	1. Die Datei, wenn nötig mit absolutem Pfadnamen oder Pfadnamen aus Sicht des PHP-Scripts.
Rückgabewert:	`true`, wenn die Datei existiert, `false`, wenn nicht.

Tabelle 12.20: Steckbrief Funktion »file_exists()«

Auf Linux/Unix-Systemen ist es mitunter zusätzlich noch von Bedeutung, ob eine Datei für den Benutzer überhaupt lesbar ist. Wenn PHP als Apache-Modul konfiguriert ist, läuft das PHP-Script unter der Benutzerkennung des Apache. Es kann durchaus Dateien geben, die dieser Benutzer nicht lesen darf. Um das herauszufinden, dazu dient die Funktion `is_readable()`. Ein Beispiel:

```
if(file_exists("/var/www/data/contents/c0253.txt")) {
   if(is_readable('/var/www/data/contents/c0253.txt'))
      # Datei einlesen
   else
      # Fehlerbehandlung: nicht genügend Rechte
}
else
  # Fehlerbehandlung: Datei nicht gefunden
```

Funktion:	`is_readable()`
Aufgabe:	Ermittelt, ob eine Datei oder ein Verzeichnis für das Script auslesbar ist.

Tabelle 12.21: Steckbrief Funktion »is_readable()«

Parameter:	1. Die Datei, wenn nötig mit absolutem Pfadnamen oder Pfadnamen aus Sicht des PHP-Scripts.
Rückgabewert:	`true`, wenn die Datei lesbar ist, `false`, wenn nicht.

Tabelle 12.21: Steckbrief Funktion »is_readable()« (Fortsetzung)

Analog dazu existiert eine Funktion namens `is_writable()`, um vor dem Versuch, in eine Datei zu schreiben, zu prüfen, ob dazu überhaupt die nötigen Rechte vorhanden sind.

Wenn der Dateiname oder die Pfadangabe nicht fest kodiert ist, sondern ermittelt wurde oder gar auf Anwendereingaben basiert, dann empfiehlt sich möglicherweise auch die Prüfung, ob es sich überhaupt um eine normale Datei handelt oder etwa um ein Verzeichnis oder eine Gerätedatei. Beispiel:

```
if(is_file("/var/www/data/contents/c0253")) {
  # Datei einlesen
else
  # Fehlerbehandlung: keine Datei
```

Funktion:	`is_file()`
Aufgabe:	Ermittelt, ob eine Datei eine reguläre Datei ist.
Parameter:	1. Die Datei, wenn nötig mit absolutem Pfadnamen oder Pfadnamen aus Sicht des PHP-Scripts. Nur Dateien des lokalen Dateisystems, keine URIs.
Rückgabewert:	`true`, wenn es eine Datei ist, `false`, wenn nicht.

Tabelle 12.22: Steckbrief Funktion »is_file()«

Um eine vorhandene Datei komplett in einem Rutsch in eine Variable einzulesen, stellt PHP zwei zu empfehlende Komfortfunktionen bereit: `file_get_contents()`, um den gesamten Inhalt in eine Zeichenkette zu lesen, und `file()`, um eine Datei mit Textinhalt zeilenweise in einen Array einzulesen. Beispiele:

```
$lines = file("/var/www/data/contents/c0253.txt");
foreach($lines as $line) {
  # Datei zeilenweise verarbeiten
  # Die jeweils aktuelle Zeile ist
  # in $line gespeichert
}
$content = file_get_contents("/var/www/data/contents/c0253.txt");
# Inhalt der Datei als Ganzes verarbeiten
```

Funktion:	`file()`
Aufgabe:	Liest eine Datei in einen Array ein. Der Funktionsaufruf sollte einer entsprechenden Variablen zugewiesen werden.

Tabelle 12.23: Steckbrief Funktion »file()«

Parameter:	1. Die Datei, wenn nötig mit absolutem Pfadnamen oder Pfadnamen aus Sicht des PHP-Scripts. Auch URIs auf fremden Servern können eingelesen werden, falls in ***php.ini*** die Direktive ***allow_url_fopen*** auf ***On*** oder ***1*** gesetzt ist. 2. (optional) `'1'`, um im Falle keiner Pfadangabe die Datei nicht nur im aktuellen Verzeichnis zu suchen, sondern überall in `$PATH`.
Rückgabewert:	Array mit den Zeilen der Datei oder `false`, falls ein Fehler aufgetreten ist.

Tabelle 12.23: Steckbrief Funktion »file()« (Fortsetzung)

Funktion:	`file_get_contents()`
Aufgabe:	Liest eine Datei in eine Zeichenkette ein. Der Funktionsaufruf sollte einer entsprechenden Variablen zugewiesen werden. Die Funktion ist sowohl zum Einlesen von Binärdateien als auch für Textdateien geeignet (»binary safe«).
Parameter:	1. Die Datei, wenn nötig mit absolutem Pfadnamen oder Pfadnamen aus Sicht des PHP-Scripts. Auch URIs auf fremden Servern können eingelesen werden, falls in ***php.ini*** die Direktive ***allow_url_fopen*** auf ***On*** oder ***1*** gesetzt ist. 2. (optional) `'1'`, um im Falle keiner Pfadangabe die Datei nicht nur im aktuellen Verzeichnis zu suchen, sondern überall in `$PATH`.
Rückgabewert:	Zeichenkette mit dem Inhalt der Datei oder `false`, falls ein Fehler aufgetreten ist.

Tabelle 12.24: Steckbrief Funktion »file_get_contents()«

In der Regel genügen diese beiden Funktionen den Erfordernissen zum Einlesen einer Datei. Da die Funktion `file_get_contents()` allerdings erst seit PHP-Version 4.3.0 verfügbar ist, kann es je nach installierter PHP-Version erforderlich sein, eine andere Variante zu verwenden. Es stehen mehrere zur Auswahl. Die Funktion `readfile()`, die ähnlich wie `file_get_contents()` den gesamten Dateiinhalt einliest, gilt als fehleranfällig und ist nicht »binary safe«. Sinnvoller ist es, den »klassischen« Weg über `fread()` zu wählen. Dabei müssen Sie jedoch wie bei allen »klassischen« Dateifunktionen mit einem Datei-Handle arbeiten. Ein Konstrukt mit `fread()` sieht typischerweise so aus:

```
$file = "/var/www/data/contents/c0253.txt";
$fh = fopen($file, "r");
$content = fread($fh, filesize($file));
fclose($fh);
```

Funktion:	`fopen()`
Aufgabe:	Öffnet eine Datei. Der Funktionsaufruf sollte einer Variablen zugewiesen werden.
Parameter:	1. Die Datei, wenn nötig mit absolutem Pfadnamen oder Pfadnamen aus Sicht des PHP-Scripts. Auch URIs auf fremden Servern können geöffnet werden, falls in ***php.ini*** die Direktive ***allow_url_fopen*** auf ***On*** oder ***1*** gesetzt ist.

Tabelle 12.25: Steckbrief Funktion »fopen()«

Parameter:	2. Art des Öffnens: `"r"` für Lesen, `"r+"` für Lesen und Schreiben, `"w"` für Neubeschreiben, `"w+"` für Lesen und Neubeschreiben, `"a"` für anhängendes Schreiben oder `"a+"` für Lesen und anhängendes Schreiben. Bei Textdateien unter MS Windows sollte dahinter außerdem `b` notiert werden, also z.B. `"rb"` oder `"r+b"`. 2. (optional) `'1'`, um im Falle keiner Pfadangabe die Datei nicht nur im aktuellen Verzeichnis zu suchen, sondern überall in `$PATH`.
Rückgabewert:	Ein Datei-Handle für die Datei oder `false`, falls ein Fehler aufgetreten ist.

Tabelle 12.25: Steckbrief Funktion »fopen()« (Fortsetzung)

Funktion:	`fread()`
Aufgabe:	Liest den gesamten Inhalt einer Datei. Der Funktionsaufruf sollte einer Variablen zugewiesen werden.
Parameter:	1. Datei-Handle, das zuvor mit `fopen()` erzeugt wurde. 2. Anzahl der zu lesenden Bytes. Wenn die gesamte Datei gelesen werden soll, wird üblicherweise `filesize("dateiname")` verwendet.
Rückgabewert:	Gibt den Inhalt der gelesenen Daten als Zeichenkette zurück.

Tabelle 12.26: Steckbrief Funktion »fread()«

Funktion:	`filesize()`
Aufgabe:	Ermittelt die Größe einer Datei in Byte.
Parameter:	1. Die Datei, wenn nötig mit absolutem Pfadnamen oder Pfadnamen aus Sicht des PHP-Scripts.
Rückgabewert:	Anzahl Byte der Datei oder `false`, falls ein Fehler auftrat.

Tabelle 12.27: Steckbrief Funktion »filesize()«

Funktion:	`fclose()`
Aufgabe:	Schließt eine mit `fopen()` geöffnete Datei.
Parameter:	1. Datei-Handle, das zuvor mit `fopen()` erzeugt wurde.
Rückgabewert:	`true`, wenn das Schließen erfolgreich war, `false`, wenn nicht.

Tabelle 12.28: Steckbrief Funktion »fclose()«

In einigen Fällen kann es auch sinnvoll sein, eine Datei während des Einlesens bereits zu verarbeiten. So etwa, wenn nur ein Teil der Datei gelesen werden soll, z.B. die ersten 20 Zeilen:

```
$file = "/var/www/data/contents/c0253.txt";
$fh = fopen($file, "r");
for($i = 0; $i < 20; $i++)
   $first20lines[] = fgets($fh);
fclose($fh);
```

Wieder wird die Datei mit `fopen()` geöffnet und mit `fclose()` geschlossen. In einer for-Schleife, die bis 20 zählt, wird die Datei mit der Funktion `fgets()` zeilenweise in den Array `$first20lines` eingelesen. Mit `fgets()` können Sie jeweils eine Zeile einlesen.

Funktion:	`fgets()`
Aufgabe:	Liest eine Zeile aus einer Datei. Der Dateizeiger wird entsprechend weiterbewegt. Der Funktionsaufruf sollte einer Variablen zugewiesen werden.
Parameter:	1. Datei-Handle, das zuvor mit `fopen()` erzeugt wurde. 2. (optional) Anzahl der zu lesenden Bytes. Wenn weggelassen, wird entweder bis zum Zeilenende gelesen oder maximal 1024 Byte. Sinnvoll ist der Parameter, wenn Dateien mit sehr langen Zeilen eingelesen werden sollen.
Rückgabewert:	Gibt den Inhalt der gelesenen Daten als Zeichenkette zurück.

Tabelle 12.29: Steckbrief Funktion »fgets()«

In manchen Fällen muss auch alles »zu Fuß« programmiert werden. Ein Beispiel sind Datendateien, deren Inhalte aus Feldern mit einer festen Anzahl Bytes bestehen. Angenommen, eine Datei hat 5 Felder, von denen das erste bei Position 1 beginnt, das zweite bei Position 17, das dritte bei Position 97, das vierte bei Position 113 und das fünfte bei Position 171. Die so gespeicherten Werte lassen sich beispielsweise so einlesen:

```
$fh = fopen("daten.dat", "r");
$field_positions = array(0, 16, 96, 112, 170, filesize("daten.dat"));
$fields = array();
for($i = 0; $i < count($field_positions) - 1; $i++) {
   fseek($fh, $field_positions[$i]);
   $field = "";
   $bytes = $field_positions[$i+1] - $field_positions[$i];
   for($j = 0; $j < $bytes; $j++)
      $field .= fgetc($fh);
   $fields[$i] = $field;
}
fclose($fh);
```

Die Datei wird wieder mit `fopen()` geöffnet und am Ende mit `fclose()` geschlossen. Die Positionen der Felder werden in einem Array namens `$field_positions` gespeichert. Die Werte darin sind 0-basiert, da das erste Zeichen an Position 0 steht, das zweite an Position 1 usw. Als letzter Wert im Array wird die Dateigröße der Datendatei ermittelt. Die Inhalte der fünf Felder sollen im Array `$fields` gespeichert werden.

In einer for-Schleife, die von 0 bis 4 zählt, werden die einzelnen Datenfelder eingelesen. Dabei spielt der ***Dateizeiger*** eine wichtige Rolle. Dies ist ein Positionszeiger, in dem jederzeit gespeichert ist, wen Zeichen wieviel er sich beim Lesen oder Schreiben einer geöffneten Datei gerade befindet. Mit der Funktion `fseek()` kann der Dateizeiger an einer beliebigen Stelle innerhalb der Datei positioniert werden. In unserem Beispiel setzen wir ihn in jedem for-Schleifendurchlauf an die Anfangsposition des jeweils nächsten Feldes. Die Anfangspositionen sind ja im Array `$field_positions` gespeichert. Mit `ftell()` kann übrigens jederzeit abgefragt werden, an welcher Position sich der Dateizeiger befindet.

Der Inhalt des aktuellen Feldes soll in die Variable `$field` eingelesen werden. Um zu ermitteln, wie viele Bytes eingelesen werden sollen, wird in `$bytes` die Differenz von der jeweils nächsten Position zur aktuellen ermittelt. Diese Rechnung ist auch der Grund, warum wir als letzte Position die Dateigröße selbst in den Array `$field_positions` mit aufgenommen haben.

In einer inneren for-Schleife wird dann mithilfe der Funktion `fgetc()` byte-weise aus der Datei gelesen. Die Variable `$field` wird dabei Zeichen um Zeichen gefüllt. Am Ende wird `$field` dem aktuellen Array-Element von `$fields` zugewiesen. Nach Durchlauf der äußeren for-Schleife stehen in `$fields` die einzelnen Dateninhalte.

Da wir in unseren Beispiel mit `filesize()` ermittelt haben, wie viele Bytes letztlich einzulesen sind, müssen wir uns um das Dateiende nicht kümmern. Es ist jedoch auch möglich, etwa in einer while-Schleife zeichenweise einzulesen und in der Durchlaufbedingung der Schleife zu formulieren `while(! feof($fh))`. Mit der Funktion `feof()` kann auf das Erreichen des Dateiendes geprüft werden.

Funktion:	`fseek()`
Aufgabe:	Positioniert den Dateizeiger in einer mit `fopen()` geöffneten Datei.
Parameter:	1. Datei-Handle, das zuvor mit `fopen()` erzeugt wurde. 2. Position, an die der Dateizeiger gesetzt werden soll. Um bei relativer Interpretation vor die aktuelle Position oder das Dateiende zu positionieren, kann ein negativer Wert übergeben werden. 3. (optional) Art, wie die Positionsangabe interpretiert werden soll: SEEK_SET bedeutet: gemessen am Dateianfang, SEEK_CUR: gemessen von der aktuellen Position und SEEK_END: gemessen am Dateiende.
Rückgabewert:	`0`, wenn das Positionieren erfolgreich war, `-1`, wenn nicht.

Tabelle 12.30: Steckbrief Funktion »fseek()«

Funktion:	`ftell()`
Aufgabe:	Ermittelt die aktuelle Position des Dateizeigers in einer mit `fopen()` geöffneten Datei.
Parameter:	1. Datei-Handle, das zuvor mit `fopen()` erzeugt wurde.
Rückgabewert:	Position des Dateizeigers oder `false`, wenn erfolglos.

Tabelle 12.31: Steckbrief Funktion »ftell()«

Funktion:	`fgetc()`
Aufgabe:	Liest ein einzelnes Byte aus einer mit `fopen()` geöffneten Datei.
Parameter:	1. Datei-Handle, das zuvor mit `fopen()` erzeugt wurde.
Rückgabewert:	Das eingelesene Zeichen als Zeichenkette oder `false`, wenn das Dateiende erreicht ist.

Tabelle 12.32: Steckbrief Funktion »fgetc()«

Funktion:	`feof()`
Aufgabe:	Prüft in einer mit `fopen()` geöffneten Datei, ob der Dateizeiger das Dateiende erreicht hat.
Parameter:	1. Datei-Handle, das zuvor mit `fopen()` erzeugt wurde.
Rückgabewert:	`true`, wenn das Dateiende erreicht ist, `false`, wenn nicht.

Tabelle 12.33: Steckbrief Funktion »feof()«

12.4.2 Daten ausgeben und schreiben

In jedem Fall muss sich ein als Webseite aufgerufenes PHP-Script darum kümmern, dass irgendwelche Daten an den Browser zurückgesendet werden. In der Regel ist das ein vollständiges HTML-Dokument, dessen Inhalte durch das, was das Script sonst tut, erst ermittelt werden. Neben diesen Nutzdaten kann ein Script jedoch auch den HTTP-Antwort-Header beeinflussen, den der Webserver an den aufrufenden Browser sendet, und dadurch einiges erreichen. Manche Seiten möchten neben den Nutzdaten außerdem Cookies an den aufrufenden Browser senden, um Daten beim Anwender zu speichern.

In vielen Fällen muss ein Script aber nicht nur Daten an den aufrufenden Browser ausgeben, sondern auch in Dateien schreiben, um sie dauerhaft zu speichern. Selbst einfache Anwendungen wie ein Seitenaufrufzähler oder ein kleines Gästebuch müssen bereits Inhalte in Dateien schreiben, damit sie beim nächsten Aufruf wieder von dort ausgelesen werden können.

Daten an den aufrufenden Browser ausgeben

»Daten an den aufrufenden Browser ausgeben« bedeutet aus Sicht von PHP nichts anderes, als Daten auf die so genannte ***Standardausgabe*** (gerne auch abgekürzt als ***stdout***) zu schreiben. PHP wird von einem Webserver wie Apache so gestartet, dass dieser das, was PHP auf die Standardausgabe schreibt, auffängt und via HTTP-Protokoll an den aufrufenden Browser sendet.

In den bisherigen Beispielen dieses Kapitels haben wir stets das PHP-Sprachkonstrukt `echo` verwendet, um Daten an den Browser auszugeben. Es gibt jedoch wenigstens zwei PHP-Funktionen, die dafür ebenfalls in Frage kommen. Außerdem können Sie neben den HTML-Nutzdaten auch HTTP-Header ausgeben, die der Webserver so an den aufrufenden Browser weitergibt wie vom PHP-Script formuliert.

Eine einfache Alternativfunktion zum Sprachkonstrukt `echo` ist `print`. Auch dies ist wie `echo` keine echte Funktion, sondern ein Sprachkonstrukt. Es gelten die gleichen Regeln und Möglichkeiten wie bei `echo`, also:

- Auszugebende Zeichenketten können wahlweise in doppelte oder einfache hohe Anführungszeichen gesetzt werden oder als so genanntes Here-Dokument definiert werden.
- Wenn nichts anderes als eine einzelne Variable ausgegeben werden soll, sind keine Anführungszeichen erforderlich. Wenn der Rückgabewert eines Funktionsaufrufs aus-

gegeben werden soll, dürfen um den Funktionsaufruf keine Anführungszeichen gesetzt werden.

- Bei doppelten Anführungszeichen werden Variablen im auszugebenden Inhalt erkannt und durch ihren Wert ersetzt. Komplexe Variablen wie Arraywerte oder Objekteigenschaften müssen jedoch in geschweifte Klammern eingeschlossen werden.
- Aufzählungen mehrerer auszugebender Teile sind möglich. Die Teile müssen durch Kommata getrennt werden.

Der einzige Unterschied zwischen `print` und `echo` ist, dass `print` den Wert `true` erzeugt, wenn die Ausgabe erfolgreich war, und `false`, wenn nicht.

Manchmal sollen Daten vor der Ausgabe auch noch in bestimmter Weise formatiert werden. So sollen z. B. errechnete Preise mit zwei Nachkommastellen ausgegeben werden. Bestens geeignet für solche Fälle ist die mächtige Funktion `printf()`.

Diese Funktion erkennt in einer auszugebenden Zeichenkette das Prozentzeichen als Steuerzeichen. Hinter dem Prozentzeichen können Sie mithilfe einer funktionseigenen Syntax Angaben zur Formatierung eines Werts notieren. In zusätzlichen Parametern geben Sie dann an, welche Werte gemeint sind. Ein einfaches Beispiel:

```
$mwst = 99.95 * 0.16;
printf("<p>&euro 99.95 (+ 16%% MwSt. = &euro; %.2f)</p>",$mwst);
```

Daraus wird folgende HTML-Ausgabe erzeugt:

```
_ 99.95 (+ 16% MwSt. = _ 15.99)
```

In diesem Beispiel werden an `printf()` zwei Parameter übergeben. Der erste Parameter ist stets eine auszugebende Zeichenkette. Diese wird jedoch auch als ***Prozentstring*** oder ***Formatstring*** bezeichnet. In unserem Beispiel kommt zunächst `%%` vor. In diesem Fall dient das erste Prozentzeichen jedoch nur dazu, das zweite zu maskieren. Die Zeichenfolge `%%` bedeutet also: Gib an dieser Stelle ein normales %-Zeichen aus.

Im zweiten Fall, weiter hinten, ist `%.2f` notiert. Damit wird an der entsprechenden Stelle ein Fließkommazahlwert so formatiert, dass er in jedem Fall mit zwei Stellen nach dem Dezimalpunkt formatiert wird. Welcher Wert an der betreffenden Stelle überhaupt formatiert werden soll, steht nicht an der Stelle selbst, sondern in den weiteren Parametern, die an `printf()` übergeben werden. Der erst im Formatstring mit `%` formatierte Wert wird `printf()` als zweiter Parameter übergeben, der zweite mit `%` gekennzeichnete Wert als dritter Parameter usw.

Funktion:	`printf()`, `sprintf()`
Aufgabe:	`printf()` gibt Zeichenketten mit formatierten Werten aus, `sprintf()` gibt sie nicht aus, sondern als Wert zurück. Ansonsten sind beide Funktionen gleich.

Tabelle 12.34: Steckbrief Funktionen »printf()« und »sprintf()«

Parameter:	1. Auszugebende Zeichenkette mit %-Formatanweisungen für Werte. 2. Weitere Parameter: von den %-Formatanweisungen betroffene Werte, in der Reihenfolge der %-Zeichen im Formatstring.
Rückgabewert:	Bei `sprintf()` die formatierte Zeichenkette.

Tabelle 12.34: Steckbrief Funktionen »printf()« und »sprintf()« (Fortsetzung)

Die funktionseigene Syntax für Formatanweisungen müssen Sie natürlich kennen. Nachfolgende Tabelle schlüsselt besonders nützliche Möglichkeiten auf:

Formatbeispiel	Bedeutung
`%b`	Für Ganzzahlen (Integer) geeignet. Ausgegeben wird die Bitfolge in 0-1-Schreibweise. Beispiel: `%b`, angewendet auf `22`, gibt `10110` aus.
`%c`	Für Ganzzahlen (Integer) zwischen 32 und 255 geeignet. Ausgegeben wird das Zeichen laut aktuellem Zeichensatz. Beispiel: `%c`, angewendet auf `252`, gibt `ü` aus.
`%04d`	Für Ganzzahlen (Integer) geeignet. Ausgegeben wird die Zahl. Bei Zahlen mit weniger als vier Stellen werden führende Nullen erzeugt. Beispiel: `%06d`, angewendet auf `23`, gibt `000023` aus.
`%04u`	Wie `%d`, jedoch ohne eventuelle Vorzeichen. Also nur auf positive Ganzzahlen sinnvoll anwendbar.
`%.2f`	Für Fließkommazahlen (Float, Double) geeignet. Ausgegeben wird die Zahl. Bei Zahlen mit mehr als zwei Stellen hinter dem Dezimalzeichen werden weitere Stellen abgeschnitten. Dabei wird sogar automatisch gerundet. Beispiel: `%.4f`, angewendet auf `1.23408957234`, gibt `1.2341` aus.
`%o`	Für Ganzzahlen (Integer) geeignet. Ausgegeben wird die Zahl in Oktalschreibweise. Beispiel: `%o`, angewendet auf `10`, gibt `12` aus.
`%x, %X`	Für Ganzzahlen (Integer) geeignet. Ausgegeben wird die Zahl in Hexadezimalschreibweise. Beispiel: `%x`, angewendet auf `28`, gibt `1c` aus. Bei `%X` würde `1C` ausgegeben, d.h., Ziffern a bis f werden als A bis F ausgegeben.

Tabelle 12.35: Wichtige Formatiermöglichkeiten bei printf() und sprintf()

Ein professionell programmiertes PHP-Script, so haben wir bereits gesehen, besteht nicht aus Hunderten von echo- oder print-Anweisungen, sondern liest Daten aus Templates und Datenressourcen ein, stellt daraus auszugebende Daten in einer Variablen zusammen und gibt die Variable am Ende aus.

Es gibt jedoch auch Fälle, in denen etwas anderes erwünscht ist. Ein Beispiel:

```
$count = 0;
while($count < 20) {
   system("ls -l /tmp/sess* | wc -l");
   echo " aktive Sessions aktuell auf dem Server<br>";
   flush();
   $count += 1;
   sleep(5);
}
```

Dieses Script soll 20-mal alle 5 Sekunden ausgeben, wie viele HTTP-Sessions momentan auf dem Server ermittelbar sind. Innerhalb der entsprechenden while-Schleife wird die Anzahl der Sessions durch einen system-Aufruf mit einem Linuxkommando ermittelt. Die Funktion `system()`, auf die wir allerdings erst in ***Abschnitt 12.4.8*** näher eingehen werden, gibt das Ergebnis des Aufrufs sofort aus. Eine echo-Anweisung besorgt die Restausgabe. Nun würde PHP normalerweise bis zum Script-Ende warten, bevor es Daten ausgibt, auch wenn die echo-Anweisungen z. B. wie im Beispiel mitten in einer while-Schleife stehen. Durch die Funktion `flush()` wird dies jedoch verhindert – sie bewirkt, dass alles, was bis dahin in den Ausgabepuffer ausgegeben wurde, auch tatsächlich ausgegeben wird.

Im Beispiel ist dies auch dringend nötig, da ansonsten erst nach 100 Sekunden eine Ausgabe erscheinen würde, falls das Script überhaupt so lange laufen darf. Denn die Funktion `sleep()` unterbricht die weitere Ausführung des Scripts für so viele Sekunden wie angegeben.

Funktion:	`flush()`
Aufgabe:	Leert den bisherigen Ausgabepuffer und gibt seinen Inhalt aus. Bei Web-Scripting nur möglich, wenn PHP als Apache-Modul installiert ist.
Parameter:	Keine.
Rückgabewert:	Keiner.

Tabelle 12.36: Steckbrief Funktion »flush()«

Funktion:	`sleep()`
Aufgabe:	Unterbricht die weitere Ausführung eines Scripts.
Parameter:	1. Anzahl Sekunden, die unterbrochen werden soll.
Rückgabewert:	Keiner.

Tabelle 12.37: Steckbrief Funktion »sleep()«

HTTP-Header vor Datenausgabe senden

Während das vorzeitige Entleeren des Ausgabepuffers und das Unterbrechen des Scripts eher exotische Wünsche sind, gibt es eine Funktion, die Sie für Datenausgaben mit PHP auf jeden Fall kennen sollten: `header()`. Mit dieser Funktion können Sie sowohl das HTTP-Kommando formulieren, das der Webserver senden soll, als auch Message Header für die HTTP-Antwort des Webservers hinzufügen. Ein Script kann mit dieser Funktion auf vielfältige Weise die Reaktion des Servers auf eine Anfrage beeinflussen. So kann beispielsweise in großem Umfang Performance gewonnen werden. Deshalb ist es wichtig, die Möglichkeiten dieser Funktion zu kennen, was allerdings auch Kenntnisse im HTTP-Protokoll voraussetzt.

Allen Aufrufen von `header()` ist eines gemeinsam: Sie müssen im Script ***vor*** jeder Ausgabe mit `echo`, `print` und dergleichen notiert sein. Es darf in einem Script, das mit `header()` arbeitet, auch kein HTML-Code vor dem ersten PHP-Bereich notiert sein, d.h., es ist zwingend erforderlich, dass die Script-Datei mit `<?php ...` beginnt. Nicht mal ein Leerzeichen darf davor stehen.

Funktion:	`header()`
Aufgabe:	Beeinflusst die HTTP-Antwort, die der Webserver an den aufrufenden Browser sendet.
Parameter:	1. HTTP-Kommando oder HTTP Message Header als Zeichenkette. 2. (optional) `true` oder `false`, ob diese Angabe eine eventuelle frühere Angabe zum gleichen Message Header ersetzt oder eine zweite Angabe hinzufügt. Default (also wenn nicht angegeben) ist `true`. 3. (optional) HTTP-Statuscode als Zahl.
Rückgabewert:	Laut PHP-Spezifikation ein Integerwert, jedoch nicht nachvollziehbar und auch nicht erwähnt.

Tabelle 12.38: Steckbrief Funktion »header()«

Erstes Beispiel – eine PHP-eigene Weiterleitung zu einer anderen Seite:

```
header("Location: http://www.example.org/");
exit();
```

In diesem Fall sollten anschließend keine Daten mehr mit `echo` oder Ähnlichem ausgegeben werden. Deshalb ist im Beispiel auch der Aufruf von `exit()` notiert. `Location` ist das Message-Header-Feld im HTTP-Protokoll, mit dessen Hilfe der Server den aufrufenden Browser anweisen kann, die angegebene andere Adresse anzufordern. Aus Sicht des Browser-Anwenders findet eine automatische Weiterleitung zur anderen Adresse statt.

Manchmal entstehen auch Probleme mit der Anzeige von dynamisch erzeugten Daten im Browser. Es werden noch »alte« Inhalte ausgegeben, obwohl das PHP-Script neuere Inhalte ausgibt. Die Gründe dafür können im Cache-Verhalten des Browsers oder von zwischengeschalteten Proxy-Servern liegen. Mithilfe von geeigneten HTTP-Headern können Sie dies jedoch beeinflussen.

Falls der Webserver HTTP1.1 unterstützt, können Sie folgende Message Header formulieren:

```
header("Expires: Sat, 01 Jan 2000 00:00:00 GMT");
header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
 header("Cache-Control: no-store, no-cache, must-revalidate");
```

Falls noch HTTP 1.0 verwendet wird, ist folgende Alternative angebracht:

```
header("Expires: Sat, 01 Jan 2000 00:00:00 GMT");
header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
 header("Pragma: no-cache");
```

Die ersten beiden Anweisungen sind in beiden Fällen gleich. Die jeweils erste sendet ein Cache-Verfallsdatum für die aktuelle Seite, das in der Vergangenheit liegt. Dadurch wird der Browser angewiesen, die Seite neu anzufordern und sie nicht aus dem Cache zu holen. In der zweiten Anweisung wird das Datum der letzten Änderung an dieser Seite auf den aktuellen Moment der Ausgabe gesetzt. Browser speichern diesen Zeitstempel und entscheiden bei neuerlichen Aufrufen einer Seite, ob die Seite neu angefordert werden muss,

weil ein neuerer Zeitstempel gesendet existiert. Die dritte Anweisung weist an, die ausgegebenen Daten überhaupt nicht im Cache-Speicher abzulegen und auf jeden Fall immer aktuell anzufordern.

Ebenfalls nötig ist die `header()`-Funktion, wenn Daten mit anderem Mime-Type als HTML gesendet werden sollen. Ein Beispiel:

```
$data = file_get_contents("/some/where/myflash.swf");
header("Content-type: application/x-shockwave-flash");
echo $data;
```

In diesem Fall fungiert das PHP-Script wie ein Flash-Movie. Der Anwender weiß jedoch nicht, wo die swf-Datei liegt. Sie kann auch außerhalb der Document Root liegen. So wird verhindert, dass ein Anwender direkt auf das Flash-Movie zugreifen kann. Das Script liest den Inhalt der swf-Datei ein. Dann sendet es den HTTP-Header für Flash-Movies, indem es dem HTTP Message Header `Content-type` den für Flash üblichen Mime-Type `application/x-shockwave-flash` zuweist. Die Datenausgabe wird dann vom Browser wie ein Flash-Movie behandelt.

Ebenso können Sie mit PHP auch Daten anderer Formate wie PDF, SVG, GIF, JPEG, PNG usw. senden. Sie müssen lediglich den jeweiligen Mime-Type kennen. Die maßgebliche Liste wird im Web unter *http://www.iana.org/assignments/media-types/* geführt.

Es gibt sogar Fälle, in denen ein Script gar keine Nutzdaten senden soll. Aus Sicht des Anwenders bleibt die bisherige Webseite am Bildschirm stehen. Dennoch hat eine serverseitige Datenverarbeitung stattgefunden, indem z.B. bei Absenden eines Formulars der Server kontaktiert und das PHP-Script ausgeführt hat. Dieses kann die Formulardaten speichern, ohne jedoch eine Antwortseite zu erzeugen. Stattdessen sendet es einfach folgenden HTTP-Header:

```
header("HTTP/1.0 204 No Response");
```

Cookies setzen

Wenn Cookies zu einer Seite existieren und vom Browser an den Webserver beim Aufruf der Seite übertragen wurden, dann stehen diese dem ausgeführten PHP-Script im superglobalen Array `$_COOKIES` zur Verfügung. Der Name des Cookies ist dabei ein Array-Schlüssel. Wenn es also ein Cookie mit dem Namen ***last_visited*** gibt, dann kann das Script über `$_COOKIES['last_visited']` darauf zugreifen.

Zur Verwaltung von Cookies gehört es aber auch, diese neu zu setzen. Dazu steht in PHP die Funktion `setcookie()` zur Verfügung. Für diese Funktion gelten die gleichen Regeln wie für `header()`: Der Aufruf im Script muss ***vor*** jeder Ausgabe mit `echo`, `print` und dergleichen notiert sein. Es darf auch kein HTML-Code vor dem ersten PHP-Bereich notiert sein, d.h., es ist zwingend erforderlich, dass die Script-Datei mit `<?php ...` beginnt.

Ein Beispiel:

```
setcookie("last_visited", time(), time()+60*60*24*30);
```

In diesem Fall wird ein Cookie mit dem Namen `last_visited` gesetzt. Sein Wert ist ein mit `time()` ermittelter Unix-Zeitstempel, also die Anzahl Sekunden seit 1. Januar 1970 bis zum aktuellen Moment. Im dritten Parameter wird ein Verfallszeitpunkt für das Cookie ermittelt, ebenfalls aus einem Aufruf der Funktion `time()` ermittelt, zuzüglich einer Multiplikation, welche den Wert »in 30 Tagen« in Sekunden ausrechnet und zum Rückgabewert von `time()` addiert.

Funktion:	`setcookie()`
Aufgabe:	Sendet gemeinsam mit den HTTP-Headerdaten ein Cookie zum aufrufenden Browser. Ob dieser das Cookie akzeptiert, liegt an den Cookie-Einstellungen des Anwenders. Sie können Cookies auf keinen Fall erzwingen.
Parameter:	1. Name des Cookies. 2. (optional) Wert des Cookies. Je nach Clients dürfen Cookies maximal 1024, manchmal auch 4096 Byte lang sein. Gedacht sind sie jedenfalls nur für kurze Daten. 3. (optional) Verfallszeitpunkt als Unix-Zeitstempel (Sekunden seit 1.1.1970). Übergeben Sie 0 oder einen Wert kleiner als `time()`, um das Cookie zu löschen. 4. (optional) Verzeichnispfad auf dem Server, für den das Cookie gültig sein soll. PHP-Scripts innerhalb des angegebenen Pfads können also das Cookie wieder auslesen. 5. (optional) Subdomain oder Domain, auf der das Cookie gültig sein soll. 6. (optional). 0 oder 1 übergeben. 1 bedeutet: Cookie wird nur gesetzt, wenn eine HTTPS-Verbindung besteht.
Rückgabewert:	`true`, wenn das Cookie übertragen wird, oder `false`, wenn nicht. Der Rückgabewert sagt nichts darüber aus, ob das Cookie beim Anwender gespeichert wird oder nicht!

Tabelle 12.39: Steckbrief Funktion »setcookie()«

Mithilfe von Cookies kann eine Webseite mit etwas Aufwand stark personalisiert werden. Wenn der Anwender seinen Namen in einem Formular eingibt, kann dieser Wert in einem Cookie gespeichert werden, z.B. durch einen Aufruf wie `setcookie("user_name", $_POST['user_name'])`. Beim nächsten Seitenaufruf kann das Script dann beispielsweise im Fall von `if(isset($_COOKIE['user_name']))` den Namen wieder abrufen und in die Ausgabe einer Webseite einbauen. Noch wesentlich weitreichendere Folgen hat es, wenn das Script z.B. auf Basis eines Cookies ermitteln kann, wann der Anwender die Seite zum letzten Mal aufgerufen hat, und in die Ausgabe mit einbaut, was seitdem alles an neuen Informationen hinzugekommen ist. Oder bei lernorientierten Inhalten, welche Übungen ein Anwender bereits absolviert hat und welche noch nicht absolviert sind. Das einzige Problem ist die Zuverlässigkeit von Cookies, sprich, ob ein Anwender sie akzeptiert oder nicht. Den in der Vergangenheit gegen Cookies geführten Feldzügen ist es zu verdanken, dass manche Anwender Cookies auf eine Stufe mit Viren und Würmern stellen. Demzufolge genießen Cookies allgemein auf Anwenderseite keinen so guten Ruf.

Daten in Dateien schreiben

Fast alle größeren Webanwendungen merken sich ermittelte Daten serverseitig, indem sie sie in Dateien speichern. So könnte beispielsweise eine Feedback-Formular-Anwendung, wie wir sie in ***Abschnitt 12.2.4*** kennen gelernt haben, erhaltene Feedback-Daten nicht nur an den Site-Betreiber mailen, sondern auch in einer Feedback-Datei dauerhaft speichern. Ein Beispiel:

```
$fh = fopen("feedbacks.log", "a");
fputs($fh, "-----------------------------------\n");
fputs($fh, "Name: {$_POST['Vorname']} {$_POST['Zuname']}\n");
fputs($fh, "E-Mail: {$_POST['Mail']}\n");
fputs($fh, "Zeitpunkt: ".date("d.m.Y, H:i:s")."\n");
fputs($fh, "Text:\n");
fputs($fh, $_POST['Text']);
fputs($fh, "\n\n");
fclose($fh);
```

In diesem Fall benutzen wir die Funktion `fputs()`, um Zeichenketten mit übergebenen Formulardaten in eine Datei zu schreiben. Die Funktion `fputs()` erfordert ein Datei-Handle, also dass die Datei mit `fopen()` geöffnet wird und der Rückgabewert in einer Datei-Handle-Variablen gespeichert wird. Da wir `fopen()` ebenso wie `fclose()` bereits in ***Abschnitt 12.4.1*** behandelt haben, gehen wir hier nicht mehr näher darauf ein.

Funktion:	`fputs()`, `fwrite()`
Aufgabe:	Schreibt eine Zeichenkette ab der aktuellen Position des Dateizeigers in eine Datei. `fputs()` und `fwrite()` sind identisch.
Parameter:	1. Datei-Handle, das zuvor mit `fopen()` erzeugt wurde. 2. Zeichenkette, die in die Datei geschrieben werden soll. 3. (optional) Länge, die auf jeden Fall geschrieben werden soll. Wenn weggelassen, wird auf jeden Fall die gesamte Zeichenkette geschrieben.
Rückgabewert:	Anzahl der geschriebenen Zeichen.

Tabelle 12.40: Steckbrief Funktion »fputs()«

Im obigen Beispiel wird die Datei zum anhängenden Schreiben geöffnet (`"a"`), so dass alle neu eingetroffenen Formulardaten jeweils ans Ende der Datei geschrieben werden.

Ab PHP 5.x steht auch die Komfortfunktion `file_put_contents()` zur Verfügung. Dadurch können Sie sich die Aufrufe von `fopen()`, `fclose()` und das Arbeiten mit einem Datei-Handle sparen. Beispiel:

```
$output = "-----------------------------------\n";
$output .= "Name: {$_POST['Vorname']} {$_POST['Zuname']}\n";
$output .= "E-Mail: {$_POST['Mail']}\n";
$output .= "Zeitpunkt: ".date("d.m.Y, H:i:s")."\n";
$output .= "Text:\n";
$output .= $_POST['Text'];
$output .= "\n\n";
file_put_contents("feedbacks.log", $output, FILE_APPEND);
```

In dieser Variante sammeln wir die zu schreibenden Daten zuerst in einer Variablen namens `$output`. In der letzten Anweisung wird `file_put_contents()` aufgerufen und die Daten werden anhängend in die angegebene Datei geschrieben.

Funktion:	`file_put_contents()`
Aufgabe:	Schreibt eine Zeichenkette ab der aktuellen Position des Dateizeigers in eine Datei.
Parameter:	1. Die Datei, wenn nötig mit absolutem Pfadnamen oder Pfadnamen aus Sicht des PHP-Scripts. 2. Zeichenkette, die in die Datei geschrieben werden soll. 3. (optional) Flag FILE_APPEND, falls anhängend geschrieben werden soll.
Rückgabewert:	Anzahl der geschriebenen Zeichen.

Tabelle 12.41: Steckbrief Funktion »file_put_contents()«

Eine Schreibfunktion analog zu `fgetc()` gibt es nicht. Um zeichenweise zu schreiben, müssen mit einer Funktionen wie `fputs()` lauter Zeichenketten mit jeweils einem Zeichen schreiben.

File-Locking

Bedenken Sie, dass es auf einem Webserver kein ungewöhnliches Vorkommnis ist, wenn zwei Anwender so ziemlich zum gleichen Zeitpunkt ein Formular absenden. Zwei unterschiedliche Apache-Prozesse starten ein verarbeitendes PHP-Script und dieses schreibt munter Daten in eine bestimmte Datei. Greifen aber unglücklicherweise zwei Prozesse zur gleichen Zeit auf die Datei zu, kann es zu Datenverlust und im schlimmsten Fall sogar zu einer Zerstörung aller Inhalte der Datei kommen. Es empfiehlt sich daher, einen Mechanismus zu realisieren, der verhindert, dass ein Prozess in eine Datei schreibt, während dies gerade ein anderer tut.

Es gibt einen einfachen Überschreibschutz mithilfe der Funktion `flock()`, der jedoch nur funktioniert, wenn alle Anwendungen sich daran halten. Er hat auch nur empfehlenden Charakter, d.h., er kann einen anderen Prozess nicht wirklich daran hindern, auf eine als gesperrt markierte Datei zuzugreifen.

Funktion:	`flock()`
Aufgabe:	Versieht eine Datei mit einer Markierung, dass sie in Benutzung ist.
Parameter:	1. Datei-Handle, das zuvor mit `fopen()` erzeugt wurde. 2. Art der Sperrung: `LOCK_SH` bedeutet nur lesende Verriegelung; `LOCK_EX` bedeutet exklusive, schreibende Verriegelung; `LOCK_UN` bedeutet Verriegelung mit `LOCK_SH` oder `LOCK_EX` freigeben; `LOCK_NB` bedeutet: Zugriffe während der Verriegelung erlauben.
Rückgabewert:	Bei Erfolg `true`, ansonsten `false`.

Tabelle 12.42: Steckbrief Funktion »flock()«

Da `flock()` jedoch nicht ganz unproblematisch ist (es funktioniert nicht einheitlich und gleich auf allen Plattformen), benutzen viele Programmierer andere, eigene Lösungen. Dabei wird vor allem gerne mit einer temporären Datei gearbeitet, die so lange existiert, wie ein Schreibzugriff auf eine andere Datei stattfindet. Findet ein schreibbereites Script die temporäre »Signaldatei« vor, weiß es, dass es nicht in die andere Datei schreiben und so lange warten sollte, bis die temporäre Datei nicht mehr existiert. Ein Beispiel:

```
$lock_file = ".lock";
$data_file = "daten.txt";
$wait = 0;
while(file_exists($lock_file)) {
   sleep(1);
   $wait += 1;
   if($wait > 10)
      die "Probleme beim Schreiben von Daten";
}
$fh_lock = fopen($lock_file,"w");
$fh_data = fopen($data_file,"w");
fputs($fh_data, $data);
fputs($fh_lock, "Lockfile: ".$data_file);
fclose($fh_data);
fclose($fh_lock);
unlink($lock_file);
```

In diesem Beispiel wird zwischen der eigentlichen Datendatei (`$data_file`) und einer temporären Lock-Datei (`$lock_file`) unterschieden. Letztere erhält den zumindest auf Linux/Unix-Systemen eleganten Namen `.lock` (so ist die Datei nicht per Default sichtbar).

Bevor irgendetwas in Dateien geschrieben wird, wird eine while-Schleife aufgerufen. Die Durchlaufbedingung ist, dass die Lock-Datei existiert. Das ist nämlich ein Hinweis darauf, dass ein anderer Script-Prozess die Datendatei gerade zum Schreiben geöffnet hat. Mit `sleep(1)` wird eine Sekunde gewartet. Damit, falls die Lock-Datei aus irgendeinem Grund nicht gelöscht wurde, keine Endlosschleife entsteht, wird nach 10 Versuchen, also 10 Sekunden, abgebrochen. Wenn alles regulär verläuft, sollte dieser Fall jedoch nicht eintreten.

Existiert die Lock-Datei nicht oder nicht mehr, geht es an der while-Schleife vorbei in die Passage unterhalb davon. Dort werden beide Dateien, also sowohl Lock-Datei als auch Datendatei, zum Schreiben geöffnet. In die Lockdatei wird irgendetwas geschrieben, während die Datendatei die Nutzdaten aufnimmt. Anschließend werden beide Dateien wieder geschlossen und die Lock-Datei wird mit `unlink()` (siehe auch ***Abschnitt 12.4.6***) gelöscht. Die Lock-Datei existiert also nur so lange, wie in die Datendatei geschrieben wird.

12.4.3 Zeichenketten und Arrays

Zeichenketten (Strings) und Arrays sind eigentlich zwei verschiedene Datentypen. Doch so weit liegen sie gar nicht auseinander. Denn jede Zeichenkette kann in PHP auch als Array aus Zeichen angesprochen werden. Beispiel:

```
$name = "Stefan";
echo $name[0];
```

Obwohl `$name` eine Zeichenkette speichert, kann `$name[0]` angewendet werden. Damit wird einfach das erste Zeichen der Zeichenkette angesprochen, im Beispiel also das »S«, mit `$name[1]` das zweite Zeichen usw. Man spricht auch von einem 0-basierten Index.

Länge einer Zeichenkette

Eine der am häufigsten erwünschten Informationen ist die Länge einer Zeichenkette. Beispiel:

```
if(strlen($_POST['vorname']) > 60)
  # Fehlerbehandlung
```

In der if-Abfrage wird geprüft, ob ein mittels `POST` übermittelter Formularfeldinhalt länger als 60 Zeichen ist. Eine solche Abfrage ist typisch für serverseitige Formularüberprüfung mit PHP.

Funktion:	`strlen()`
Aufgabe:	Ermittelt die Länge einer Zeichenkette.
Parameter:	1. Zeichenkette, deren Länge ermittelt werden soll.
Rückgabewert:	Anzahl Zeichen der Zeichenkette.

Tabelle 12.43: Steckbrief Funktion »strlen()«

Teilzeichenketten extrahieren

Verschiedene Zeichenkettenfunktionen nutzen den 0-basierten Index von Zeichenketten – so etwa die Standardfunktion `substr()`, mit deren Hilfe sich eine Zeichenkette extrahieren lässt. Beispiel:

```
$abc = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
echo substr($abc,0,13),"<br>";
echo substr($abc,13),"<br>";
echo substr($abc,0,1),"<br>";
```

Funktion:	`substr()`
Aufgabe:	Extrahiert aus einer Zeichenkette eine Teilzeichenkette.
Parameter:	1. Zeichenkette, aus der extrahiert werden soll. 2. Startposition im 0-basierten Index, ab der extrahiert werden soll. 3. (optional) Anzahl zu extrahierender Zeichen. Wenn weggelassen, wird bis zum Ende der Zeichenkette extrahiert.
Rückgabewert:	Gibt die extrahierte Zeichenkette zurück.

Tabelle 12.44: Steckbrief Funktion »substr()«

Der erste Aufruf im obigen Beispiel bewirkt, dass die erste Hälfte des Alphabets, also ABCDEFGHIJKLM extrahiert wird. Der zweite Aufruf extrahiert die zweite Hälfte des Alphabets, also ab 0-basierter Position 13 (= ab dem 14. Zeichen) bis zum Ende, da der

dritte Parameter weggelassen wurde. Der dritte Aufruf extrahiert genau ein Zeichen (1), nämlich das erste (0) – Letzteres ist identisch mit `$abc[0]`.

Die Funktion `substr()` eignet sich jedoch nur für Fälle, in denen Sie als Programmierer die Zeichenkette genau kennen. Bei Zeichenketten mit unbekannten Positionen empfiehlt sich beispielsweise die Funktion `strstr()`. Beispiel:

```
$mail = "someone@example.org";
echo strstr($mail,"@");
```

Dieses Beispiel gibt am Ende `@example.org` aus, weil `strstr()` nach dem ersten Vorkommen einer Zeichenkette innerhalb einer Zeichenkette sucht, im Beispiel nach dem einzelnen Zeichen »@«. Der Rest der durchsuchten Zeichenkette inklusive der gefundenen Zeichenkette wird zurückgegeben.

Falls im Beispiel wirklich nur der Domain-Name ermittelt werden soll, so bietet sich eine Kombination von `substr()` und `strstr()` an:

```
echo substr(strstr($mail,"@"), 1);
```

Damit wird `substr()` als erster Parameter das Ergebnis von `strstr()` übergeben. Da als zweiter Parameter 1 angegeben wird, gibt `substr()` die von `strstr()` ermittelte Teilzeichenkette ab dem zweiten Zeichen aus, also ohne das @. Übrig bleibt der nackte Domain-Name.

Funktion:	`strstr()`, `stristr()`, `strrstr()`
Aufgabe:	Sucht in einer Zeichenkette nach einer Teilzeichenkette und ermittelt den Rest der durchsuchten Zeichenkette ab dem ersten Vorkommen der Teilzeichenkette. `stristr()` verhält sich wie `strstr()`, ignoriert jedoch Groß- und Kleinschreibung. `strrstr()` verhält sich ebenfalls wie `strstr()`, sucht jedoch nicht nach dem ersten, sondern nach dem letzten Vorkommen der Teilzeichenkette.
Parameter:	1. Zeichenkette, die durchsucht werden soll. 2. Zeichenkette, nach der gesucht werden soll.
Rückgabewert:	Gibt die extrahierte Teilzeichenkette aus der durchsuchten Zeichenkette zurück oder `false`, falls die Zeichenkette, nach der gesucht werden sollte, nicht gefunden wurde.

Tabelle 12.45: Steckbrief Funktionen »strstr()«,»stristr()« und »strrstr()«

Eine andere Möglichkeit, eine Funktion wie `substr()` vorbereitend zu bedienen, besteht darin, mithilfe von `strpos()` nach der Position einer Teilzeichenkette im 0-basierten Index einer Zeichenkette zu suchen. Beispiel:

```
$mail = "someone@example.org";
echo substr($mail, strpos($mail,"@") + 1);
```

Die Funktion `strpos()` ermittelt wie `strstr()` das erste Vorkommen einer Teilzeichenkette in einer Zeichenkette, gibt jedoch nicht die restliche Zeichenkette zurück, sondern die Positi-

onsnummer im 0-basierten Index, an der die gefundene Teilzeichenkette beginnt. Im Beispiel wird durch Aufruf von `substr()` derjenige Teil der Mailadresse ausgegeben, der um 1 hinter der Position beginnt, die das Suchzeichen @ hat. Der Ausdruck `strpos($mail,"@") + 1` ermittelt dieses Ergebnis.

Funktion:	`strpos()`, `strrpos()`
Aufgabe:	Sucht in einer Zeichenkette nach einer Teilzeichenkette und ermittelt die Positionsnummer in der durchsuchten Zeichenkette ab dem ersten Vorkommen der Teilzeichenkette. `strrpos()` verhält sich wie `strpos()`, sucht jedoch nicht nach dem ersten, sondern nach dem letzten Vorkommen der Teilzeichenkette.
Parameter:	1. Zeichenkette, die durchsucht werden soll. 2. Zeichenkette, nach der gesucht werden soll.
Rückgabewert:	Gibt die Positionsnummer aus der durchsuchten Zeichenkette zurück oder `false`, falls die Zeichenkette, nach der gesucht werden sollte, nicht gefunden wurde.

Tabelle 12.46: Steckbrief Funktionen »strpos()« und »strrpos()«

Wenn Sie nach Teilzeichenketten mit variablem Inhalt suchen wollen, müssen Sie zu Funktionen greifen, die reguläre Ausdrücke unterstützen. Zum Suchen eignet sich vor allem die Funktion `preg_match()`, die wir bereits in ***Abschnitt 12.3.9*** kennen gelernt haben. Hier noch ein Beispiel:

```
$content = "Ein Text mit einem [link:http://www.selfhtml.org/[Link auf
SELFHTML]]";
$link_pattern = "/\[link:(.*?)\[(.*?)\]\]/";
preg_match($link_pattern, $content, $matches);
if(count($matches >= 3))
   echo "<a href=\"{$matches[1]}\">{$matches[2]}</a>";
```

Das Beispiel extrahiert aus einer Zeichenkette ein Konstrukt `[URI[Linktext]]` mit Hilfe eines regulären Ausdrucks, der `/\[link:(.*?)\[(.*?)\]\]/` lautet. Wie erkennbar, enthält der reguläre Ausdruck auch zwei geklammerte Teile, nämlich den URI und den Linktext. Nach dem Aufruf von `preg_match()` steht in `$matches[0]` die gesamte gefundene Teilzeichenkette, auf die der reguläre Ausdruck passt, in `$matches[1]` der erste geklammerte Teilausdruck, also der URI, und in `$matches[2]` der Linktext.

Funktion:	`preg_match()`, `preg_match_all()`
Aufgabe:	Sucht in einer Zeichenkette nach einem regulären Ausdruck. `preg_match()` stoppt die Weiterverarbeitung, wenn der reguläre Ausdruck gefunden wurde, `preg_match_all()` findet dagegen alle Treffer.
Parameter:	1. Regulärer Ausdruck, nach dem gesucht werden soll. 2. Zeichenkette, in der gesucht werden soll.

Tabelle 12.47: Steckbrief Funktionen »preg_match()« und »preg_match_all()«

Parameter:	3. (optional) Eine Variable, aus der bei Auffinden des regulären Ausdrucks ein Array wird, in dem an Position 0 die Teilzeichenkette steht, auf die der reguläre Ausdruck passte, und an den Positionen 1 und weiteren die Inhalte von gemerkten Klammerausdrücken innerhalb des regulären Ausdrucks.
Rückgabewert:	Anzahl der Suchtreffer, wenn das Suchmuster gefunden wurde (bei `preg_match()` maximal 1), `0` wenn es nicht gefunden wurde, und `false`, falls ein Fehler aufgetreten ist.

Tabelle 12.47: Steckbrief Funktionen »preg_match()« und »preg_match_all()« (Fortsetzung)

Die Funktion `preg_match()` bricht allerdings ab, sobald eine Teilzeichenkette auf das Muster passt. Sollen alle Treffer ermittelt werden, ist `preg_match_all()` die geeignete Wahl. Beispiel:

```
$content = "Mailen Sie an jemand@example.org oder an jemand_anders@example.org";
$mail_pattern = "/[a-zA-Z0-9\.-_]+@([a-zA-Z0-9\.-_]+)/";
preg_match_all($mail_pattern, $content, $matches);
```

Die zu durchsuchende Zeichenkette enthält zwei Mailadressen. Mithilfe der Funktion `preg_match_all()` sollen alle Mailadressen gefunden werden. Der reguläre Ausdruck für Mailadressen ist sicherlich nicht perfekt – Mailadressen können überraschend schwer zu erkennen sein. Einen vollständigen regulären Ausdruck mit beeindruckender Länge können Sie auf *http://www.ex-parrot.com/~pdw/Mail-RFC822-Address.html* bewundern.

Wichtig ist, was `preg_match_all()` in der übergebenen Array-Variablen speichert. Eine Anwendung von `var_dump()` bringt im obigen Beispiel folgendes Ergebnis:

```
array(2) {
  [0]=>
  array(2) {
    [0]=>
    string(18) "jemand@example.org"
    [1]=>
    string(25) "jemand_anders@example.org"
  }
  [1]=>
  array(2) {
    [0]=>
    string(11) "example.org"
    [1]=>
    string(11) "example.org"
  }
}
```

Es wird also ein verschachtelter Array erzeugt. Das erste Element im äußeren Array ist ein Array mit allen gefundenen Treffern, auf die das Suchmuster passte. Weitere Elemente des äußeren Arrays speichern geklammerte Ausdrücke. In unserem Beispiel haben wir den Domain-Namen geklammert. Deshalb gibt es genau ein weiteres Element im äußeren Array. Dies ist ein Array aus allen gefundenen Teilausdrücken. Die Positionsnummern der inneren Arrays sind so angeordnet, dass `$matches[0][0]` den ersten Suchtreffer enthält

und `$matches[1][0]` den ersten Teilausdruck, `$matches[0][1]` den zweiten Suchtreffer und `$matches[1][1]` den zweiten Teilausdruck.

Suchen und Ersetzen in Zeichenketten

Die Standardfunktion zum Suchen und Ersetzen in Zeichenketten ist `str_replace()`. Ein Beispiel:

```
$text = "wenn du weißt, daß man das nicht darf, mußt du nicht mehr fürchten, daß
man mit dir schimpft";
$text = str_replace("daß", "dass", $text);
$text = str_replace("muß", "muss", $text);
echo $text;
```

Im Beispiel wird `str_replace()` angewendet, um veraltete Schreibweisen von Wörtern in einem Text zu beseitigen.

Funktion:	`str_replace(), str_ireplace()`
Aufgabe:	Sucht in einer Zeichenkette nach einer Teilzeichenkette und ersetzt sie durch eine andere Teilzeichenkette. `str_ireplace()` unterscheidet im Gegensatz zu `str_replace()` nicht zwischen Groß- und Kleinschreibung.
Parameter:	1. Teilzeichenkette, nach der gesucht werden soll. 2. Teilzeichenkette, durch die (1.) ersetzt werden soll. 3. Zeichenkette, in der gesucht und ersetzt werden soll.
Rückgabewert:	Gibt die geänderte Zeichenkette zurück. Mehrere Fundstellen werden alle ersetzt.

Tabelle 12.48: Steckbrief Funktionen »str_replace()« und »str_ireplace()«

Falls eine fixe Suchzeichenkette nicht reicht, weil es Varianten oder variable Teile darin gibt, müssen Sie mit regulären Ausdrücken arbeiten. Die Suche-Ersetze-Funktion dafür ist `preg_replace()`. Ein Beispiel:

```
$text = "Heute ist der 06.04.2004. Es ist 14:54 Uhr.";
$text = preg_replace("/\d+\.\d+\.\d{4}/", date("d.m.Y"), $text);
$text = preg_replace("/\d\d:\d\d/", date("H:i"), $text);
echo $text;
```

In einem Text sind veraltete Angaben zum aktuellen Zeitpunkt gespeichert. Diese sollen durch die Werte zum Ausführungszeitpunkt ersetzt werden. Ein regulärer Ausdruck sucht zunächst nach der Datumsangabe und ersetzt sie durch einen Aufruf der Funktion `date()`, die das aktuelle Datum formatiert zurückgibt. Ein zweiter Aufruf sucht analog dazu die Uhrzeit und ersetzt sie.

Funktion:	`preg_replace()`
Aufgabe:	Sucht in einer Zeichenkette nach einem regulären Ausdruck und ersetzt gefundene Passagen.
Parameter:	1. Regulärer Ausdruck, nach dem gesucht werden soll. 2. Zeichenkette, durch die gefundene Teilzeichenketten ersetzt werden sollen. Dabei kann mit `$1`, `$2` usw. auf geklammerte Teilausdrücke im Suchausdruck zurückgegriffen werden. 3. Zeichenkette, in der gesucht werden soll. 4. (optional) Numerische Angabe, wie viele Fundstellen maximal ersetzt werden sollen. Ohne Angabe werden alle Fundstellen ersetzt.
Rückgabewert:	Gibt die geänderte Zeichenkette zurück.

Tabelle 12.49: Steckbrief Funktion »preg_replace()«

Zeichenketten zerlegen

Angenommen, Sie haben mit PHP eine Linux-Systemdatei wie *passwd* eingelesen, deren Zeilen typischerweise so aussehen:

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
```

Die einzelnen Felddaten sind jeweils durch Doppelpunkt getrennt. Um die einzelnen Felddaten einer Zeile – nehmen wir an, diese ist in einer Variablen `$line` gespeichert – zu ermitteln, können Sie die Zeile anhand des Doppelpunkts aufsplitten:

```
$fields = explode(":", $line);
```

Mithilfe von `explode()` zerlegen Sie eine Zeichenkette anhand eines Trennzeichens in einen Array. Die Funktion gibt den Array der Einzelelemente zurück, d.h., eine Variable, der dieser Funktionsaufruf zugewiesen ist, wie im Beispiel `$fields`, ist anschließend ein Array aus den Teilzeichenketten. Mit `$fields[0]` können Sie im Beispiel also auf den Benutzernamen zugreifen und mit `$fields[5]` auf den Pfadnamen seines Heimatverzeichnisses.

Die Funktion `explode()` eignet sich auch hervorragend um die Einzelwerte kommabegrenzter Daten zu separieren.

Funktion:	`explode()`
Aufgabe:	Zerlegt eine Zeichenkette in einen Array aus Teilzeichenketten an Hand einer Begrenzerzeichenkette.
Parameter:	1. Begrenzerzeichenkette. 2. Zeichenkette, die zerlegt werden soll. 3. (optional) Numerische Angabe, wie viele Teile maximal im Array gespeichert werden sollen. Wenn weggelassen, werden alle Teile gespeichert.
Rückgabewert:	Gibt einen Array aus ermittelten Teilzeichenketten zurück.

Tabelle 12.50: Steckbrief Funktion »explode()«

Falls keine fixe Begrenzerzeichenkette oder ein einzelnes Begrenzerzeichen existiert, können Sie mit regulären Ausdrücken und der Funktion `preg_split()` arbeiten. Ein Beispiel:

```
$text = "Ein Satz aus Wörtern. Halt, nicht nur, sondern auch aus Satzzeichen!";
$words = preg_split("/[\s\.,:\?\!]+/",
         $text, -1, PREG_SPLIT_NO_EMPTY);
echo count($words);
```

Ziel soll sein, im Beispielsatz die Wörter zu zählen. Dazu wird `preg_split()` auf den Satz angewendet. Der Trennzeichenausdruck findet ein oder mehrere Zeichen in Folge aus der Zeichenklasse Leerraum (`\s`), Punkt, Komma, Doppelpunkt, Fragezeichen und Ausrufezeichen. Da in einem Satz üblicherweise auch das letzte Zeichen zu dieser Zeichenklasse gehört, würde `preg_split()` am Ende jedoch ein leeres Array-Element erzeugen, das nicht erwünscht ist. Deshalb werden auch die Parameter drei und vier übergeben, wobei es auf den vierten Parameter ankommt: Mit der Konstante `PREG_SPLIT_NO_EMPTY` wird verhindert, dass leere Array-Elemente erzeugt werden.

Funktion:	`preg_split()`
Aufgabe:	Zerlegt eine Zeichenkette in einen Array aus Teilzeichenketten anhand eines regulären Begrenzerausdrucks.
Parameter:	1. Begrenzerausdruck. 2. Zeichenkette, die zerlegt werden soll. 3. (optional) Numerische Angabe, wie viele Teile maximal im Array gespeichert werden sollen. Wenn weggelassen oder `-1`, werden alle Teile gespeichert. 4. (optional): `PREG_SPLIT_NO_EMPTY` bedeutet: keine leeren Array-Elemente erzeugen. `PREG_SPLIT_DELIM_CAPTURE` bedeutet: Auch geklammerte Teilausdrücke im Begrenzerausdruck werden im Array mit zurückgegeben. `PREG_SPLIT_OFFSET_CAPTURE` bedeutet: Mit jedem Element wird auch dessen Position in der zerlegten Zeichenkette mit zurückgegeben (jedes Array-Element ist dabei selber ein Array).
Rückgabewert:	Gibt einen Array aus ermittelten Teilzeichenketten zurück.

Tabelle 12.51: Steckbrief Funktion »preg_split()«

Funktionen für HTML-Inhalte

Da PHP in der Praxis viel mit dem Erzeugen und auch mit dem Auslesen von HTML-Code zu tun hat, gibt es einige Funktionen, die speziell für diese Arbeiten entwickelt wurden und den Programmierer unterstützen.

Ein Anwendungsfall besteht darin, in einer Zeichenkette alle HTML-eigenen Zeichen durch ihre HTML-Maskierungen zu ersetzen, also < durch `<` usw. Das ist praktisch, wenn Sie HTML- oder auch XML-Code in HTML ausgeben wollen. Beispiel:

```
echo htmlspecialchars("<html><head></head><body></body></html>");
```

Die Funktion `htmlspecialchars()` erzeugt daraus eine Zeichenfolge, in der alle HTML-eigenen Zeichen maskiert sind, so dass der Code als Code im Browser ausgegeben werden kann.

Funktion:	`htmlspecialchars()`, `htmlentities()`
Aufgabe:	Maskiert in einer Zeichenkette alle HTML-eigenen Zeichen nach HTML-Maskierungsvorschrift. Die Funktion `htmlentities()` wandelt zusätzlich alle Zeichen oberhalb des ASCII-Zeichensatzes in HTML-Entities um.
Parameter:	1. Zeichenkette, in der die Ersetzungen vorgenommen werden sollen. 2. (optional): `ENT_COMPAT` (Default) ersetzt nur doppelte Anführungszeichen, nicht aber einfache. `ENT_QUOTES` ersetzt beide Arten von Anführungszeichen, `ENT_NOQUOTES` ersetzt gar keine Anführungszeichen. 3. (optional) Zeichensatzangabe. Default ist `ISO-8859-1`.
Rückgabewert:	Gibt die geänderte Zeichenkette zurück.

Tabelle 12.52: Steckbrief Funktionen »htmlspecialchars()« und »htmlentities()«

Noch einen Schritt weiter geht die Funktion `strip_tags()`. Sie entfernt einfach alles HTML-Markup aus einer Zeichenkette. Sehr nützlich ist diese Funktion etwa, um Formulareingaben von Anwendern, die wieder als Webseite sichtbar sein sollen (wie etwa in Gästebüchern, Bugtrackern oder Foren), von Versuchen zu bereinigen, sich mittels HTML-Markup »optisch hervorzutun« oder gar Schaden anzurichten. Typisches Anwendungsbeispiel:

```
$text = strip_tags($_POST['text']);
```

In diesem Beispiel wird aus einem mittels `POST` übermittelten Formularfeldinhalt alles darin enthaltene HTML-Markup entfernt. Der restliche Text bleibt erhalten und wird in `$text` gespeichert.

Funktion:	`strip_tags()`
Aufgabe:	Entfernt HTML-Markup aus einer Zeichenkette.
Parameter:	1. Zeichenkette, aus der Tags entfernt werden sollen. 2. (optional) Zeichenkette aus Tags, die trotzdem erlaubt sein sollen.
Rückgabewert:	Gibt die geänderte Zeichenkette zurück.

Tabelle 12.53: Steckbrief Funktion »strip_tags()«

Sehr interessant ist auch das Arbeiten mit dem zweiten Parameter. Dadurch lässt sich sehr einfach skalieren, welche »harmlosen« Tags erlaubt sein sollen. Alle anderen werden entfernt. Beispiel:

```
$html = "<i>erlaubt</i> und <b>erlaubt</b> und <script>document.write('nicht
erlaubt');</script>";
$text = strip_tags($html, "<i></i><b></b>");
echo $text;
```

In diesem Beispiel enthält die zu bearbeitende Zeichenkette eine mit ... als kursiv und eine mit ... als fett markierte Passage sowie einen script-Bereich. Im Aufruf von strip_tags() wird festgelegt, dass Markierungen für fett und kursiv nicht entfernt werden sollen. Alles andere wird dagegen entfernt.

Um HTML-Code, der voll von HTML-Entities ist, für ein anderes Datenformat genießbar zu machen, können die Entities in normale Zeichen umgewandelt werden. Beispiel:

```
$content = file_get_contents("news.html");
$content = strip_tags($content);
$content = html_entity_decode($content);
```

Im Beispiel werden aus einer eingelesenen HTML-Datei zunächst alle Tags entfernt und anschließend mit html_entity_decode() alle Entities in ihre Zeichen umgewandelt.

Funktion:	html_entity_decode()
Aufgabe:	Wandelt HTML-Entities in normale Zeichen um.
Parameter:	1. Zeichenkette, in der die Ersetzungen vorgenommen werden sollen. 2. (optional): ENT_COMPAT (Default) ersetzt Entities für doppelte Anführungszeichen, nicht aber einfache. ENT_QUOTES ersetzt Entities für beide Arten von Anführungszeichen, ENT_NOQUOTES ersetzt gar keine Entities für Anführungszeichen. 3. (optional) Zeichensatzangabe. Default ist ISO-8859-1.
Rückgabewert:	Gibt die geänderte Zeichenkette zurück.

Tabelle 12.54: Steckbrief Funktion »html_entity_decode()«

Manchmal kann es auch wünschenswert sein, HTML-Code automatisch einzufügen, um zuvor ermittelte Inhalte an den Browser auszugeben. Einfaches Beispiel: Eine Textdatei wurde mit file_get_contents() in eine Variable namens $content eingelesen und diese soll nun im Browser ausgegeben werden. Ein einfaches echo $content würde den Text nicht sehr schön darstellen, da HTML-Parser bekanntlich Zeilenumbrüche ignorieren bzw., falls es das einzige Whitespace-Zeichen an der Stelle ist, durch ein Leerzeichen ersetzen. Um Zeilenumbrüche explizit darzustellen, muss jeder Zeilenumbruch durch <br> ausgezeichnet werden. Genau das übernimmt die Funktion nl2br(). Beispiel:

```
$content = file_get_contents("readme.txt");
echo "<p>", nl2br($content), "</p> ";
```

Auf diese Weise wird die gesamte Textdatei in einem p-Element ausgegeben. Zeilenumbrüche innerhalb der eingelesenen Datei werden durch <br> ersetzt, so dass sie in der HTML-Ausgabe ordentlich dargestellt werden.

Funktion:	nl2br()
Aufgabe:	Ersetzt in einer Zeichenkette Zeilenumbrüche durch das HTML-Markup .

Tabelle 12.55: Steckbrief Funktion »nl2br()«

Parameter:	1. Zeichenkette, in der die Ersetzung vorgenommen werden soll.
Rückgabewert:	Gibt die geänderte Zeichenkette zurück.

Tabelle 12.55: Steckbrief Funktion »nl2br()« (Fortsetzung)

Kryptografische Zeichenkettenfunktionen

PHP ermöglicht es, Zeichenketten nach üblichen Standards zu verschlüsseln oder für Sicherheitsmaßnahmen zu behandeln. So steht beispielsweise `md5()` als Hash-Funktion zur Verfügung. Beispiel:

```
if(md5($_POST['password']) == $db_md5_passwd)
   # alles OK
```

Das Beispiel fragt ab, ob die Anwendung von `md5()` auf ein mittels HTML-Formular übergebenes Passwort dem Wert einer Variablen namens `$db_md5_passwd` entspricht. In dieser könnte beispielsweise ein Wert gespeichert sein, der zuvor aus einer Datenbank eingelesen wurde. Dort sollten in sensiblen Anwendungen nämlich sicherheitshalber nicht die Passwörter selbst, sondern lediglich »eindeutige Abdrücke« davon abgespeichert werden. MD5 erlaubt es, aus einer beliebigen Zeichenkette einen 32 Byte langen »Abdruck« zu erzeugen. Es handelt sich um ein Einwegverfahren, d.h., aus dem so erzeugten Abdruck kann die Originalzeichenkette nicht mehr ermittelt werden. Wohl aber erzeugt eine erneute Anwendung von MD5 den gleichen »Abdruck« wie zuvor. Auf diese Weise lassen sich Passwörter überprüfen, ohne selbst gespeichert werden zu müssen.

Funktion:	`md5()`
Aufgabe:	Wendet das MD5-Verfahren auf eine Zeichenkette an.
Parameter:	1. Zeichenkette, auf die das MD5-Verfahren angewendet werden soll.
Rückgabewert:	Gibt den 32 Byte langen MD5-Abdruck der Zeichenkette zurück.

Tabelle 12.56: Steckbrief Funktion »md5()«

Das Hash-Funktionsverfahren von SHA1 bietet die gleiche Funktionalität, gilt aber als noch robuster. Die Verwendung der entsprechenden PHP-Funktion `sha1()` ist analog zu der von `md5()`.

Funktion:	`sha1()`
Aufgabe:	Wendet das SHA1-Verfahren auf eine Zeichenkette an.
Parameter:	1. Zeichenkette, auf die das SHA1-Verfahren angewendet werden soll.
Rückgabewert:	Gibt den 32 Byte langen SHA1-Abdruck der Zeichenkette zurück.

Tabelle 12.57: Steckbrief Funktion »sha1()«

Anstelle einer eindeutigen Zeichenkette als »Abdruck« gibt es auch Verfahren, die eine Prüfsumme erzeugen. Durch Wiederanwendung kann ermittelt werden, ob eine Zeichenkette noch die gleiche Prüfsumme hat wie z.B. die, die zu einem früheren Zeitpunkt

gespeichert wurde. Geeignet ist das Prüfsummenverfahren vor allem, um die Integrität von Daten zu überprüfen und zu prüfen, ob Daten sich geändert haben. Die entsprechende PHP-Funktion heißt `crc32()` und ermittelt einen »unsigned Integer« (vorzeichenlose, d.h. positive Ganzzahl) als Wert. Typisches Beispiel:

```
if(crc32($content) == crc32($backup_content))
   # Produktivdaten sind identisch zur Backup-Version
```

Funktion:	`crc32()`
Aufgabe:	Wendet das 32-Bit-Prüfsummenverfahren auf eine Zeichenkette an.
Parameter:	1. Zeichenkette, auf die das Prüfsummenverfahren angewendet werden soll.
Rückgabewert:	Gibt die Prüfsumme zurück.

Tabelle 12.58: Steckbrief Funktion »crc32()«

Diverse Zeichenkettenfunktionen

Es gibt noch zahlreiche andere Zeichenkettenfunktionen, die jedoch teilweise exotisch sind und in der Praxis selten zur Anwendung gelangen. Einige der häufigeren sollen noch kurz vorgestellt werden:

- Mit `strtoupper()` können Sie alle Kleinbuchstaben einer Zeichenkette in Großbuchstaben umwandeln und mit `strtolower()` alle Großbuchstaben in Kleinbuchstaben.

Funktion:	`strtoupper()`, `strtolower()`
Aufgabe:	Wandelt Klein- in Großbuchstaben um bzw. umgekehrt.
Parameter:	1. Zeichenkette, in der ersetzt werden soll.
Rückgabewert:	Gibt die ersetzte Zeichenkette zurück.

Tabelle 12.59: Steckbrief Funktionen »strtoupper()« und »strtolower()«

- Mit `ltrim()` können Sie Leerraumzeichen (Leerzeichen, Tabulatorzeichen, Hex-0-Zeichen, Zeilenumbruchzeichen) am Beginn einer Zeichenkette entfernen, mit `rtrim()` am Ende einer Zeichenkette und mit `trim()` an beiden Enden.

Funktion:	`ltrim()`, `rtrim()`, `trim()`
Aufgabe:	Entfernt Leerraumzeichen am Anfang, Ende oder beiden Enden einer Zeichenkette.
Parameter:	1. Zeichenkette, in der Leerraumzeichen entfernt werden sollen.
Rückgabewert:	Gibt die ersetzte Zeichenkette zurück.

Tabelle 12.60: Steckbrief Funktionen »ltrim()«, »rtrim()« und »trim()«

- Mit `convert_uuencode()` können Sie eine Zeichenkette nach dem UU-Algorithmus kodieren und mit `convert_uudecode()` eine UU-kodierte Zeichenkette wieder dekodieren.

Funktion:	`convert_uuencode(), convert_uudecode()`
Aufgabe:	Kodiert bzw. dekodiert Zeichenketten nach UU-Standard.
Parameter:	1. Zeichenkette, die kodiert bzw. dekodiert werden soll.
Rückgabewert:	Gibt die kodierte bzw. dekodierte Zeichenkette zurück.

Tabelle 12.61: Steckbrief Funktionen »convert_uuencode()«,und »convert_uudecode()«

Funktionen für Arrays

PHP bietet eine Menge mächtiger Funktionen an, die das Management von Serienvariablen (Arrays) vereinfachen. Eine der am häufigsten verlangten Funktionen ist `count()`, welche die Länge eines Arrays, also die Anzahl der darin gespeicherten Elemente ermittelt. Die Funktion wird beispielsweise häufig in for-Schleifen mit Zählervariable verwendet:

```
for($i = 0; $i < count($Arrayname); $i++)
```

Die for-Schleife zählt dann von 0 bis zu 1 weniger als die Anzahl der Array-Elemente. Das ist auch genau der Wertebereich für Zugriffe auf den 0-baiserten Index eines Arrays.

Funktion:	`count()`
Aufgabe:	Ermittelt die Anzahl der Elemente in einem Array.
Parameter:	1. Array, dessen Länge ermittelt werden soll.
Rückgabewert:	Gibt die Anzahl der Array-Elemente zurück.

Tabelle 12.62: Steckbrief Funktion »count()«

Eine andere wichtige Gruppe von Funktionen betrifft das Hinzufügen oder Löschen von Array-Elementen. Um neue Elemente am Ende hinzuzufügen, genügt es, Folgendes zu notieren:

```
$Arrayname[] = "neuer Wert";
```

Durch die leeren Klammern hinter dem Variablennamen des Arrays wird PHP mitgeteilt, dass ein neues Element hinzugefügt werden soll. Der neue Wert kann einfach zugewiesen werden. Diese Syntax funktioniert insbesondere auch bei assoziativen Arrays. Beispiel:

```
$user_config['page_bg_color'] = "rgb(255,204,153)";
```

Für 0-index-basierte Arrays, also solche ohne Schlüsselnamen, können Sie jedoch auch Funktionen verwenden. Die Funktion zum Einfügen neuer Elemente am Ende heißt in PHP `array_push()`:

```
array_push($Arrayname, "neuer Wert");
```

Dies ist identisch mit der obigen Zuweisung.

Funktion:	`array_push()`
Aufgabe:	Fügt ein oder mehrere Elemente am Ende eines Arrays ein.
Parameter:	1. Betroffener Array. 2. Erster einzufügender Wert. 3. Weitere: (optional) weitere einzufügende Werte.
Rückgabewert:	Gibt die neue Anzahl der Elemente des Arrays zurück.

Tabelle 12.63: Steckbrief Funktion »array_push()«

Soll der neue Wert am Anfang eines Arrays oder irgendwo in der Mitte eingefügt werden, sind Aufrufe entsprechender Funktionen erforderlich. Um einen Wert am Anfang einzufügen, können Sie die Funktion `array_unshift()` verwenden. Beispiel:

```
$names = array("Lukas", "Sahrah", "Marvin", "Julia");
array_unshift($names, "Fabian");
echo $names[0];
```

Im Beispiel wird einem Array ein neuer Name am Anfang hinzugefügt. Ausgegeben wird »Fabian«, da dies das neue erste Element des Arrays ist und mit `$names[0]` auf das erste Array-Element zugegriffen wird.

Funktion:	`array_unshift()`
Aufgabe:	Fügt ein oder mehrere Elemente am Anfang eines Arrays ein.
Parameter:	1. Betroffener Array. 2. Erster einzufügender Wert. 3. Weitere: (optional) weitere einzufügende Werte.
Rückgabewert:	Gibt die neue Anzahl der Elemente des Arrays zurück.

Tabelle 12.64: Steckbrief Funktion »array_unshift()«

Um ein Array-Element zu ersetzen, dessen Schlüsselname oder Indexnummer Ihnen bekannt ist, genügt wieder eine einfache Zuweisung. Beispiele:

```
$names[2] = "Florian";
$user_config['page_bg_color'] = "rgb(255,223,200)";
```

Wenn Sie ein bestimmtes Array-Element löschen wollen, müssen Sie ebenfalls dessen Indexnummer oder – bei assoziativen Arrays – den Schlüsselnamen kennen. Indexbasierte Arrays und assoziative Arrays unterscheiden sich dabei jedoch. Für 0-indexbasierte Arrays gibt es die mächtige und vielfältig einsetzbare Funktion `array_splice()`, die sowohl das Löschen von Elementen als auch das Ersetzen durch andere Elemente ermöglicht, wobei die Anzahl der gelöschten und der neu eingefügten Elemente unterschiedlich sein kann. Beispiele:

```
$names = array("Lukas", "Sahrah", "Marvin", "Julia");
array_splice($names,2,1);
```

Damit wird der Name »Marvin« aus dem Array `$names` gelöscht. Der zweite an die Funktion `array_splice()` übergebene Parameter teilt die Indexnummer eines zu entfernenden Array-Elements mit und der dritte Parameter die Anzahl der ab dort zu entfernenden Elemente.

```
$names = array("Lukas", "Sahrah", "Marvin", "Julia");
array_splice($names,2,0,"Florian");
```

Damit wird gar nichts gelöscht, sondern der Name »Florian« wird zwischen »Sahrah« und »Florian« neu eingefügt. Mit dem Wert 2 wird das dritte Array-Element angesprochen. Gelöscht werden soll nichts (0), aber eingefügt: Der vierte an `array_splice()` übergebene Parameter erlaubt das Einfügen neuer Elemente.

```
$names = array("Lukas", "Sahrah", "Marvin", "Julia");
array_splice($names,2,1, array("Florian", "Daniela", "Maximilian"));
```

In diesem Fall wird der Name »Marvin« gelöscht (2, 1). Als vierter Parameter wird ein Array aus drei neuen Namen übergeben. Dieser Array wird an der Stelle des gelöschten Namens eingefügt. Hinterher speichert `$namen` also die sechs Elemente »Lukas«, »Sahrah«, »Florian«, »Daniela«, »Maximilian« und »Julia«.

Funktion:	`array_splice()`
Aufgabe:	Löscht Elemente aus einem Array ab einer bestimmten Indexposition, falls gewünscht, und ersetzt sie, falls gewünscht, durch ein oder mehrere neue Elemente.
Parameter:	1. Betroffener Array. 2. Indexnummer, ab der gelöscht werden soll. 3. (optional) Anzahl zu löschender Elemente. 0 bedeutet: kein Element löschen. Wenn weggelassen, wird alles bis zum Array-Ende gelöscht. 4. (optional) Element, das an der Stelle neu eingefügt werden soll. Array übergeben, um mehrere Elemente einzufügen.
Rückgabewert:	Gibt den geänderten Array zurück.

Tabelle 12.65: Steckbrief Funktion »array_splice()«

In assoziativen Arrays können Sie Array-Elemente mit der sonst für Einzelvariablen typischen Funktion `unset()` löschen. Beispiel:

```
unset($user_config['page_bg_color']);
```

Das Array-Element mit dem Schlüsselnamen `page_bg_color` wird dadurch aus dem Array `$user_config` gelöscht.

Sprachkonstrukt:	`unset()`
Aufgabe:	Löscht Variablen, aber auch Elemente aus assoziativen Arrays oder Werte aus 0-index-basierten Arrays, wobei bei Letzteren das Element selbst erhalten bleibt (der Speicherplatz bleibt also erhalten, ist nur leer).
Parameter:	1. Betroffene Variable.
Rückgabewert:	`unset()` ist ein Sprachkonstrukt und gibt nichts zurück.

Tabelle 12.66: Steckbrief Sprachkonstrukt »unset()«

Einfacher ist es, wenn Elemente am Anfang oder am Ende eines Arrays gelöscht werden sollen. Dazu stehen die Funktionen `array_shift()` und `array_pop()` zur Verfügung, die sowohl auf assoziative als auch auf 0-index-basierte Arrays anwendbar sind. Beispiel:

```
$names = array("Lukas", "Sahrah", "Marvin", "Julia");
echo array_shift($names), ", ", array_pop($names);
```

Das Beispiel entfernt mit `array_shift()` den ersten Namen des Arrays, also »Lukas«, und mit `array_pop()` den letzten, also »Julia«. Außerdem macht sich die echo-Anweisung zu Nutze, dass beide Funktionen das jeweils entfernte Array-Element zurückgeben und gibt daher beide Namen aus.

Funktion:	`array_shift()`
Aufgabe:	Löscht das erste Array-Element. Alle anderen wandern eins nach vorne.
Parameter:	1. Betroffener Array.
Rückgabewert:	Gibt das gelöschte Element zurück.

Tabelle 12.67: Steckbrief Funktion »array_shift()«

Funktion:	`array_pop()`
Aufgabe:	Löscht das letzte Array-Element.
Parameter:	1. Betroffener Array.
Rückgabewert:	Gibt das gelöschte Element zurück.

Tabelle 12.68: Steckbrief Funktion »array_pop()«

Arrays sortieren

Eine weitere wichtige Aktion, die häufig mit Arrays durchgeführt wird, ist das Sortieren. PHP bietet dazu gleich ein ganzes Set an Funktionen an:

- `sort()` sortiert einen 0-index-basierten Array aufsteigend.
- `rsort()` sortiert einen 0-index-basierten Array absteigend.
- `asort()` sortiert einen assoziativen Array nach Werten aufsteigend.
- `arsort()` sortiert einen assoziativen Array nach Werten absteigend.
- `ksort()` sortiert einen assoziativen Array nach Schlüsselnamen aufsteigend.

- `krsort()` sortiert einen assoziativen Array nach Schlüsselnamen absteigend.
- `natsort()` sortiert einen 0-index-basierten Array aufsteigend mit zusätzlicher Intelligenz, so dass etwa *img10.gif* nicht vor *img9.gif* kommt, sondern hinterher.
- `usort()` sortiert einen 0-index-basierten Array mithilfe einer benutzerdefinierten Vergleichsfunktion.
- `uasort()` sortiert einen assoziativen Array nach Werten mithilfe einer benutzerdefinierten Vergleichsfunktion.
- `uksort()` sortiert einen assoziativen Array nach Schlüsselnamen mithilfe einer benutzerdefinierten Vergleichsfunktion.

Bei allen Sortierfunktionen, die nicht mit benutzerdefinierten Vergleichsfunktionen arbeiten, können Sie in einem zusätzlichen Flag die Art der Sortierung angeben. Allerdings denkt PHP beim Sortieren von vorneherein mehr mit als die meisten anderen Programmiersprachen. Beispiel:

```
$str_numbers = array("67","23","18","100");
sort($str_numbers);
```

In diesem Beispiel werden Zahlen als Zeichenketten in einem Array gespeichert. Dennoch bewirkt die einfache Anwendung von `sort()` die Sortierung 18, 23, 67, 100. Andere Programmiersprachen würden stur 100, 18, 23, 67 sortieren.

Funktionen:	`sort()`, `asort()`, `rsort()`, `arsort()`, `ksort()`, `krsort()`
Aufgabe:	Sortiert einen Array nach Werten. Unterschiede siehe Aufzählung weiter oben.
Parameter:	1. Betroffener Array. 2. (optional): `SORT_REGULAR` für normalen Vergleich, `SORT_NUMERIC` für numerischen Vergleich und `SORT_STRING` für Zeichenkettenvergleich.
Rückgabewert:	Gibt nichts zurück.

Tabelle 12.69: Steckbrief Sortierfunktionen für Arrays

Arrays durchsuchen

Bei assoziativen Arrays kommt es häufig vor, dass überprüft werden soll, ob ein Array-Element mit einem bestimmten Schlüsselnamen vorhanden ist. Dazu dient die Funktion `array_key_exists()`, die sich hervorragend für if-Bedingungen eignet. Beispiel:

```
if(array_key_exists('user_bg_color'), $config)
   # tu was
```

Das Beispiel fragt ab, ob im assoziativen Array `$config` ein Element mit dem Schlüsselnamen `user_bg_color` existiert.

Funktion:	`array_key_exists()`
Aufgabe:	Sucht nach einem Schlüsselnamen in einem assoziativen Array.
Parameter:	1. Gesuchter Schlüsselname. 2. Betroffener Array.
Rückgabewert:	Gibt `true` zurück, wenn der Schlüsselname existiert, und `false`, wenn nicht.

Tabelle 12.70: Steckbrief Funktion »array_key_exists()«

Die entsprechende Funktion für Werte lautet `in_array()`:

```
if(in_array('Lukas'), $names)
   # tu was
```

Funktion:	`in_array()`
Aufgabe:	Sucht in einem Array nach einem Wert.
Parameter:	1. Gesuchter Wert. 2. Betroffener Array. 3. (optional): `TRUE`, wenn auch Datentypen übereinstimmen sollen, oder `FALSE` (Default), wenn nicht.
Rückgabewert:	Gibt `true` zurück, wenn der Wert existiert, und `false`, wenn nicht.

Tabelle 12.71: Steckbrief Funktion »in_array()«

Um einen Array nach einem Wert zu durchsuchen und später auf das Element zugreifen zu können, können Sie `array_search()` verwenden. Beispiel:

```
$names = array("Lukas", "Sahra", "Marvin", "Julia");
$result = array_search("Julia", $names);
if($result == false)
 echo "nichts gefunden";
else echo "an Position $result gefunden";
```

Die Funktion gibt ein boolesches `false` zurück, wenn nichts gefunden wurde, und ansonsten den Schlüsselnamen oder die Indexnummer des ersten gefundenen Elements. Daher die Konstruktion der Abfrage im Beispiel.

Funktion:	`array_search()`
Aufgabe:	Sucht in einem Array nach einem Wert.
Parameter:	1. Gesuchter Wert. 2. Betroffener Array. 3. (optional): `TRUE`, wenn auch Datentypen übereinstimmen sollen, oder `FALSE` (Default), wenn nicht.
Rückgabewert:	Gibt `true` zurück, wenn der Schlüsselname existiert, und `false`, wenn nicht.

Tabelle 12.72: Steckbrief Funktion »array_search()«

Speziellere Array-Funktionen

PHP bietet noch etliche andere Array-Funktionen an. Hier nur Auswahl:

- `array_sum($arrayvar)` liefert die Summe von numerischen Werten aus dem Array `$arrayvar`.
- `array_reverse($arrayvar)` dreht einen Array um. Das vorher erste Element wird zum letzten, das zweite zum vorletzten usw.
- `array_unique($arrayvar)` entfernt doppelte Werte aus einem Array.
- `array_map("Funktionsname", $arrayvar)` ruft für alle Elemente des Arrays `$arrayvar` die Funktion `Funktionsname()` auf und übergibt ihr das jeweils aktuelle Array-Element als Parameter.

In Zusammenhang mit Zeichenketten wurde die Funktion `explode()` vorgestellt, die eine Zeichenkette anhand eines Begrenzerzeichens in einen Array zerteilt. Die Umkehrfunktion heißt `implode()` und führt Arraywerte zu einer Zeichenkette zusammen. Beispiel:

```
$names = array("Lukas", "Sahra", "Marvin", "Julia");
$csv_str = implode("," $names);#
```

Das Beispiel erzeugt aus den Array-Elementen von `$names` eine Zeichenkette, in der die Namen durch Kommata getrennt sind.

Funktion:	`implode()`
Aufgabe:	Führt Array-Elemente zu einer Zeichenkette zusammen und verwendet dabei eine Begrenzerzeichenkette. Der Funktionsaufruf sollte einer Zeichenkettenvariablen zugewiesen werden.
Parameter:	1. Begrenzerzeichenkette. 2. Betroffener Array.
Rückgabewert:	Gibt eine Zeichenkette mit den Arraywerten und Begrenzerzeichen zurück.

Tabelle 12.73: Steckbrief Funktion »implode()«

12.4.4 Datum und Zeit

PHP rechnet intern wie andere, auf C basierende Programmiersprachen mit dem »Unix-Urknall«, welcher auf den 1.1.1970, 0:00 Uhr datiert ist. So liefert z. B. die Funktion `time()` die Anzahl Sekunden zurück, die zwischen diesem Zeitpunkt in der Vergangenheit und dem aktuellen Zeitpunkt laut Rechnerzeit vergangen sind (Unix-Zeitstempel). Dabei wird `time()` meistens benötigt, um andere Funktionen zu bedienen. Ein Beispiel:

```
srand(time());
$random_value = rand(1,99);
echo $random_value;
```

In diesem Beispiel soll eine Zufallszahl zwischen 1 und 99 ermittelt werden. Dazu sollte der Zufallsgenerator zunächst mithilfe der Funktion `srand()` initialisiert werden. Die

Funktion erwartet als Parameter eine Zahl. Als Wert übergeben wir ihr den Rückgabewert von `time()`. So wird der Zufallsgenerator jede Sekunde mit einem anderen Wert initialisiert und `rand()` liefert jede Sekunde einen neuen Zufallswert.

Funktion:	`time()`
Aufgabe:	Liefert die Anzahl Sekunden seit dem 1.1.1970, 0:00 Uhr GMT (Unix-Zeitstempel).
Parameter:	1. Begrenzerzeichenkette. 2. Betroffener Array.
Rückgabewert:	Gibt den numerischen Wert der Sekundenzahl zurück.

Tabelle 12.74: Steckbrief Funktion »time()«

Benutzt wird die Funktion `time()` auch gerne in der wohl bekanntesten und beliebtesten Datums-/Zeit-Funktion von PHP, in `date()`. Zunächst einmal jedoch ein Beispiel ohne Aufruf von `time()`:

```
$date = date("d.m.Y");
$time = date("H:i");
echo "<p>Heute ist der $date. Es ist $time Uhr</p>";
```

Die Funktion `date()` macht aus einem Unix-Zeitstempel ein lesbares Format. Der gewünschte Unix-Zeitstempel kann im zweiten Parameter übergeben werden. Wird kein zweiter Parameter übergeben, so nimmt `date()` automatisch den aktuellen Zeitpunkt an. `$date = date("d.m.Y")` ist also das Gleiche wie `$date = date("d.m.Y", time())`.

Funktion:	`date()`
Aufgabe:	Formatiert Datum und Uhrzeitangaben für den aktuellen oder einen übergebenen Zeitpunkt nach Wunsch.
Parameter:	1. Formatstring für die Art, wie der Zeitpunkt dargestellt werden soll. 2. (optional) Sekundenwert eines Unix-Zeitstempels.
Rückgabewert:	Gibt eine Zeichenkette auf Basis des Formatstrings zurück.

Tabelle 12.75: Steckbrief Funktion »date()«

Um `date()` richtig zu verstehen, müssen Sie die Möglichkeiten des Formatstrings kennen, den die Funktion als ersten Parameter erwartet. Es handelt sich um eine Zeichenkette, in der nachfolgende Zeichen als Platzhalter erkannt werden und bei der Rückgabe durch den entsprechenden Wert ersetzt werden:

Zeichen	Bedeutung
`a`	Je nach Uhrzeit ***am*** oder ***pm***
`A`	Je nach Uhrzeit ***AM*** oder ***PM***
`B`	Swatch-Internetzeit

Tabelle 12.76: Zeichen für Formatstring bei date()

Zeichen	Bedeutung
c	Zeitstempel nach ISO 8601, z.B. *2005-03-15T12:18:21+00:00*. Erst ab PHP 5.x
d	Monatstag zweistellig, bei Werten unter 10 mit führender 0
D	Wochentag dreistellig englisch, also *Mon*, *Tue*, *Wed*, *Thu*, *Fri*, *Sat* oder *Sun*
F	Monatsname englisch, also *January*, *February* usw. bis *December*
g	Stunden der Uhrzeit 12-stündig, bei Werten unter 10 einstellig
G	Stunden der Uhrzeit 24-stündig, bei Werten unter 10 einstellig
h	Stunden der Uhrzeit 12-stündig und zweistellig, bei Werten unter 10 mit führender 0
H	Stunden der Uhrzeit 24-stündig und zweistellig, bei Werten unter 10 mit führender 0
i	Minuten der Uhrzeit zweistellig, bei Werten unter 10 mit führender 0
I	1 bei Sommerzeit, 0 bei Winterzeit
j	Monatstag, bei Werten unter 10 einstellig
l	Wochentag englisch, also *Monday*, *Tuesday* usw. bis *Sunday*
L	1, wenn Schaltjahr; 0, wenn nicht
m	Monat als Zahl, bei Werten unter 10 mit führender 0
M	Monatsname englisch abgekürzt, also *Jan, Feb, Mar, Apr, May; Jun, Jul, Aug, Sep, Oct, Nov, Dec*
n	Monat als Zahl, bei Werten unter 10 einstellig
O	Unterschied zu Greenwich Mean Time (GMT), z.B. *+0100*
r	Zeitstempel nach RFC 1822, z.B. *Thu, 15 Mar 2005 12:18:21 +0100*
s	Sekunden der Uhrzeit, bei Werten unter 10 mit führender 0
t	Anzahl Tage des Monats (28, 29, 30 oder 31)
T	Zeitzoneneinstellung des Rechners, z.B. *EST*, *MDT* usw.
U	Anzahl Sekunden seit 1.1.1970, identisch mit `time()`
w	Tag der Woche numerisch, 0 = Sonntag, 1 = Montag usw. bis 6 = Samstag
W	Wochennummer des Jahres nach ISO-8601
Y	Jahreszahl, vierstellig
y	Jahreszahl, zweistellig
z	Tag des Jahres
Z	Zeitzonen-Offset in Sekunden, bei West ein negativer Wert, bei Ost ein positiver

Tabelle 12.76: Zeichen für Formatstring bei date() (Fortsetzung)

Ein Aufruf wie `date("W")` gibt also die Wochennummer des Jahres aus und ein Aufruf wie `date("h:i:s a")` eine sekundengenaue 12-stündige Uhrzeit mit am/pm-Angabe.

Interessant ist es, mit `time()` einen anderen Zeitpunkt als die Gegenwart zu berechnen und diesen dann als Basis für eine Ausgabe mit `date()` zu nutzen. Beispiel:

```
echo "<p>Ihr Account verfällt am ", date("d.m.Y", time()+(14*24*60*60)),"</p>";
```

In diesem Fall wird `date()` dazu genutzt, um einem Benutzer mitzuteilen, dass sein Account in 14 Tagen ausläuft. Das Datum, welches vom aktuellen Zeitpunkt an gerechnet

14 Tage in der Zukunft liegt, wird `date()` im zweiten Parameter mitgeteilt, und zwar durch Aufruf von `time()` plus 14 Tage mal 24 Stunden mal 60 Minuten mal 60 Sekunden.

Um deutsche Wochentags- und Monatsnamen in einer Ausgabe zu benutzen, müssen Sie zunächst Arrays mit den entsprechenden Namen definieren. Diese können Sie dann in die Ausgabe mit einbauen. Beispiel:

```
$weekdays_de = array("Sonntag", "Montag", "Dienstag",
"Mittwoch", "Donnerstag", "Freitag", "Samstag");
$months_de = array("Januar", "Februar", "März", "April", "Mai",
"Juni", "Juli", "August", "September", "Oktober",
"November", "Dezember");

$weekday_de = $weekdays_de[date("w")];
$month_de = $months_de[date("n") - 1];
$dd = date("d");
$YYYY = date("Y");
$time = date("H:i");
echo "<p>Heute ist $weekday_de, der $dd. $month_de $YYYY. Es ist $time Uhr</p>";
```

In `$weekdays_de` werden die deutschen Wochentagnamen gespeichert, und zwar in der Reihenfolge, wie sie `date("w")` als numerische Werte liefert, also mit Sonntag beginnend. Für die deutschen Monatsnamen wird der Array `$months_de` eingerichtet. Anschließend werden die benötigten Einzelteile für die gewünschte Ausgabe ermittelt. Dabei wird der benötigte Arrayindex von `$weekdays_de` und `$months_de` durch geeignete Aufrufe von `date()` ermittelt. Die Ausgabe am Ende liefert Ergebnisse wie:

```
Heute ist Dienstag, der 15. März 2005. Es ist 12:53 Uhr
```

Weiter oben haben wir den Unix-Zeitstempel für das Datum in 14 Tagen mit `time()+(14*24*60*60)` ermittelt. Es gibt jedoch auch den Fall, dass ein Datum oder ein Zeitpunkt in der Vergangenheit oder Zukunft bekannt ist und dessen Unix-Zeitstempel ermittelt werden soll. Dazu gibt es die Funktion `mktime()`. Ein Beispiel:

```
echo "<p>News seit Kalenderwoche ", date("W", mktime(0, 0, 0, 3, 1, 2005)), "</
p>";
```

In diesem Beispiel wird `date()` als Unix-Zeitstempel ein Aufruf von `mktime()` übergeben. Dabei wird der Zeitstempel für den 1.3.2005, 0:00 Uhr, ermittelt.

Funktion:	`mktime()`
Aufgabe:	Gibt den Unix-Zeitstempel für einen festlegbaren Zeitpunkt zurück. Die übergebenen Parameter bestimmen den Zeitpunkt.
Parameter:	1. (optional) Stunden der Uhrzeit des gewünschten Zeitpunkts 2. (optional) Minuten der Uhrzeit des gewünschten Zeitpunkts 3. (optional) Sekunden der Uhrzeit des gewünschten Zeitpunkts 4. (optional) Monat des gewünschten Zeitpunkts

Tabelle 12.77: Steckbrief Funktion »mktime()«

Parameter:	5. (optional) Monatstag des gewünschten Zeitpunkts 6. (optional) Jahr des gewünschten Zeitpunkts 7. (optional) 1 für Sommerzeit oder 0 für Winterzeit
Rückgabewert:	Gibt den Unix-Zeitstempel für den festgelegten Zeitpunkt zurück.

Tabelle 12.77: Steckbrief Funktion »mktime()« (Fortsetzung)

Die Funktion `mktime()` eignet sich auch für einige Spezialitäten, da sie nicht schlüssige Werte automatisch korrigiert. Wird beim Parameter für den Monatstag beispielsweise 0 übergeben, so wird der letzte Tag des vorangegangenen Monats angenommen. Dies kann sogar hilfreich sein, wie das folgende Beispiel zeigt:

```
echo "<p>Geben Sie Ihre Unterlagen bis spätestens ", date("d", mktime(0, 0, 0,
date("n")+1, 0, 2005)), ". dieses Monats ab!</p>";
```

Das Beispiel nennt einen Monatstag, der in jedem Monat den letzten Tag des Monats ausgibt, also im Januar den 31., im Februar abhängig von der Schaltjahrsregelung den 28. oder 29. usw. Dazu wird beim vierten Parameter von `mktime()` ein Aufruf von `date()` übergeben, der die Monatszahl des nächsten Monats (`+1`) ermittelt. Da im fünften Parameter `0` übergeben wird, ermittelt `mktime()` den letzten Tag des aktuellen Monats.

Falls Sie ein gegebenes Datum auf Gültigkeit überprüfen wollen, eignet sich die Funktion `checkdate()`. Angenommen, ein Anwender hat in einem HTML-Formular seinen Geburtstag in einem Feld `name="birthday"` eingegeben. Das Formular wurde über die POST-Methode abgesendet. In diesem Fall können Sie das eingegebene Datum ohne komplexe eigene Datumsberechnungen überprüfen:

```
$birthday = explode(".", $_POST['birthday']);
if(! checkdate($birthday[1], $birthday[0], $birthday[2]))
   echo "Geburtsdatum fehlerhaft";
else echo "Geburtsdatum korrekt";
```

Angenommen, der Anwender hat den 29.2.1961 angegeben. Mittels `explode()` wird die Angabe aus dem Formular zunächst zerlegt. In `$birthday` sollte dann ein Array aus drei Elementen stehen. In einer if-Abfrage wird geprüft, ob das Ergebnis, das `checkdate()` liefert, negativ ist (`!`-Operator).

Funktion:	`checkdate()`
Aufgabe:	Überprüft ein Datum auf Gültigkeit.
Parameter:	1. Monat des Datums. 2. Monatstag des Datums. 3. Jahr des Datums.
Rückgabewert:	`true`, wenn das Datum gültig ist, und `false`, wenn nicht.

Tabelle 12.78: Steckbrief Funktion »checkdate()«

Kalender- und Feiertagsberechnung

Im Gegensatz zu vielen anderen Programmiersprachen verfügt PHP seit dem final Release von PHP 4.0 auch über ein Modul mit Kalenderfunktionen. Die Funktionen dienen vor allem für das Arbeiten und Konvertieren zwischen verschiedenen Datumsformaten (julianisches Datum, Datum nach gregorianischem Kalender, Datum nach jüdischem Kalender usw.). In der Praxis interessant ist vermutlich vor allem die Funktion `easter_date()`, die einen Unix-Zeitstempel für den Ostersonntag, 0:00 Uhr, eines Jahres ermittelt. Das erspart eigene Funktionen dieser Art. Beispiel:

```
$easter_sunday_2005 = date("d.m.Y", easter_date(2005));
```

In der Variablen `$easter_sunday_2005` wird das Datum des Ostersonntags 2005 in der Form TT.MM.JJJJ gespeichert.

Funktion:	`easter_date()`
Aufgabe:	Ermittelt den Unix-Zeitstempel für einen Ostersonntag.
Parameter:	1. (optional) Jahr. Wenn weggelassen, wird automatisch das aktuelle Jahr angenommen.
Rückgabewert:	Unix-Zeitstempel für den Ostersonntag, 0:00:00 Uhr.

Tabelle 12.79: Steckbrief Funktion »easter_date()«

Durch einfache Arithmetik lassen sich ausgehend von Ostersonntag die übrigen osterabhängigen beweglichen Tage im Jahr ermitteln. Eine selbst geschriebene Funktion gibt diese Tage als assoziativen Array zurück. Nachfolgendes Beispiel zeigt die Funktion und Beispielaufrufe:

```
function get_easter_depending_days($year) {
   $es = easter_date($year);
   $sd = 24 * 60 * 60;
   $days = array(
      'rose_monday' => $es - (48 * $sd),
      'shrove_tuesday' => $es - (47 * $sd),
      'ash_wednesday' => $es - (46 * $sd),
      'palm_sunday' => $es - (7 * $sd),
      'good_friday' => $es - (2 * $sd),
      'easter_sunday' => $es,
      'easter_monday' => $es + (1 * $sd),
      'low_sunday' => $es + (7 * $sd),
      'ascension_day' => $es + (39 * $sd),
      'whit_sunday' => $es + (49 * $sd),
      'whit_monday' => $es + (50 * $sd),
      'corpus_christi' => $es + (60 * $sd)
   );
   return($days);
}

$days = get_easter_depending_days(2005);
```

```
echo "Rosemontag: ",
     date ("d.m.Y", $days['rose_monday']), "<br>";
echo "Fastnachtsdienstag: ",
     date ("d.m.Y", $days['shrove_tuesday']), "<br>";
echo "Aschermittwoch: ",
     date ("d.m.Y", $days['ash_wednesday']), "<br>";
echo "Palmsonntag: ",
     date ("d.m.Y", $days['palm_sunday']), "<br>";
echo "Karfreitag: ",
     date ("d.m.Y", $days['good_friday']), "<br>";
echo "Ostersonntag: ",
     date ("d.m.Y", $days['easter_sunday']), "<br>";
echo "Ostermontag: ",
     date ("d.m.Y", $days['easter_monday']), "<br>";
echo "Weißer Sonntag: ",
     date ("d.m.Y", $days['low_sunday']), "<br>";
echo "Christi Himmelfahrt: ",
     date ("d.m.Y", $days['ascension_day']), "<br>";
echo "Pfingstsonntag: ",
     date ("d.m.Y", $days['whit_sunday']), "<br>";
echo "Pfingstmontag: ",
     date ("d.m.Y", $days['whit_monday']), "<br>";
echo "Fronleichnam: ",
     date ("d.m.Y", $days['corpus_christi']), "<br>";
```

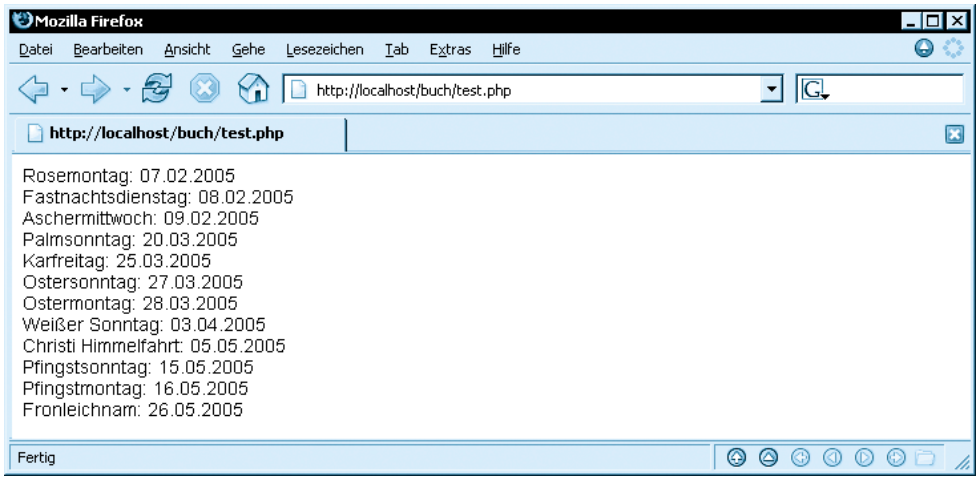


Abbildung 12.15: Browser-Ausgabe des Quelltextbeispiels für osterabhängige Feiertage

12.4.5 Mathematische Funktionen

PHP bietet auch für wissenschaftliche oder kaufmännische Berechnungen ein üppiges Arsenal an vordefinierten Funktionen an. Viele davon erwarten einen oder zwei Werte und geben das Ergebnis einer Standardoperation zurück. Nachfolgende Tabelle listet diese Funktionen auf:

Funktion	Bedeutung
`abs($x)`	Gibt den Absolutwert (immer positiv) von `$x` zurück.
`acos($x)`	Gibt den Arcus Cosinus von `$x` zurück.
`asin($x)`	Gibt den Arcus Sinus von `$x` zurück.
`atan($x)`	Gibt den Arcus Tangens von `$x` zurück.
`atan2($x,$y)`	Gibt den Arcus Tangens von `$x` und `$y` zurück.
`ceil($x)`	Gibt die nächsthöhere Ganzzahl zu `$x` zurück.
`cos($x)`	Gibt den Cosinus von `$x` zurück.
`exp($x)`	Gibt e (Konstante) hoch `$x` zurück.
`floor($x)`	Gibt die nächstniedrigere Ganzzahl zu `$x` zurück.
`hypot($x,$y)`	Gibt die Hypothenusenlänge eines rechtschenkligen Dreiecks zurück, wobei `$x` und `$y` die beiden Seitenlängen sind. Entspricht `sqrt($x*$x + $y*$y)`.
`log10($x)`	Gibt den Logarithmus von `$x` zur Basis 10 zurück.
`log($x)`	Gibt den Logarithmus von `$x` zur Basis e (Konstante) zurück.
`max($x,$y, ...)`	Gibt den größten der übergebenen Werte zurück.
`min($x,$y, ...)`	Gibt den kleinsten der übergebenen Werte zurück.
`pow($x,$y)`	Gibt `$x` hoch `$y` zurück.
`round($x)`	Gibt die kaufmännisch gerundete nächste Ganzzahl von `$x` zurück.
`sin($x)`	Gibt den Sinus von `$x` zurück.
`sqrt($x)`	Gibt die Quadratwurzel von `$x` zurück.
`tan($x)`	Gibt den Tangens von `$x` zurück.

Tabelle 12.80: Auswahl mathematischer Standardfunktionen in PHP

Interessant ist darüber hinaus das Set an Zahlenbasis-Konvertierungsfunktionen, die PHP anbietet:

Funktion	Bedeutung
`bindec($str)`	Konvertiert einen als Zeichenkette (`$str`) aus 0 und 1 bestehenden Binärwert in eine Dezimalzahl und gibt diese zurück, z. B. `bindec("10110")`.
`decbin($x)`	Konvertiert die Dezimalzahl `$x` in eine aus 0 und 1 bestehende Zeichenkette, welche die Binärdarstellung von `$x` ist, und gibt diese zurück.
`dechex($x)`	Konvertiert die Dezimalzahl `$x` in eine aus 0123456789ABCDEF bestehende Zeichenkette, welche die Hexadezimalschreibweise von `$x` darstellt, und gibt diese zurück.
`decoct($x)`	Konvertiert die Dezimalzahl `$x` in eine aus 01234567 bestehende Zeichenkette, welche die Oktalschreibweise von `$x` darstellt, und gibt diese zurück.
`octdec($str)`	Konvertiert einen als Zeichenkette (`$str`) aus 01234567 bestehenden Oktalwert in eine Dezimalzahl und gibt diese zurück, z. B. `octdec("1000")`.
`hexdec($str)`	Konvertiert einen als Zeichenkette (`$str`) aus 0123456789ABCDEF bestehenden Hexadezimalwert in eine Dezimalzahl und gibt diese zurück, z. B. `hexdec("ABC")`.

Tabelle 12.81: Zahlenbasis-Konvertierfunktionen in PHP

Wichtig für die Praxis sind auch Funktionen, die Zahlen auf Gültigkeit hin überprüfen. Diese Funktionen geben `true` oder `false` zurück, je nachdem, ob die Zahl im Sinne der Prüfung gültig ist oder nicht. Die Funktionen eignen sich damit hervorragend zum Einbau in if-Abfragen, z. B. `if(is_finite($x)) ...`.

Funktion	Bedeutung
`is_finite($x)`	Überprüft, ob `$x` eine endliche Zahl innerhalb des Wertebereichs ist, den PHP auf der benutzten Plattform verarbeiten kann. Maßgeblich sind die Grenzen für Float-Zahlen.
`is_infinite($x)`	Überprüft, ob `$x` eine Zahl außerhalb des Wertebereichs ist, den PHP auf der benutzten Plattform verarbeiten kann. Maßgeblich sind die Grenzen für Float-Zahlen.
`is_nan($x)`	Prüft, ob `$x` keine Zahl ist.

Tabelle 12.82: PHP-Funktionen zur Überprüfung von Zahlen

12.4.6 Datei- und Verzeichnisverwaltung

Auch wenn PHP seltener als etwa Perl als komfortabler Ersatz für die Shellprogrammiersprache zum Einsatz kommt, so bietet es doch alle Funktionen, die für eine komplette Dateiverwaltung nötig sind. Dadurch eignet es sich beispielsweise dazu, um webbasierte Oberflächen für die Verwaltung von Server-Rechnern zu realisieren. Viele größere Hosting-Provider bieten ihren Kunden solche Verwaltungsoberflächen bereits fertig an.

In der Regel sollten Dateioperationen allerdings im Hintergrund und ohne direkte Schnittstelle zum Webanwender durchgeführt werden. Legen Sie zum Speichern von Benutzerdaten ruhig Verzeichnisse an oder kopieren oder löschen Sie Dateien. Kritisch wird es jedoch in dem Augenblick, wo Sie Benutzereingaben ohne weitere Prüfung in Dateioperationen umsetzen.

Verzeichnis-Management

Sofern Sie nicht mit absoluten Pfadnamen und bekannten Dateien arbeiten, ist es in der Regel nötig, sich erst einmal im Dateisystem zu orientieren. So wie Sie sich als menschlicher Benutzer auf Shell-Ebene mithilfe entsprechender Kommandos orientieren, kann Ihr PHP-Script sich mithilfe dafür vorgesehener PHP-Funktionen orientieren.

Das aktuelle Arbeitsverzeichnis kann mit der Funktion `getcwd()` ermittelt werden. In der Regel ist dies das Verzeichnis, in dem das PHP-Script abgelegt ist. Die Funktion ermittelt den vollen Pfadnamen. Die Schreibweise des Pfadnamens ist allerdings betriebssystemabhängig, d. h. unter MS Windows werden zur Trennung von Verzeichnispfaden Backslashes im Rückgabewert benutzt, auf Unix/Linux-Systemen dagegen einfache Schrägstriche (Slashes).

Funktion:	`getcwd()`
Aufgabe:	Ermittelt das aktuelle Arbeitsverzeichnis.
Parameter:	Keine.
Rückgabewert:	Absoluter Pfadname des aktuellen Arbeitsverzeichnisses in der Schreibweise des Betriebssystems.

Tabelle 12.83: Steckbrief Funktion »getcwd()«

Um in ein anderes Verzeichnis zu wechseln, können Sie `chdir()` benutzen. Hierbei können Sie einheitlich auf allen Betriebssystemen einfache Schrägstriche zur Verzeichnistrennung verwenden. PHP setzt diese wenn nötig in Backslashes um. Beispiel:

```
echo getcwd(),"<br>";
chdir("/");
echo getcwd();
```

Das Beispiel gibt mit `getcwd()` zunächst das Verzeichnis aus, in dem sich das PHP-Script selbst als Datei befindet. Dann wird mit `chdir()` zum Wurzelverzeichnis gewechselt, wobei dies unter MS Windows lediglich das Startverzeichnis des aktuellen Laufwerks ist, während es auf Unix/Linux-Plattformen das Startverzeichnis des gesamten Dateisystems ist. Das neue aktuelle Verzeichnis wird dann mit `getcwd()` ausgegeben.

Funktion:	`chdir()`
Aufgabe:	Wechselt das aktuelle Arbeitsverzeichnis.
Parameter:	Pfadangabe zum neuen Verzeichnis
Rückgabewert:	`true`, wenn der Wechsel erfolgreich was, und `false`, wenn nicht.

Tabelle 12.84: Steckbrief Funktion »chdir()«

Um den Inhalt eines Verzeichnisses einzulesen, müssen Sie ähnlich wie beim Einlesen von Dateien mit »klassischen« Funktionen das Verzeichnis öffnen, dabei ein Handle erzeugen, den Inhalt einlesen und das Verzeichnis dann wieder schließen. Ein typisches Konstrukt sieht so aus:

```
$dh = opendir("/var/www");
if($dh) {
   while($file = readdir($dh)) {
      if ($file == "." || $file == "..")
          continue;
       else
          $files[] = $file;
   }
   closedir($dh);
}
else
   die("Fehler beim Öffnen des Verzeichnisses");
```

Mit `opendir()` wird ein Verzeichnis geöffnet. Der Pfadname des Verzeichnisses muss als Parameter übergeben werden. Ob das Verzeichnis geöffnet werden kann, hängt natürlich davon ab, ob es überhaupt existiert, aber z. B. auch davon, ob der Benutzer, unter dem das PHP-Script läuft (z. B. der Benutzer, als der der ausführende Webserver läuft), überhaupt Leserecht für das betreffende Verzeichnis hat. Wenn das Verzeichnis geöffnet werden kann, liefert `opendir()` ein Verzeichnis-Handle zurück. Im Beispiel wird es in der Variablen `$dh` gespeichert (steht für ***directory handle***).

Damit existiert ein Verzeichniszeiger. Mit jedem Aufruf der Funktion `readdir()` wird der jeweils nächste Verzeichniseintrag gelesen. Es bietet sich an, dies in einer while-Schleife zu tun, so wie im Beispiel. Dabei ist der Rückgabewert von `readdir()` die Durchlaufbedingung. Liefert `readdir()` keinen Verzeichniseintrag mehr, wird die while-Schleife beendet.

Jeder eingelesene Verzeichniseintrag steht innerhalb der while-Schleife zunächst in der Variablen `$file`. Der Variablenname darf nicht missverstanden werden: Es kann sich um Dateinamen handeln, aber auch um Namen von Unterverzeichnissen. Sogar die reservierten Einträge `.` (aktuelles Verzeichnis) und `..` (Elternverzeichnis) werden mit erfasst. Diese beiden filtern wir in unserer Beispiel-while-Schleife jedoch heraus. Alle anderen Einträge wandern in den Array `$files`.

Am Ende sollte ein Verzeichnis-Handle mit `closedir()` wieder freigegeben werden.

Funktion:	`opendir()`
Aufgabe:	Öffnet ein Verzeichnis zum Auslesen.
Parameter:	1. Pfadname zum gewünschten Verzeichnis, relativ aus Sicht des aktuellen Arbeitsverzeichnisses oder absolut.
Rückgabewert:	Verzeichnis-Handle, wenn erfolgreich, oder `false`, wenn nicht.

Tabelle 12.85: Steckbrief Funktion »opendir()«

Funktion:	`readdir()`
Aufgabe:	Liest den jeweils nächsten Eintrag aus einem mit `opendir()` geöffneten Verzeichnis.
Parameter:	1. Verzeichnis-Handle, das mit `opendir()` erzeugt wurde.
Rückgabewert:	Name eines Verzeichniseintrags.

Tabelle 12.86: Steckbrief Funktion »readdir()«

Funktion:	`closedir()`
Aufgabe:	Schließt ein mit `opendir()` geöffnetes Verzeichnis.
Parameter:	1. Verzeichnis-Handle, das mit `opendir()` erzeugt wurde.
Rückgabewert:	Keiner.

Tabelle 12.87: Steckbrief Funktion »closedir()«

Ab PHP 5.0 steht diesem klassischen Funktionsensemble auch eine Komfortfunktion als Alternative zur Seite: `scandir()`.

Funktion:	`scandir()`
Aufgabe:	Ermittelt die Einträge eines Verzeichnisses.
Parameter:	1. Pfadname zum gewünschten Verzeichnis, relativ aus Sicht des aktuellen Arbeitsverzeichnisses oder absolut. 2. (optional): 1 übergeben, wenn die Einträge absteigend statt aufsteigend sortiert werden sollen.
Rückgabewert:	Array mit allen ermittelten Einträgen.

Tabelle 12.88: Steckbrief Funktion »scandir()«

Mit PHP können Sie auch neue Verzeichnisse anlegen und bestehende Verzeichnisse löschen. Dazu stehen die Funktionen `mkdir()` und `rmdir()` bereit. Das PHP-Script muss allerdings Schreibrecht in dem Verzeichnis besitzen, in dem das Verzeichnis angelegt werden soll oder aus dem ein Verzeichnis gelöscht werden soll. Beim Löschen muss selbstverständlich auch für das zu löschende Verzeichnis Schreibrecht existieren.

Funktion:	`mkdir()`
Aufgabe:	Erstellt ein neues Verzeichnis.
Parameter:	1. Pfadname des gewünschten neuen Verzeichnisses, relativ aus Sicht des aktuellen Arbeitsverzeichnisses oder absolut. 2. (optional) Unix-Rechtestring in Oktalschreibweise mit führender 0 für das neue Verzeichnis, z. B. `0775`.
Rückgabewert:	`true`, wenn das Verzeichnis angelegt wurde, und `false`, wenn nicht.

Tabelle 12.89: Steckbrief Funktion »mkdir()«

Funktion:	`rmdir()`
Aufgabe:	Löscht ein Verzeichnis. Das Verzeichnis muss leer sein, darf also keine Dateien oder Unterverzeichnisse mehr enthalten. Diese müssen gegebenenfalls zuerst gelöscht werden. Dateien können mit `unlink("Dateiname")` gelöscht werden.
Parameter:	1. Pfadname des zu löschenden Verzeichnisses, relativ aus Sicht des aktuellen Arbeitsverzeichnisses oder absolut.
Rückgabewert:	`true`, wenn das Verzeichnis gelöscht wurde, und `false`, wenn nicht.

Tabelle 12.90: Steckbrief Funktion »rmdir()«

Informationen zu Verzeichniseinträgen

Wenn Sie wie weiter oben beschrieben die Einträge eines Verzeichnisses eingelesen haben, so könnte der nächste Wunsch darin bestehen, Informationen zu den einzelnen Einträgen zu ermitteln oder ein ordentlich formatiertes Verzeichnislisting auszugeben. Hier noch einmal ein komplettes Beispiel mit Einlesen eines Verzeichnisses:

```
$dir = "/var/www";
$dh = opendir($dir);
if($dh) {
   while($file = readdir($dh)) {
      if ($file == "." || $file == "..")
          continue;
       else
          $files[] = $file;
   }
   closedir($dh);
}
$dirs = 0;
$normal_files = 0;
$links = 0;
foreach($files as $file) {
   if(is_dir($dir."/".$file))
      $dirs += 1;
   if(is_file($dir."/".$file))
      $normal_files += 1;
   if(is_link($dir."/".$file))
      $links += 1;
}
echo "Verzeichnis enthält $normal_files Dateien, ";
echo "$dirs Verzeichnisse und $links Links";
```

Nach dem Einlesen der Verzeichniseinträge werden drei Variablen `$dirs`, `$normal_files` und `$links` mit 0 initialisiert. In einer foreach-Schleife wird der Array `$files`, der die eingelesenen Verzeichniseinträge enthält, abgearbeitet. Der jeweils aktuelle Eintrag steht innerhalb des Schleifenkörpers in der Variablen `$file` zur Verfügung. Mithilfe der drei Prüffunktionen `is_dir()`, `is_file()` und `is_link()` werden die Verzeichniseinträge auf die Art des Eintrags hin überprüft. Beachten Sie, dass in `$file` jeweils nur ein Verzeichniseintrag, aber kein Pfadname steht. Die drei genannten Prüffunktionen erwarten jedoch eine eindeutige Dateiadressierung. Deshalb werden ihnen im Beispiel eine zusammengesetzte Zeichenkette aus der anfänglich definierten Variablen `$dir`, dem Literal `"/"` und dem aktuellen Verzeichniseintag `$file` übergeben.

Das Beispiel zählt einfach die Eintragstypen und gibt am Ende aus, wie viele Einträge des jeweiligen Typs im Verzeichnis gefunden wurden.

Funktion:	`is_dir()`, `is_file()`, `is_link()`
Aufgabe:	`is_dir()` prüft, ob ein Verzeichniseintrag ein Verzeichnis ist, `is_file()`, ob es eine normale Datei ist, und `is_link()`, ob es ein symbolischer (nicht harter) Link ist.
Parameter:	1. Pfadname des Verzeichniseintrags, relativ aus Sicht des aktuellen Arbeitsverzeichnisses oder absolut.
Rückgabewert:	`true`, wenn es der überprüfte Typ ist, und `false`, wenn nicht.

Tabelle 12.91: Steckbrief Funktionen »is_dir()«, »is_file()« und »is_link()«

Bevor Sie mit PHP eine Datei zum Lesen oder Schreiben öffnen oder sie gar als Kommando ausführen möchten, kann es ratsam sein zu prüfen, ob diese für das Script überhaupt les-, schreib- oder ausführbar ist. Ebenso kann es vor dem Öffnen von Verzeichnissen sinnvoll sein zu prüfen, ob das Verzeichnis für das Script überhaupt lesbar ist. Entscheidend ist dabei die Benutzerkennung, unter der das Script läuft. Wenn PHP als Apache-Modul läuft, dann hat das Script die Rechte des Users, der für den Apache Webserver angelegt wurde. Dieser darf üblicherweise manches lesen, schreiben und ausführen, aber bei weitem nicht alles.

Die Funktionen für entsprechende Prüfungen heißen `is_readable()`, `is_writable()` und `is_executable()`. Vom Aufbau her funktionieren sie genau wie `is_dir()` und die anderen zuvor behandelten Funktionen.

Funktion:	`is_readable()`, `is_writable()`, `is_executable()`
Aufgabe:	`is_readable()` prüft, ob ein Verzeichniseintrag für das Script lesbar ist, `is_writable()`, ob er änderbar oder löschbar ist, und `is_executable()`, ob er ausführbar ist.
Parameter:	1. Pfadname des Verzeichniseintrags, relativ aus Sicht des aktuellen Arbeitsverzeichnisses oder absolut.
Rückgabewert:	`true`, wenn die Prüfung zutrifft, und `false`, wenn nicht.

Tabelle 12.92: Steckbrief Funktionen »is_readable()«, »is_writable()« und »is_executable()«

Beachten Sie, dass die Funktion `is_executable()` unter MS Windows erst seit PHP 5.0 zur Verfügung steht.

Um weitere Detailinformationen zu Verzeichniseinträgen zu ermitteln, gibt es eine ganze Reihe von Einzelfunktionen. Eine davon, `filesize()`, haben wir bereits früher in Zusammenhang mit `fread()` kennen gelernt. Nachfolgende Tabelle listet weitere Funktionen auf, die solche Detailinformationen ermitteln. Dabei steht `$f` jeweils für einen relativen oder absoluten Pfadnamen zur gewünschten Datei.

Funktion	Bedeutung
`fileatime($f)`	Gibt den Zeitpunkt des letzten Zugriffs auf `$f` als Unix-Zeitstempel zurück.
`filectime($f)`	Gibt den Zeitpunkt der letzten Änderung des Dateizeigers `Inode` auf `$f` als Unix-Zeitstempel zurück.
`filegroup($f)`	Gibt unter Linux/Unix die numerische ID der Primärgruppe von `$f` zurück. Der zugehörige Gruppenname kann mit `posix_getgrgid(filegroup($f))` ermittelt werden.
`fileinode($f)`	Gibt unter Linux/Unix die Inode-Nummer von `$f` zurück.
`filemtime($f)`	Gibt den Zeitpunkt der letzten Änderung von `$f` als Unix-Zeitstempel zurück.

Tabelle 12.93: Funktionen zur Ermittlung von Dateiinformationen

Funktion	Bedeutung
`fileowner($f)`	Gibt unter Linux/Unix die numerische ID des Eigentümers von `$f` zurück. Der zugehörige Benutzername kann mit `posix_getpwuid(fileowner($f))` ermittelt werden.
`fileperms($f)`	Gibt die Zugriffsrechte für `$f` als numerische Summe aus folgenden Einzelwerten zurück: Eigentümer »r«: 1024 Eigentümer »w«: 512 Eigentümer »x«: 256 Gruppe »r«: 64 Gruppe »w«: 32 Gruppe »x«: 16 Rest der Welt »r«: 4 Rest der Welt »w«: 2 Rest der Welt »x«: 1
`filesize($f)`	Gibt Dateigröße von `$f` in Byte zurück.
`filetype($f)`	Gibt den Dateityp von `$f` zurück: fifo, char, dir, block, link, file oder unknown.

Tabelle 12.93: Funktionen zur Ermittlung von Dateiinformationen (Fortsetzung)

Neben diesen Einzelfunktionen gibt es auch die Sammelfunktion `stat()`, welche die verschiedenen Detailinformationen in einem Array zurückgibt. Die Einzelfunktionen sollten dann den Vorzug erhalten, wenn bestimmte Informationen benötigt werden, und die Sammelfunktion dann, wenn alle Informationen auf einmal benötigt werden.

Funktion:	`stat()`
Aufgabe:	Ermittelt Informationen zu einem Verzeichniseintrag.
Parameter:	1. Pfadname des Verzeichniseintrags, relativ aus Sicht des aktuellen Arbeitsverzeichnisses oder absolut.

Rückgabewert:	Ein Array mit folgenden assoziativen Schlüsselnamen:
	`'dev'`: Laufwerk (numerische Angabe)
	`'ino'`: entspricht `fileinode()`
	`'mode'`: entspricht `fileperms()`
	`'nlink'`: Anzahl Links, die auf den Eintrag verweisen
	`'uid'`: entspricht `filegroup()`
	`'gid'`: entspricht `fileowner()`
	`'rdev'`: Laufwerkstyp, wenn Inode-Laufwerk (nur Unix/Linux)
	`'size'`: entspricht `filesize()`
	`'atime'`: entspricht `fileatime()`
	`'mtime'`: entspricht `filemtime()`
	`'ctime'`: entspricht `filectime()`
	`'blksize'`: Blockgröße für das Dateisystem (nur Unix/Linux)
	`'blocks'`: Anzahl zugewiesener Speicherblöcke

Tabelle 12.94: Steckbrief Funktion »stat()«

Datei-Management

Wie Sie Inhalte einzelner Dateien ausgeben oder ändern, wurde bereits in den *Abschnitten 12.4.1* und *12.4.2* beschrieben. Hier geht es darum, wie Dateien kopiert, umbenannt, verschoben und gelöscht werden können. Wie auch bei Verzeichnissen muss das PHP-Script bei allen Operationen mit Dateien ausreichende Lese- oder Schreibrechte besitzen.

Eine Datei kopieren können Sie ganz einfach mit `copy()`, eine Datei löschen mit `unlink()` und eine Datei umbenennen und verschieben mit `rename()`. Die Funktionen sind eigentlich selbsterklärend.

Funktion:	`copy()`
Aufgabe:	Kopiert eine Datei.
Parameter:	1. Zu kopierende Datei, wenn nötig mit absolutem Pfadnamen oder Pfadnamen aus Sicht des PHP-Scripts. Auch URIs auf fremden Servern können adressiert werden, falls in *php.ini* die Direktive *allow_url_fopen* auf *On* oder *1* gesetzt ist. 2. Verzeichnis, in das die Datei kopiert werden soll, und/oder neuer Dateiname, unter dem die Kopie gespeichert werden soll.
Rückgabewert:	`true`, wenn die Operation erfolgreich war, und `false`, wenn nicht.

Tabelle 12.95: Steckbrief Funktion »copy()«

Funktion:	`unlink()`
Aufgabe:	Löscht eine Datei.
Parameter:	1. Zu löschende Datei, wenn nötig mit absolutem Pfadnamen oder Pfadnamen aus Sicht des PHP-Scripts.
Rückgabewert:	`true`, wenn die Operation erfolgreich war, und `false`, wenn nicht.

Tabelle 12.96: Steckbrief Funktion »unlink()«

Funktion:	`rename()`
Aufgabe:	Benennt eine Datei innerhalb des gleichen Verzeichnisses um oder verschiebt sie in ein anderes Verzeichnis.
Parameter:	1. Umzubenennende oder zu verschiebende Datei, wenn nötig mit absolutem Pfadnamen oder Pfadnamen aus Sicht des PHP-Scripts. 2. Verzeichnis, in das die Datei verschoben werden soll, oder neuer Dateiname, unter dem die Datei gespeichert werden soll.
Rückgabewert:	`true`, wenn die Operation erfolgreich war, und `false`, wenn nicht.

Tabelle 12.97: Steckbrief Funktion »rename()«

12.4.7 Session-Funktionen

Eines der Features, die PHP wirklich berühmt gemacht haben, ist das Konzept der Sessions. Dieses Konzept hat es schon vielen Webprogrammierern ermöglicht, benutzerindividuelle Anwendungen zu erstellen, und jeder, der sich mit PHP beschäftigt, wird früher

oder später über das Konzept der Sessions stolpern und anfangen, sich mit den Möglichkeiten von Sessions zu befassen.

Sessions ermöglichen es, Daten einzelner Website-Besucher seitenübergreifend zu speichern, und zwar für den Zeitraum des Besuchs. Es kann sich um beliebige Daten handeln, die ein PHP-Script auch von Seitenaufruf zu Seitenaufruf dynamisch fortführen kann. Ein Shop-System etwa kann mithilfe von Session-Daten die Funktion »Warenkorb« realisieren, sich also seitenübergreifend merken, welche Artikel ein Kunde zum Einkauf vorgemerkt hat. Eine E-Learning-Anwendung oder ein mehrseitiges Anwendungsszenario wie »theoretische Führerscheinprüfung« kann sich merken, welche Übungen oder Fragen ein Anwender bereits absolviert hat, und mit welchem Erfolg. Gerade auch in Verbindung mit Datenbankanbindung können Sessions wertvolle Hilfe leisten und für mehr Performance sorgen. Denn in Session-Daten können möglicherweise Daten festgehalten werden, die sonst bei jedem Seitenwechsel erneut aus einer Datenbank geholt werden müssten, wie z. B. gespeicherte Benutzerprofile.

Der Zeitraum einer Session dauert so lange, wie der Site-Besucher innerhalb eines Timeouts (einstellbar, z. B. 10 Minuten) neue Seiten aufruft bzw. sein Browser mit dem Webserver kommuniziert. Ein PHP-Script kann eine Session allerdings nicht nur explizit starten, sondern auch explizit beenden. Manche Anwendungen starten Sessions beispielsweise nur für registrierte Benutzer nach deren erfolgreichem Login und beenden sie nach einem Logout. Sessions sind jedoch nicht an die Registrierung von Besuchern gebunden.

Welchem Site-Besucher welche Session-Daten zugeordnet sind, wird über eine so genannte Session-ID verwaltet. Beim Anlegen einer neuen Session wird eine eindeutige Session-ID erzeugt. Auf dem Server-Rechner werden die damit verbundenen Session-Daten in Dateien gespeichert, die PHP in einem in der *php.ini* einstellbaren Verzeichnis verwaltet.

Die Session-ID muss der Browser des Site-Besuchers an den Webserver und damit weiter an das aufgerufene PHP-Script übertragen, damit das Script mit den Session-Daten des Benutzers arbeiten kann. Das bedeutet aber, dass die Session-ID zunächst irgendwie vom session-erzeugenden Script über den Webserver zum Browser des Benutzers gelangen muss, damit dieser sie bei weiteren Seitenaufrufen an den Server zurückübertragen kann.

Die beste und sicherste Lösung dafür sind temporäre Cookies. Bei Anwendern, die in ihren Browsern überhaupt keine Cookies akzeptieren, fällt diese Möglichkeit allerdings flach. Eine andere Möglichkeit besteht darin, dass ein PHP-Script im Browser ausgegebene Links zu anderen Projektseiten dynamisch mit einem GET-String versieht, in dem die Session-ID gespeichert ist. Dieser Weg ist jedoch nicht mehr ernsthaft empfehlenswert, da Session-IDs sich in URIs ausspähen lassen und es Angreifern leicht machen, sich als andere Benutzer auszugeben. Je sensibler die Daten sind, die in Sessions gespeichert werden, desto gravierender können die Auswirkungen solcher Angriffe sein. Es genügt beispielsweise schon, wenn eine Anwendung in den Session-Daten eines Anwenders speichert, ob dieser sich erfolgreich als registrierter Benutzer angemeldet hat. Durch einfachen Klau einer entsprechenden Session-ID könnte sich ein Angreifer dann als registrierter Benutzer am System ausgeben.

Aus diesen Gründen sollten Sie, wenn Sie mit Sessions ernsthaft arbeiten wollen, Ihren Seitenbesuchern erklären, warum das Akzeptieren von temporären Cookies im Browser aktiviert sein sollte. Glücklicherweise akzeptieren die Browser per Voreinstellung solche Cookies, d.h. die Mehrheit der Besucher, die sich um ihre Browser-Einstellungen gar nicht kümmert, stellt aus Sicht der Anwendung kein Problem dar. Lediglich eine Minderheit von Anwendern, die Cookies meist ohne genauere Kenntnis von deren Funktionsweise aus Panik verweigern, stellen eine problematische Zielgruppe dar.

Um Session-IDs wenn möglich mit Cookies an den Browser zu übertragen, muss in der *php.ini* die Direktive *session.use_cookies* auf `1` oder `On` stehen (dies entspricht der Default-Einstellung). Um zu verhindern, dass überhaupt ein anderer Übertragungsweg als Cookies gewählt wird, können Sie außerdem *session.use_only_cookies* auf `1` oder `On` stellen (Default ist hier `0` / `Off`). Sollten Sie die *php.ini* nicht editieren können, so können Sie im PHP-Script vor dem Starten einer Session die gewünschten Werte für Ihr Script folgendermaßen festlegen:

```
ini_set("session.use_cookies", "1");
ini_set("session.use_only_cookies", "1");
```

Weiterführender Link:

Sessions mit PHP:
http://develnet.org/26.html

Sessions erzeugen und löschen

Eine gute Variante, Sessions zu starten, besteht in folgenden beiden Funktionsaufrufen:

```
session_name('MyServer_Session');
session_start();
```

Der Aufruf von `session_name()` ist zwar nicht zwingend, doch es gehört zum guten Stil, einer Session einen Namen zu geben. Bei der Namensvergabe gelten die gleichen Regeln wie bei Variablen- oder Funktionsnamen. Der Session-Name erscheint in Cookies und GET-Strings von URIs. Wenn Sie `session_name()` notieren, dann muss dieser Funktionsaufruf vor `session_start()` notiert werden.

Mit `session_start()` wird eine Session initialisiert oder erneut gestartet. Jedes Script, das via URI aufgerufen wird und mit Sessions arbeiten will, muss also diesen Funktionsaufruf enthalten. Wenn noch keine Session für einen aufrufenden Benutzer existiert, wird sie angelegt. Die Session-ID wird mit der nächsten HTTP-Antwort des Webservers an den Browser des Benutzers übertragen. Das PHP-Script kann ab sofort damit beginnen, Session-Daten für den Benutzer zu speichern. Wenn bereits eine Session existiert, stellt `session_start()` den erneuten Bezug zwischen Script und Session-Daten her.

Den Aufruf von `session_start()` müssen Sie innerhalb des Scripts, wenn die Session-ID via Cookies übertragen werden soll, *vor* jeder Ausgabe von HTML-Code oder anderen Inhalten notieren!

Funktion:	`session_name()`
Aufgabe:	Vergibt einen Namen für eine Session.
Parameter:	1. Gewünschter Name.
Rückgabewert:	Gibt den Namen der Session zurück, kann also auch zum Ermitteln des Session-Namens verwendet werden.

Tabelle 12.98: Steckbrief Funktion »session_name()«

Funktion:	`session_start()`
Aufgabe:	Startet eine neue Session oder stellt den Bezug zu einer bereits vorhandenen Session wieder her.
Parameter:	Keine.
Rückgabewert:	`true`

Tabelle 12.99: Steckbrief Funktion »session_start()«

Session-Daten speichern und lesen

Um Session-Daten zu speichern, steht Ihnen der superglobale Array `$_SESSION` zur Verfügung. Auf diesen können Sie überall im Script lesend und schreibend zugreifen.

Um neue Session-Daten zu speichern, weisen Sie einfach einem neuen Schlüsselnamen des Arrays einen gewünschten Wert zu. Beispiel:

```
$_SESSION['timestamp'] = date("d.m.Y, H:i:s");
```

In diesem Beispiel wird in einem Element des $_SESSION-Arrays namens `timestamp` der aktuelle Zeitpunkt gespeichert. So kann sich ein Script beispielsweise merken, wann der letzte Seitenaufruf dieses Benutzers stattgefunden hat.

In vielen Fällen ist es allerdings sinnvoller, gleich mit untergeordneten Arrays zu arbeiten. Beispiel:

```
$_SESSION['user'] = array();
$_SESSION['user']['status'] = 'unauthorized';
$_SESSION['user']['config'] = array();
$_SESSION['user']['config']['style_sheet'] = 'standard.css';
$_SESSION['user']['config']['news_count'] = 20;
$_SESSION['user']['config']['info_level'] = 3;
```

Der Array `$_SESSION` ist also geduldig und bereit, auch stärker strukturierte Daten aufzunehmen.

Zum Auslesen der Daten können Sie mit dem Array `$_SESSION` wie mit jeder anderen Variablen verfahren.

Da Session-Daten letztlich aus einem mehr oder weniger komplexen Array bestehen, mussten sich die Entwickler von PHP etwas einfallen lassen, um die Datenstruktur in Session-Dateien zu speichern. Dabei ist ein interessantes Format herausgekommen, welches

die so genannte Serialisierung beliebig komplexer Datenstrukturen in eine Zeichenkette ermöglicht. Die oben angelegte Struktur mit `$_SESSION['user']...` sieht in einer Session-Datei, die PHP daraus erzeugt, wie folgt aus:

```
user|a:2:{s:6:"status";s:12:"unauthorized";s:6:"config";a:3:{s:11:"style_sheet";
s:12:"standard.css";s:10:"news_count";i:20;s:10:"info_level";i:3;}}
```

Das ist so zu lesen: Es gibt einen Schlüssel `user`. Dieser ist ein Array (`a`) mit `2` Elementen. Sein erstes Element ist ein Variablenname (Zeichenkette `s`) namens `status` (`6` Zeichen) mit dem Wert `unauthorized` (Zeichenkette `s` mit `12` Zeichen). Zweites Element ist eine Variable namens `config` (Zeichenkette `s` mit `6` Zeichen). Diese ist ein Array (`a`) mit `3` Elementen ... usw. Untergeordnete Arrays sind durch geschweifte Klammern markiert.

Da es auch für andere Zusammenhänge nützlich sein kann, Array-Strukturen in eine Zeichenkette zu serialisieren, z. B. um den Inhalt in einem Datenbankfeld zu speichern, bietet PHP dem Programmierer zwei Funktionen an, die aus Array-Strukturen solche serialisierten Zeichenketten erzeugen und umgekehrt aus Zeichenketten in diesem Format wieder die ursprünglichen Array-Strukturen: `serialize()` und `unserialize()`.

Funktion:	`serialize()`
Aufgabe:	Erzeugt aus einer komplexen Array-Struktur eine Zeichenkette, welche die Informationen zur Datenstruktur beibehält.
Parameter:	Zu serialisierende Variable, in der Regel ein assoziativer Array mit Unter-Arrays.
Rückgabewert:	Serialisierte Zeichenkette.

Tabelle 12.100: Steckbrief Funktion »serialize()«

Funktion:	`unserialize()`
Aufgabe:	Stellt aus einer mit `serialize()` serialisierten Variablen die Ursprungsstruktur wieder her. Es empfiehlt sich, den Aufruf dieser Funktion einer Variablen zuzuweisen.
Parameter:	Zeichenkette, welche das serialisierte Datenformat enthält.
Rückgabewert:	Die ursprüngliche Datenstruktur.

Tabelle 12.101: Steckbrief Funktion »unserialize()«

Session-Daten löschen und Sessions beenden

Um alle Session-Daten, die im superglobalen Array `$_SESSIONS` gespeichert sind, auf einmal zu löschen, genügt es, zu notieren:

```
unset($_SESSION);
```

Die Funktion `unset()`, die eine Variable löscht, ist also auch auf `$_SESSION` anwendbar.

Damit existieren jedoch noch die Session-ID und die Session-Datei auf dem Server-Rechner.

Zuerst sollten Sie das Cookie löschen. Dazu bietet sich folgender Code an:

```
if (isset($_COOKIE[session_name()])) {
   setcookie(session_name(), '', time()-42000, '/');
}
```

Abhängig davon, ob im superglobalen Array `$_COOKIES` noch ein Cookie für die mit `session_name()` ermittelte Session existiert, wird das Cookie gelöscht, indem sein Wert auf eine leere Zeichenkette gesetzt wird und sein Verfallszeitpunkt in die Vergangenheit verlegt wird. Bei der nächsten Antwort des Webservers an den Browser wird das so veränderte Cookie übertragen, was den Browser zum sofortigen Löschen des Cookies veranlassen sollte.

Zuletzt können Sie noch die auf dem Server-Rechner als Datei gespeicherten Session-Daten durch folgenden Funktionsaufruf löschen:

```
session_destroy();
```

Funktion:	`session_destroy()`
Aufgabe:	Löscht Session-Daten auf dem Server-Rechner.
Parameter:	Keine.
Rückgabewert:	`true`, wenn die Daten gelöscht werden konnten, und `false`, wenn nicht.

Tabelle 12.102: Steckbrief Funktion »session_destroy()«

Kleines Testbeispiel

Ein kleines, ganz einfaches Beispiel soll noch einmal die Session-Funktionalität erläutern. Es besteht aus zwei Scripts *test.php* und *test2.php*.

Der Code von *test.php* lautet:

```
<?php
session_name('test');
session_start();
$_SESSION['called_by'] = "test.php";
echo "<a href=\"test2.php\">nächste Seite</a>";
?>
```

Der Code von *test2.php* lautet:

```
<?php
session_name('test');
session_start();
echo $_SESSION['called_by'];
?>
```

In *test.php* wird eine Session initialisiert. Im $_SESSION-Array wird ein Schlüssel namens `called_by` mit dem Dateinamen des aktuellen Scripts belegt. Ein HTML-Link erlaubt den Aufruf von *test2.php*.

In *test2.php* wird die Session wieder aufgegriffen. Damit ist der Zugriff auf die in `$_SESSION` gespeicherten Daten möglich. Das Script im Beispiel kann also auf `$_SESSION['called_by']` zugreifen und ausgeben, von welchem anderen Script es aufgerufen wurde.

Weitere Session-Funktionen

Eine Reihe weiterer Session-Funktionen hilft, eventuelle Probleme bei Sessions zu beheben oder die Session-Funktionalität zu optimieren.

Mit `session_id()` kann das Script jederzeit die bei `session_start()` erzeugte oder benutzte Session-ID ermitteln.

Funktion:	`session_id()`
Aufgabe:	Ermittelt oder ändert die aktuelle Session-ID
Parameter:	1. (optional): Session-ID. Wenn eine andere als die aktuelle ID übergeben wird, muss anschließend `session_start()` neu aufgerufen werden, um mit der geänderten Session-ID zu arbeiten. Um nur die Session-ID zu ermitteln, muss `session_id()` ohne Parameter aufgerufen werden.
Rückgabewert:	Gibt die Session-ID zurück.

Tabelle 12.103: Steckbrief Funktion »session_id()«

Um sich über das Session-Cookie zu informieren, kann ein Script die Funktion `session_get_cookie_params()` aufrufen. Die Informationen können als Basis dazu dienen, das Cookie in veränderter Form mit `setcookie()`neu zu setzen.

Funktion:	`session_get_cookie_params()`
Aufgabe:	Ermittelt die Parameter des Session-Cookies. Der Funktionsaufruf sollte einer Variablen zugewiesen werden. Diese enthält anschließend den Array, den die Funktion zurückgibt, und über diese Variable kann auf die Schlüssel des Arrays zugegriffen werden.
Parameter:	keine
Rückgabewert:	Assoziativer Array mit den Parametern des Session-Cookies. Folgende Schlüssel sind enthalten: `'lifetime'`: Lebensdauer des Cookies in Sekunden. `'path'`: Pfad, für den das Cookie gültig ist. `'domain'`: Domain, für die das Cookie gültig ist. `'secure'`: `true`, wenn das Cookie nur über HTTPS-Verbindungen gesendet wird.

Tabelle 12.104: Steckbrief Funktion »session_get_cookie_params()«

Falls Sie in Ihrem Webprojekt Frames verwenden und in mehreren Frame-Fenstern PHP-Scripts haben, die auf die gleiche Session zugreifen, empfiehlt es sich, innerhalb jedes Scripts so frühzeitig wie möglich alle gewünschten Session-Daten in `$_SESSION` zu speichern und anschließend `session_write_close()` aufzurufen. Damit beenden Sie die Ses-

sion im aktuellen Scriptablauf. Die Session-Daten werden aber gespeichert. Hintergrund ist, dass bei Frames mehrere Scripts auf die gleiche Session-Datei zugreifen. Damit sie sich dabei nicht gegenseitig ins Gehege kommen, sorgt PHP dafür, dass der Webserver die Frame-Inhalte nacheinander zum Browser sendet. Die Frames werden also weniger zügig geladen. Durch die beschriebene Verwendung von `session_write_close()` können Sie den Ladevorgang beschleunigen.

Funktion:	`session_write_close()`
Aufgabe:	Beendet die Arbeit mit der Session im aktuellen Scriptlauf und speichert die Session-Daten für die nächste Verwendung.
Parameter:	Keine.
Rückgabewert:	Keiner.

Tabelle 12.105: Steckbrief Funktion »session_write_close()«

Damit Sessions ordentlich funktionieren, dürfen Browser oder in der Internetverbindung zwischengeschaltete Proxy-Server keine Daten aus dem Cache holen. Sollten Sessions einmal nicht funktionieren, veraltete Daten liefern usw., müssen Sie das Cache-Verhalten überprüfen und gegebenenfalls ändern. Dazu dient die Funktion `session_cache_limiter()`. Ein Beispiel:

```
session_name('SHOP_SESSION');
session_cache_limiter('private, must-revalidate');
session_start();
```

Wird die Funktion `session_cache_limiter()` ohne Parameter aufgerufen, liefert sie einfach die aktuellen Cache-Verhaltenseinstellungen, die der Webserver an den aufrufenden Browser und damit auch an Proxy-Server sendet. Um Cache-Verhaltensweisen zu ändern, müssen Sie einen Parameter wie im Beispiel übergeben. Wichtig ist in diesem Fall, dass die Funktion *vor* `session_start()` aufgerufen wird.

Funktion:	`session_cache_limiter()`
Aufgabe:	Ermittelt die Cache-Verhaltensanweisungen, die PHP über den Webserver bei Verwendung von Sessions automatisch an den Browser sendet. Wenn kein Parameter übergeben wird, sollte der Funktionsaufruf einer Variablen zugewiesen werden. Diese enthält anschließend die Einstellungen als Zeichenkette.
Parameter:	1. (optional): Zeichenkette, die neue Cache-Verhaltensanweisungen definiert. Folgende Schlüsselwörter können z. B. vorkommen (und ansonsten alles, was HTTP 1.1 für den Header ***Cache-Control*** erlaubt): `'private'`: Browser-Cache darf benutzt werden, Proxy-Caches nicht. `'public'`: Sowohl Browser-Cache als auch Proxy-Caches dürfen benutzt werden. `'nocache'`: Für den HTML-Code darf kein Cache benutzt werden, wohl aber für referenzierte Dateien wie Grafiken, CSS, JavaScript usw.

Tabelle 12.106: Steckbrief Funktion »session_cache_limiter()«

Parameter:	`'nostore'`: Weder für den HTML-Code noch für referenzierte Dateien darf irgendein Cache benutzt werden. `'must-revalidate'`: Die Cache-Einstellungen müssen bei jedem Aufruf neu überprüft werden.
Rückgabewert:	Zeichenkette mit aktuell eingestellten Cache-Verhaltensanweisungen.

Tabelle 12.106: Steckbrief Funktion »session_cache_limiter()« (Fortsetzung)

Solange Sie das Cachen in irgendeiner Form erlauben, also nicht `nocache` oder `nostore` veranlassen, können Sie zusätzlich einstellen, wie lange die Inhalte maximal im Cache-Speicher vorgehalten werden dürfen, bevor sie dort gelöscht und von der cache-verwaltenden Instanz (Browser, Proxy) nicht mehr verwendet werden dürfen. Die Verfallszeit von Session-Seiten im Cache können Sie mit `session_cache_expire()` ermitteln und beeinflussen.

Funktion:	`session_cache_expire()`
Aufgabe:	Ermittelt die Cache-Verfallszeit, die PHP über den Webserver bei Verwendung von Sessions automatisch an den Browser sendet. Wenn kein Parameter übergeben wird, sollte der Funktionsaufruf einer Variablen zugewiesen werden. Diese enthält anschließend die Cache-Verfallszeit als Minutenzahl.
Parameter:	1. (optional): Zahl, welche die neue Cache-Verfallszeit in Minuten definiert.
Rückgabewert:	Integer mit der Minutenzahl, wie lange Session-Seiten im Cache bleiben dürfen.

Tabelle 12.107: Steckbrief Funktion »session_cache_expire()«

12.4.8 Starten externer Programme und Kommandos

PHP eignet sich unter anderem dazu, webbasierte Verwaltungsoberflächen für die Verwaltung von kommandogesteuerten Programmen oder für die Shell-Ebene anzubieten. Viele Hosting-Provider stellen solche webbasierten Verwaltungsoberflächen bereit. Um eine solche Funktionalität zu realisieren, kann PHP Programme und Kommandos ausführen und auf Wunsch deren Ausgaben »auffangen«.

Aber auch für die Verwaltung einer Site kann die Möglichkeit des Ausführens von Programmen oder Kommandos auf dem Server-Rechner nützlich sein. So ist ein interner Site-Bereich denkbar, den Benutzer nur durch Authentifizierung aufrufen können und in dem z.B. Datensicherungen, Aufräumaktionen, Statistikfunktionen usw. angestoßen werden können.

Sinnvolle Einsatzzwecke für den Start anderer Programme, Scripts oder Kommandos sind also leicht erkennbar. Problematischer wird es bei der Sicherheit. Vor allem, wenn Programmnamen, Parameter usw. direkt aus Formulardaten geholt werden, droht Gefahr. Erfahrene Hacker kennen Umwege, um über scheinbar harmlose Parameter Drittprogramme auszuführen, die wiederum großen Schaden anrichten, Daten löschen und das System lahmlegen können. Seien Sie daher extrem vorsichtig mit Ausführungsdaten, die Sie über HTML-Formulare ermitteln. Besser ist es, nur Hyperlinks anzubieten, die ein PHP-Script aufrufen, das dem Benutzer sonst keine Chance lässt, Einfluss darauf zu nehmen, was genau und mit welchen Parametern ausgeführt werden soll.

Grundsätzlich gilt, dass alle Funktionen zum Ausführen von Programmen, Kommandos, Scripts usw. natürlich plattformabhängig sind. Außerdem muss das PHP-Script über die nötigen Ausführrechte verfügen.

Funktionen zur Programmausführung

PHP unterstützt den so genannten Backtick-Operator. Da dieser aber eine Zumutung beim Eintippen darstellt und optisch auch sehr hässlich ist, empfehlen wir die Verwendung der Funktion `shell_execute()`. Ein Beispiel:

```
$output = shell_exec('ps -a');
echo "<pre>$output</pre>";
```

Das Beispiel ermittelt durch Aufruf des Linux-Kommandos `ps -a` aktuelle Prozesse auf dem Server-Rechner. Die gesamte Ausgabe, die das Kommando erzeugt, wird durch Zuweisung an die Variable `$output` in dieser gespeichert.

Funktion:	`shell_exec()`
Aufgabe:	Startet einen Programm-, Script- oder Kommandoaufruf von der Shell und gibt eventuelle Ausgaben zurück.
Parameter:	1. Vollständiges Kommando als Zeichenkette, gegebenenfalls mit Parametern usw.
Rückgabewert:	stdout-Ausgaben des aufgerufenen Programms, Kommandos oder Scripts.

Tabelle 12.108: Steckbrief Funktion »shell_exec()«

Es können nur Shell-Kommandos ausgeführt werden, die von der benutzten Shell oder dem benutzten Kommandointerpreter zur Verfügung gestellt werden. Maßgeblich ist die Shell, die dem Benutzer, unter dem das PHP-Script läuft, zugewiesen ist. Wenn PHP als Apache-Modul läuft, ist dies die Benutzerkennung des Apache.

Ferner gibt es die Funktionen `system()` und `exec()`. Der Unterschied zu `shell_exec()` besteht darin, dass diese beiden Funktionen Programme oder Scripts ohne Umweg über eine Shell starten (shell-interne Kommandos können also nicht direkt ausgeführt werden, sondern nur durch Ausführen der Shell als Programm). Der Unterschied zwischen den beiden Funktionen besteht darin, dass `system()` Standardausgaben des aufgerufenen Scripts, Programms oder Kommandos direkt in PHP ausgibt, während `exec()` den Befehl nur ausführt, ohne seine Ausgabe wiederzugeben.

Ein Beispiel für `system()`:

```
echo "<pre>";
system('ps -a');
echo "</pre>";
```

Diese Anweisungsfolge liefert das gleiche Ergebnis wie obiges Beispiel zu `shell_execute()`. Der Unterschied ist, dass `system()` nicht den gesamten Kommando-Output zurückgibt, sondern ihn direkt ausgibt.

Ein Beispiel für `exec()`:

```
exec("perl /usr/bin/www/backup.pl");
```

Damit wird das Perl-Script gestartet, ohne dass irgendetwas an Output geliefert wird.

Funktion:	`system()`
Aufgabe:	Startet ein Programm oder Script und gibt eventuelle Ausgaben selber direkt aus.
Parameter:	1. Auszuführendes Programm oder Script als Zeichenkette, gegebenenfalls mit Parametern usw. 2. (optional) Variable, in welcher der Returnwert des aufgerufenen Scripts oder Programms aufgefangen wird.
Rückgabewert:	Gibt bei Erfolg die letzte Zeile der Befehlsausgabe zurück oder `false`, wenn ein Fehler auftrat.

Tabelle 12.109: Steckbrief Funktion »system()«

Funktion:	`exec()`
Aufgabe:	Startet ein Programm oder Script, ohne dessen Ausgaben auszugeben.
Parameter:	1. Auszuführendes Programm oder Script als Zeichenkette, gegebenenfalls mit Parametern usw. 2. (optional) Variable, in welcher die Ausgaben des aufgerufenen Scripts oder Programms aufgefangen werden. Die Variable ist anschließend ein Array, bei dem jede Zeile eine Ausgabezeile der Programmausgabe darstellt. 3. (optional) Variable, in welcher der Returnwert des aufgerufenen Scripts oder Programms aufgefangen wird.
Rückgabewert:	Gibt die letzte Zeile der Befehlsausgabe zurück.

Tabelle 12.110: Steckbrief Funktion »exec()«

Sowohl bei `system()` als auch bei `exec()` wartet das PHP-Script mit der Abarbeitung weiterer Anweisungen, bis die Ausführung des anderen Programms oder Scripts beendet ist. Gleichen Sie daher die Laufzeit von aufgerufenen Batch-Prozessen mit dem in ***php.ini*** eingestellten Wert für ***max_execution_time*** ab!

Zu `system()`gibt es auch noch eine Alternative, die syntaktisch ähnlich ist: `passthru()`. Verwenden Sie diese Funktion, wenn das aufgerufene Programm oder Script binäre Daten liefert, etwa Daten einer Grafik oder eines Flash-Movies. Die Funktion gibt die vom Programm ausgegebenen Daten selber direkt aus. Bei Binärdaten sollten Sie vor einem Aufruf dieser Funktion so etwas wie `header("Content-type: image/gif")` notieren.

Funktion:	`passthru()`
Aufgabe:	Startet ein Programm oder Script und gibt eventuelle Ausgaben selber direkt aus.

Tabelle 12.111: Steckbrief Funktion »passthru()«

Parameter:	1. Auszuführendes Programm oder Script als Zeichenkette, gegebenenfalls mit Parametern usw. 2. (optional) Variable, in welcher der Returnwert des aufgerufenen Scripts oder Programms aufgefangen wird.
Rückgabewert:	Keiner.

Tabelle 12.111: Steckbrief Funktion »passthru()« (Fortsetzung)

Falls Sie Daten für die Ausführung eines Shell-Kommandos aus HTML-Formularen einlesen, sollten Sie unbedingt die Funktion `escapeshellcmd()` anwenden. Dadurch verhindern Sie, dass durch Benutzereingaben ein willkürliches anderes Shell-Kommando ausgeführt werden kann. Angenommen, Sie möchten es dem Anwender ermöglichen, ein Verzeichnis unterhalb der Document Root als Verzeichnislisting anzeigen zu lassen. Das gewünschte Verzeichnis soll der Anwender in einem Formularfeld eingeben. Sie könnten nun notieren:

```
$output = shell_exec('ls -l '.$_POST['directory']);
echo "<pre>$output</pre>";
```

Dies ist jedoch brandgefährlich, da die Shell es erlaubt, mehrere Kommandos in Folge durch Semikolon getrennt zu notieren. Gibt ein Anwender also ein:

```
pics; rm -fR *
```

so würde die Ausführung ohne weitere Rückfrage einen ganzen Verzeichnisbaum löschen, da `shell_exec()` brav an die Shell übergeben würde: `ls -l pics; rm -fR *`. Durch folgende Maßnahme verhindern Sie derartige Desaster:

```
$output = shell_exec(escapeshellcmd('ls -l '.$_POST['directory'] ));
echo "<pre>$output</pre>";
```

Die Funktion `escapeshellcmd()` maskiert alle Zeichen, die zu gefährlichen Seitenausführungen von anderen Kommandos führen könnten.

Funktion:	`escapeshellcmd()`
Aufgabe:	Maskiert in einem Shell-Kommando oder Programmaufruf alle Zeichen, die das Ausführen nicht erwünschter anderer Programme verursachen würden.
Parameter:	1. Auszuführendes Programm oder Script als Zeichenkette, gegebenenfalls mit Parametern usw.
Rückgabewert:	Gibt den mit Maskierungszeichen versehenen Befehl zurück. Dieser kann in dieser Form für `shell_exec()`, `exec()`, `system()` oder `passthru()` verwendet werden.

Tabelle 12.112: Steckbrief Funktion »escapeshellcmd()«

12.4.9 Diverse Funktionen

Unsere Rundreise durch die Funktionen von PHP hat Ihnen ein Arsenal an Funktionen vermittelt, die für die alltägliche Programmierarbeit wichtig bis unerlässlich sind. Dane-

ben bietet PHP aber auch noch etliche Funktionen an, die bestimmte andere Softwareprodukte unterstützen, wie z.B. Funktionen für verschiedene Datenbanksysteme – die MySQL-Funktionen von PHP werden wir noch kennen lernen. Aber auch andere, allgemeine Funktionen gehören dazu. Ein paar wichtige davon stellen wir noch kurz vor.

Funktionen für beliebig genaue Rechenergebnisse

Diese Funktionen erwarten Zahlen als Input, liefern jedoch Zeichenketten als Ergebnis. Mittels eines Binary Calculator erlaubt PHP, auf dieser Ebene zu rechnen. Wenn es aus technischen oder theoretischen Gründen von Bedeutung ist, extrem genaue Ergebnisse zu erhalten, die das Fließkommaprinzip nicht schafft, dann helfen diese Funktionen weiter:

Funktion	Bedeutung
`bcadd($x, $y)`	Addiert `$x` und `$y` und gibt das Ergebnis als Zeichenkette zurück. Als dritter Parameter kann optional die gewünschte Anzahl Nachkommastellen übergeben werden.
`bcsub($x, $y)`	Subtrahiert `$y` von `$x` und gibt das Ergebnis als Zeichenkette zurück. Als dritter Parameter kann optional die gewünschte Anzahl Nachkommastellen übergeben werden.
`bcmul($x, $y)`	Multipliziert `$x` und `$y` und gibt das Ergebnis als Zeichenkette zurück. Als dritter Parameter kann optional die gewünschte Anzahl Nachkommastellen übergeben werden.
`bcdiv($x, $y)`	Dividiert `$x` durch `$y` und gibt das Ergebnis als Zeichenkette zurück. Als dritter Parameter kann optional die gewünschte Anzahl Nachkommastellen übergeben werden.
`bcpow($x, $y)`	Berechnet `$x` hoch `$y` und gibt das Ergebnis als Zeichenkette zurück. Als dritter Parameter kann optional die gewünschte Anzahl Nachkommastellen übergeben werden.
`bcsqrt($x)`	Berechnet die Quadratwurzel von `$x` und gibt das Ergebnis als Zeichenkette zurück. Als dritter Parameter kann optional die gewünschte Anzahl Nachkommastellen übergeben werden.

Tabelle 12.113: Funktionen für genaue Rechenergebnisse

Funktionen für Klassen und Funktionen

Die nachfolgenden Funktionen können helfen, Aufruffehler im Script zu vermeiden.

Funktion	Bedeutung
`class_exists($s)`	Ermittelt, ob eine Klasse mit dem als Zeichenkette zu übergebenden Namen `$s` definiert ist. Gibt `true` zurück, wenn ja, und `false`, wenn nein.
`get_class_methods($s)`	Ermittelt alle Methoden, die zu einer Klasse mit dem als Zeichenkette zu übergebenden Namen `$s` gehören. Gibt einen Array mit den Methodennamen zurück.

Tabelle 12.114: Funktionen für Klassen und Funktionen

Funktion	Bedeutung
`get_class_vars($s)`	Ermittelt alle Klassenvariablen, die zu einer Klasse mit dem als Zeichenkette zu übergebenden Namen `$s` gehören. Gibt einen Array mit den Variablennamen zurück.
`get_declared_classes()`	Ermittelt alle im Script definierten Klassen. Gibt einen Array mit den Klassennamen zurück.
`get_object_vars($o)`	Ermittelt alle aktuell in einer Objektvariablen `$o` gespeicherten Eigenschaften mit Wert. Gibt einen assoziativen Array mit den Eigenschaftsnamen und Werten zurück.
`method_exists($o, $s)`	Ermittelt, ob in über die Objektvariable `$o` eine Methode mit Namen `$s` (Zeichenkette) aufrufbar ist. Gibt `true` zurück, wenn ja, und `false`, wenn nein.
`function_exists($s)`	Ermittelt, ob eine Funktion mit dem als Zeichenkette zu übergebenden Namen `$s` definiert ist. Anwendbar auch, um abzufragen, ob PHP-Funktionen in der aktuell verwendeten PHP-Version verfügbar sind. Gibt `true` zurück, wenn ja, und `false`, wenn nein.

Tabelle 12.114: Funktionen für Klassen und Funktionen (Fortsetzung)

Funktionen für die Datenauswertung

Wie Sie etwa Inhalte aus HTML-Formularen mithilfe von regulären Ausdrücken überprüfen können, wurde bereits beschrieben. Es gibt jedoch diverse praktische PHP-Funktionen, die Standardüberprüfungen abnehmen. Alle diese Funktionen sind so konstruiert, dass sie `true` oder `false` zurückgeben. Damit eignen sich ihre Aufrufe hervorragend, um in if-Abfragen als Bedingung zu dienen, also `if(Funktionsname())`.

Funktion	Bedeutung
`ctype_alnum($s)`	Gibt `true` zurück, wenn die Zeichenkette `$s` ausschließlich aus Klein- und Großbuchstaben und/oder Ziffern besteht, ansonsten `false`.
`ctype_alpha($s)`	Gibt `true` zurück, wenn die Zeichenkette `$s` ausschließlich aus Klein- und Großbuchstaben besteht, ansonsten `false`.
`ctype_digit($s)`	Gibt `true` zurück, wenn die Zeichenkette `$s` ausschließlich aus Ziffern besteht, ansonsten `false`.
`ctype_print($s)`	Gibt `true` zurück, wenn die Zeichenkette `$s` ausschließlich aus druckbaren Zeichen besteht (keine Steuerzeichen, allenfalls Leerzeichen), ansonsten `false`.
`ctype_lower($s)`	Gibt `true` zurück, wenn die Zeichenkette `$s` ausschließlich aus Kleinbuchstaben besteht, ansonsten `false`.
`ctype_upper($s)`	Gibt `true` zurück, wenn die Zeichenkette `$s` ausschließlich aus Großbuchstaben besteht, ansonsten `false`.
`ctype_xdigit($s)`	Gibt `true` zurück, wenn die Zeichenkette `$s` ausschließlich aus Hexadezimalziffern besteht, ansonsten `false`.

Tabelle 12.115: Funktionen für die Datenauswertung

Funktionen für die PHP-Optimierung

Jede Direktive der zentralen Konfigurationsdatei *php.ini* kann ein Script für sich selber durch Aufruf der Funktion `ini_set()` ändern. Ein Beispiel:

```
ini_set("memory_limit", "16M");
ini_set("display_errors", "off")
```

Im Beispiel erlaubt sich das Script, bis zu 16 Mbyte Arbeitsspeicher zu reservieren (Standard ist meistens 8 Mbyte). Außerdem sollen alle PHP-Fehlermeldungen unterdrückt werden.

Funktion:	`ini_set()`
Aufgabe:	Ändert eine PHP-Konfigurationsdirektive aus *php.ini* für das aktuelle Script.
Parameter:	1. Name der Direktive wie in *php.ini* vorhanden. 2. Gewünschter Wert.
Rückgabewert:	Gibt bei Erfolg den alten Wert zurück oder `false`, wenn ein Fehler auftrat.

Tabelle 12.116: Steckbrief Funktion »ini_set()«

Nachfolgend weitere Funktionen zum Ermitteln von PHP-Umgebungsbedingungen oder zum Ändern derselben:

Funktion	Bedeutung
`ini_get($s)`	Gibt den Wert zur Direktive `$s` (Zeichenkette) aus *php.ini* zurück.
`ini_get_all()`	Gibt alle Einträge aus *php.ini* in einem assoziativen Array zurück. Schlüsselnamen sind die Namen der Direktiven.
`ini_restore($s)`	Ersetzt den Wert der Direktive $s (Zeichenkette), der z. B. durch `ini_set()` gesetzt wurde, wieder durch den Wert der Originalkonfiguration.
`memory_get_usage()`	Ermittelt die Anzahl Bytes, die PHP für das Script aktuell im Arbeitsspeicher reserviert hat.

Tabelle 12.117: Funktionen für die PHP-Umgebung

13 MySQL-Datenbanken und PHP

Zuverlässiges Management von wichtigen Daten und schnelles Information-Retrieval sind für professionelle Webprojekte unerlässlich. In einem Shop-System etwa muss über einen langen Zeitraum nachvollziehbar sein, welcher Kunde wann was bestellt hat und welche Abwicklungsstadien ein Geschäft durchlaufen hat. In einer Community müssen die Stammdaten angemeldeter Mitglieder und ihre individuelle Site-Konfiguration gespeichert und bei Abruf schnell verfügbar sein.

Für all diese und viele andere Zwecke gibt es *Datenbank-Management-Systeme* (DBMS). MySQL ist ein solches System. Es gibt diverse Konkurrenzprodukte, die zum Teil wesentlich umfangreicher oder schneller sind oder noch mehr Datensicherheit bieten. Dennoch hat MySQL zumindest im Webbereich eine klare Führungsrolle unter den DBM-Systemen übernommen. Vermutlich liegt es an der richtigen Mischung zwischen Zuverlässigkeit und Performance einerseits und Handhabbarkeit und Überschaubarkeit andererseits. Dazu kommt natürlich, dass es sich wie bei Apache, PHP, Linux usw. um OpenSource handelt, auch wenn es nicht von einer Community, sondern von einer Firma entwickelt und vertrieben wird.

Solange es sich nicht um sehr große Projekte handelt, bei denen Spezialisten für entsprechend verteilte Aufgaben zum Einsatz kommen, ist es häufig so, dass Webdesigner, PHP-Programmierer und Datenbankzuständiger ein und dieselbe Person sind. Dadurch ist MySQL für viele Menschen das erste DBMS, mit dem sie ernsthaft in Berührung kommen. Umso wichtiger ist es, nicht nur ein paar PHP-Funktionen zu kennen, über die sich auf MySQL zugreifen lässt, sondern daneben auch ein Grundverständnis für die Arbeitsweise von DBM-Systemen zu entwickeln.

13.1 Basiswissen Datenbanken und MySQL

Ein DBMS hat folgende Aufgabenbereiche:

- Es ermöglicht die anwendungsunabhängige Aufbewahrung von Daten.
- Es erlaubt die Administration von Benutzern sowie die Festlegung, welcher Benutzer welche Daten sehen oder bearbeiten darf.
- Es sorgt für die Integrität und Konsistenz gespeicherter Daten im Mehrbenutzerbetrieb.
- Es ermöglicht den schnellen Zugriff auf gespeicherte Daten.

13.1.1 DBM-Systeme und Datenbanken

Etwas irreführend wird häufig die Bezeichnung »Datenbank« mit »DBMS« gleich gesetzt. Eine »Datenbank« ist aus Sicht des DBMS lediglich eine »Anwendung«, so wie aus Sicht von MS Word eine .doc-Datei eine »Anwendungsdatei« ist. Eine einzelne Datenbank kann aus vielen Dateien bestehen – Datendateien für einzelne »Tabellen«, Indexdateien, Formatdateien usw.

Hierarchische, relationale und objektorientierte DBM-Systeme

DBM-Systeme haben schon eine lange Geschichte. Ältere Systeme arbeiteten mit einem hierarchischen Datenmodell. Wenn beispielsweise Kunden und Aufträge zu verwalten waren, dann wurden »unter« jedem Kunden alle seine Aufträge gespeichert. Der Zugriff auf einen bestimmten Auftrag war nur über den zugehörigen Kunden möglich.

Dieses Modell hat sich als recht starr erwiesen. Einen großen Durchbruch bedeuteten deshalb die relationalen DBM-Systeme. Um beim Beispiel zu bleiben, konnten nun einfach Kunden und Aufträge getrennt gespeichert werden. Über ein Datenfeld im Datensatz für einen Auftrag wurde die Verknüpfung zu dem Datensatz in den Kundenstammdaten hergestellt. Heute bilden relationale DBMS die Mehrheit. Auch MySQL gehört zu diesem Typ.

Ein neuerer Ansatz sind objektorientierte DBMS. Anstelle von Datensätzen wird hierbei wie in der objektorientierten Programmierung mit Objekten gearbeitet und anstelle von Datensatzdefinitionen mit Klassen. Klassen können wie in der objektorientierten Programmierung Ableitungshierarchien bilden. Felder können private, protected oder public sein. Mithilfe von Script- oder Programmiersprachen lassen sich auch Methoden definieren.

Bekannte DBM-Systeme

Die heute bekanntesten DBM-Systeme gehören allesamt zum Typ der relationalen Datenbanken. Bekannte Systeme sind DB/2, Oracle, Informix, Ingres, dBase oder MS Access. Im Gegensatz zu deren Geschichte ist die von MySQL noch recht jung. Das erste Release von MySQL geht ins Jahr 1994 zurück, doch der Durchbruch gelang erst nach der Jahrtausendwende. Seitdem hat sich MySQL im Bereich der Webanwendungen zum bei weitem populärsten DBMS entwickelt. Den Erfolg verdankt dieses DBMS nicht zuletzt der Tatsache, dass es, ähnlich wie PHP, Einsteigern wenig Hürden entgegenstellt.

Der heute größte Konkurrent für MySQL unter Webprogrammierern ist PostgreSQL (auszusprechen »postgres-q-l«). PostgreSQL ist wie MySQL OpenSource, jedoch etwas ambitionierter und dafür nicht ganz so leicht zugänglich wie MySQL.

13.1.2 Client-Server-Struktur von Datenbanken

Ein DBMS wird in aller Regel in einer Mehrbenutzerumgebung eingesetzt. Deshalb entspricht die Architektur eines DBMS normalerweise dem Client-Server-Modell. Ein DBM-

Daemon (Server) wird im Arbeitsspeicher eines vernetzten Rechners gehalten und erhält über einen Port (bei TCP/IP) oder eine vergleichbare Schnittstelle Anfragen von Clients.

So verhält es sich auch bei MySQL. Ein Client-Prozess stellt eine Verbindung zum DBMS, also zu dessen laufendem Server her. Üblicherweise besteht der erste Kommunikationsaustausch darin, dass sich der Client-Prozess als Benutzer am DBMS anmelden muss. Ähnlich wie unter Linux oder anderen mehrbenutzerfähigen Betriebssystemen gibt es bei MySQL nach einer Neuinstallation zunächst nur einen Benutzer – den Administrator, der das gesamte System einrichten kann. Er kann auch andere Benutzer anlegen und ihre Zugriffsrechte festlegen. Wenn MySQL für Webanwendungen eingesetzt wird, sollte auf jeden Fall ein Benutzerkonto für den Webserver eingerichtet werden. PHP-Scripts etwa, die als Apache-Modul laufen, können sich dann mit der Benutzerkennung des Webservers beim MySQL-Server anmelden.

Socketverbindungen und Protokolle

DBMS-Client-Prozesse können vom DBMS-Server Daten aus Datenbanken anfordern oder Daten in eine Datenbank neu einfügen oder zurückschreiben, Daten aus einer Datenbank löschen, neue Datenbanken anlegen und strukturieren, Datenbanken optimieren, Daten sichern und vieles mehr. Um das alles zu bewerkstelligen, ist eine ausgereifte Kommandosyntax erforderlich. Die Aufgabe der DBM-spezifischen Kommunikation zwischen Client und Server übernehmen die so genannten Transformationssprachen, von denen heute eigentlich nur noch eine von Bedeutung und der De-facto-Standard ist, nämlich SQL.

Zur Client-Server-Verbindung gehört aber noch mehr. In einer TCP/IP-Umgebung, wie sie in unserem Zusammenhang gegeben ist, lauscht der MySQL-Server an einer bestimmten Portnummer (per Default an Port 3306). Daneben gibt es, ähnlich wie bei der Kommunikation zwischen Browser und Webserver über HTTP, ein Protokoll auf Anwendungsebene. Darin werden Authentifizierungs- und Verbindungsinformationen sowie Informationen zu übertragenen Daten transportiert.

In diesem Buch wird vermittelt, wie Sie über PHP mit einem MySQL-Server kommunizieren können. PHP oder genauer gesagt der Webserver-Prozess, zu dem ein laufendes PHP-Script gehört, fungiert dabei als MySQL-Client. PHP verfügt bereits über ein Set an fertigen Funktionen, welche das MySQL-Protokoll unterstützen und in der Lage sind, TCP/IP-Verbindungen zu MySQL-Servern zu errichten. Um Details auf Protokollebene müssen Sie sich daher nicht weiter kümmern.

13.1.3 Die Transformationssprache SQL

Mit SQL müssen Sie sich in jedem Fall auseinander setzen, wenn Sie eine PHP- und MySQL-basierte Webanwendung entwickeln. PHP nimmt Ihnen zwar die Kommunikation auf Protokollschicht ab, doch was Sie von MySQL überhaupt begehren, müssen Sie innerhalb Ihres PHP-Codes mithilfe von SQL formulieren.

SQL (*Structured Query Language*, zu Deutsch: *Strukturierte Abfragesprache*) erlaubt das Formulieren von Such-, Bearbeitungs- und Verwaltungskommandos für relationale Daten-

banken. Es handelt sich um einen DBMS-unabhängigen Standard, dessen Geschichte bis in die 70er Jahre zurückreicht, als in den IBM-Forschungslabors zunächst eine Sprache namens SEQUEL entwickelt wurde. Seit 1982 kümmert sich das ANSI (*American National Standards Institute*) um die Standardisierung von SQL. Seit 1987 ist SQL auch ISO-Standard.

Fast alle bekannten relationalen DBMS unterstützen SQL, natürlich auch MySQL, wo bereits der Name darauf hinweist. Aber auch ganz andere Systeme, wie DB/2, Informix, dBase (IV), InterBase oder MS Access unterstützen SQL. Die Implementierungen sind leider nicht immer völlig identisch. Ähnlich wie Browser Sprachen wie HTML oder CSS nicht hundertprozentig vollständig und einheitlich interpretieren, so interpretieren DBMS auch SQL nicht vollständig und identisch. Dennoch ist SQL ähnlich segensreich wie HTML/CSS, indem es eine DBMS-unabhängige Kommandosyntax zur Verfügung stellt.

SQL-Statements

SQL ist eine so genannte 4th Generation Language, d.h. eine Sprache, deren Syntax sich weitgehend an gewöhnlichem Englisch orientiert. Ein Beispiel:

```
select * from mybase.books where author_family_name = 'Münz'
```

Kommandos dieser Art werden in SQL als *Statements* bezeichnet. SQL-Statements werden vom DBMS-Client zum DBMS-Server einer SQL-fähigen Datenbank übertragen. Mit SQL-Statements teilt der Client dem Server also mit, was er von ihm will bzw. welche Aktion er im DBMS durchführen möchte. Die Antwort des DBMS-Servers besteht dann z.B. in einem Set von gefundenen Daten, in einer Statusantwort usw.

Es ist übrigens egal, ob Sie Schlüsselwörter in SQL-Statements groß oder klein schreiben – `SELECT` und `select` bewirken das Gleiche.

Statisches und dynamisches SQL

SQL-Statements müssen von einem SQL-fähigen DBMS natürlich intern interpretiert und umgesetzt werden. Dies nimmt eine gewisse Rechenzeit in Anspruch und je komplexer ein SQL-Statement ist, desto aufwändiger die Interpretation. Daher wurde und wird noch zwischen statischem und dynamischem SQL unterschieden. Beim statischen SQL werden benötigte SQL-Statements vorkompiliert und im DBMS gespeichert. Das erhöht die Geschwindigkeit bei Anfragen und verringert die Rechenzeit. Nachteil ist allerdings die mangelnde Flexibilität, die etwa für die meisten Webanwendungen inakzeptabel ist. Dort kommt in der Regel dynamisches SQL zum Einsatz, d.h., das DBMS muss SQL-Statements zur Anfragezeit in internen Code übersetzen. Um den damit verbundenen Performance-Verlust zumindest teilweise auszugleichen, führen viele DBMS Statistik über häufig abgesetzte SQL-Statements und halten deren Kompilate in einem Speicherpuffer fertig vor.

Weiterführende Links:

Wikipedia – alle Artikel zur Kategorie »Datenbank«:
http://de.wikipedia.org/wiki/Kategorie:Datenbank
SQL-Tutorial:
http://www.sql-und-xml.de/sql-tutorial/
Relationale Datenbanken:
http://v.hdm-stuttgart.de/~riekert/lehre/db-kelz/
Datenbank, MySQL und PHP:
http://ffm.junetz.de/members/reeg/DSP/

13.1.4 Versionen, Bezug und Installation von MySQL

Nachdem Sie nun mit der prinzipiellen Funktionsweise von DBM-Systemen vertraut sind, fällt es Ihnen vermutlich auch leichter, sich vorzustellen, was Sie erhalten, wenn Sie MySQL downloaden oder auf einem gemieteten Root-Server fertig installiert vorfinden. MySQL besteht bei den meisten Installationen aus einem Paket von unter anderem folgenden Programmen:

- dem MySQL-Server (*mysqld* mit Namen),
- einem kommandozeilenorientiertem Client mit Namen *mysql*, der Datenbankverbindungen herstellen kann, das Absetzen von SQL-Statements erlaubt und Datenausgaben mit eingeschränktem Komfort anzeigen kann,
- einem Dienstprogramm namens *mysqladmin* (unter MS Windows auch *winmysqladmin*), welches Verwaltungsarbeiten auf MySQL-Protokollebene ermöglicht, wie z.B. das Abschießen problematischer Prozesse, das Durchführen von MySQL-Server-Tests, die Überwachung laufender MySQL-Aktivitäten usw.

Diese Programme befinden sich nach einer MySQL-Installation neben einigen anderen Programmen im Programmunterverzeichnis *bin*. Daneben besteht eine Neuinstallation von MySQL aus diversen Testscripts, einer umfangreichen Dokumentation, Programmbibliotheken, Konfigurationsdateien sowie aus Default-Datenbanken, die z.B. für die Benutzerverwaltung gedacht sind.

Überprüfen einer bereits installierten MySQL-Version

Um auf einem entfernten Linux-Server-Rechner, auf dem Sie sich über SSH angemeldet haben, herauszufinden, ob dort MySQL installiert ist, geben Sie als Shell-Kommando ein:

```
which mysqladmin
```

Ist die Antwort eine Pfadangabe, so können Sie als Nächstes testen, ob der MySQL-Server gestartet ist. Angenommen, das which-Kommando hat */usr/local/mysql/bin/mysqladmin* ausgegeben. Dann können Sie als Nächstes eingeben:

```
/usr/local/mysql/bin/mysqladmin version
```

Bei einer jungfräulichen Installation gibt ein gestarteter MySQL-Server nun Informationen zu seiner Version sowie einige Statusinformationen aus. Wurde die Installation bereits konfiguriert, kann es sein, dass Sie eine Fehlermeldung ***Access denied ...*** erhalten. In diesem Fall müssen Sie sich beim Aufruf mit einem Benutzernamen und einem Passwort anmelden, das Sie vom Provider oder einem Systemadministrator erhalten haben müssen. Angenommen, Sie melden sich als Benutzer ***root*** an, dann sieht der Aufruf so aus:

```
/usr/local/mysql/bin/mysqladmin version -u root -p
```

Anschließend erhalten Sie eine Eingabeaufforderung für das Passwort. Die Passworteingabe erfolgt aus Sicherheitsgründen ohne sichtbares Echo am Bildschirm. Nach erfolgreicher Passworteingabe sollte ein gestarteter MySQL-Server die bereits erwähnten Versions- und Statusinformationen ausgeben.

Falls Sie MySQL so vorfinden und kein Passwort dafür besitzen, müssen Sie den Provider oder Server-Administrator fragen, mit welchen Zugangsdaten Sie MySQL benutzen können.

Unter neueren Windows-Versionen wie Windows 2000 und Windows XP können Sie in der Liste der gestarteten Dienste (Services) nachsehen, ob der MySQL-Server dort eingerichtet ist und ob er gestartet ist.

Download, Versionen und Installationshinweise zu MySQL

Die Download-Adresse für MySQL lautet *http://dev.mysql.com/downloads/*.

Für Linux werden Download-Pakete in allen nur erdenklichen Varianten angeboten. Für den Betrieb unter Linux sollte ein Linux-Kernel 2.2.19 oder höher laufen. Es werden wahlweise selbst zu kompilierende tar-gz-Archive oder vorkompilierte RPM-Pakete angeboten. Die RPM-Pakete sind, sofern für die eingesetzte Prozessor-Hardware verfügbar, sicher die bessere Wahl, da sie sich weniger aufwändig installieren lassen und bereits optimal kompiliert sind. Wichtig für den Praxisbetrieb ist am Ende einer Neu- und Erstinstallation von MySQL unter Linux, dass das mitgelieferte Shellscript ***mysql_install_db*** ausgeführt wird. Es befindet sich im Programmunterverzeichnis ***scripts*** oder ***bin***. Dieses Script besorgt die nötige Anfangskonfiguration von MySQL.

Auch unter MS Windows können Sie MySQL installieren. Für den produktiven Einsatz empfiehlt es sich auf jeden Fall, eine fertig kompilierte Version zu nehmen. Für Windows werden dazu ein ***Essential Package*** und ein ***Complete Package*** angeboten. Für normale Zwecke genügt das Essential Package vollauf. Beide Pakete verwenden den Windows Installer (das Setup ist eine ***msi***-Datei). Die Windows-Installation beinhaltet auch einen über mehrere Schritte führenden Wizard, der MySQL so installiert, wie es für Ihre Zwecke am sinnvollsten ist. Da keine Shellprogrammiersprache zur Verfügung steht, übernimmt der Wizard auch die Anfangskonfiguration wie z. B. das Festlegen eines Passworts für den MySQL-Administrator (***root***).

Zu allen Installationspaketen gibt es ausführliche Hinweise im MySQL-Benutzerhandbuch unter der folgenden Adresse:

http://netmirror.org/mirror/mysql.com/doc/mysql/de/ (deutschsprachig)



Abbildung 13.1: Deutschsprachige Version des Benutzerhandbuchs zu MySQL

Wegen der zahlreichen Installationspakete und den Hinweisen dazu verweisen wir auf diese Adresse, da eine Wiedergabe aller Installationsaspekte den Rahmen des vorliegenden Buches sprengen würde.

Auf manchen Server-Rechnern sind noch 3.x-Versionen von MySQL installiert. Aktuell ist jedoch die 4er-Version. Bei Neuinstallationen spricht nichts gegen die Verwendung einer 4er-Version. Die Performance ist im Detail optimiert worden und gerade für professionelle Ansprüche in Sachen Transaktionsfähigkeit (konsistente Datenspeicherung) ist die 4er-Version ein Muss, da die 3er-Version solche Features noch gar nicht kannte.

Starten und Beenden des MySQL-Servers

Unter Linux verwenden Sie zum Starten, Beenden oder Neustarten des MySQL-Servers am besten das Shellscript *mysql.server*. Es befindet sich üblicherweise im MySQL-Programmunterverzeichnis *share* und dort im Unterverzeichnis *mysql*. Im Zweifelsfall können Sie auf Shell-Ebene mit folgendem Kommando danach suchen:

```
find / -name mysql.server
```

Wechseln Sie mit `cd [Pfad]` in das Verzeichnis des Scripts.

Um den MySQL-Server zu starten, geben Sie ein:

```
mysql.server start
```

Um ihn zu stoppen, geben Sie ein:

```
mysql.server stop
```

Unter MS Windows (2000, 2003, NT, XP) wird der MySQL-Server als Service (Dienst) gestartet (üblicherweise erreichbar unter EINSTELLUNGEN/SYSTEMSTEUERUNG/VERWALTUNG/DIENSTE). Suchen Sie in der Liste der Dienste den Eintrag für den MySQL-Server. Dieser sollte bei der Installation dort eingerichtet worden sein. Wenn der Server den Status gestartet hat, dann läuft er. Andernfalls können Sie, indem Sie mit der rechten Maustaste auf den Eintrag klicken, über das Kontextmenü den Eintrag STARTEN wählen. Damit starten Sie den Server.

13.2 Basiswissen SQL

SQL, so wurde bereits erwähnt, ist die führende Transformationssprache, mit deren Hilfe einem DBMS mitgeteilt werden kann, welche Aktion es ausführen soll.

13.2.1 Erste SQL-Schritte mit dem mysql-Client

Der mysql-Client, ein einfaches, kommandozeilenorientiertes Werkzeug, ist bei allen MySQL-Distributionen mit dabei. Wenn Sie auf Shell-Ebene arbeiten, etwa über einen SSH-Zugang, ist er – SQL-Kenntnisse vorausgesetzt – ein hervorragendes Werkzeug, um MySQL zu administrieren.

Starten und Beenden des MySQL-Clients

Durch das Shell-Kommando `which mysql` können Sie herausfinden, wo der MySQL-Client installiert ist. Starten Sie das betreffende Programm, also *mysql*, entweder durch Angabe des Pfadnamens oder durch vorheriges Wechseln ins Programmverzeichnis. Je nach Umgebungskonfiguration kann es auch sein, dass das Programmverzeichnis von MySQL im Pfad liegt und das Clientprogramm daher von überall ohne Angabe eines Pfads aufrufbar ist. Beim Start sollten Sie Benutzernamen und Wunsch nach Passwortanforderung gleich mitsenden. Beispiel:

```
mysql -u root -p
```

Durch die Option `-u` wird ein anschließend genannter Benutzer am MySQL-Server angemeldet. Die Option `-p` gibt an, dass der Benutzer ein Passwort eingeben muss. Beim Start fordert der mysql-Client dann zur Eingabe des Passworts auf. Die Eingabe erfolgt aus Sicherheitsgründen verdeckt.

Der angegebene Benutzer muss in MySQL bereits eingerichtet worden sein. Nach einer vollständigen Installation von MySQL, in deren Verlauf (gilt für Linux/Unix) auch das Shellscript *mysql_install_db* ausgeführt wurde, existiert zumindest ein Benutzer namens *root*, dem jedoch noch kein Passwort zugewiesen wurde. In diesem Fall können Sie mysql wie oben beschrieben aufrufen und die Passwortaufforderung einfach durch Drücken der Enter-Taste überspringen.

Wenn ein Prompt, beschriftet mit `mysql>`, erscheint, befinden Sie sich im MySQL-Client. An diesem Prompt können Sie SQL-Statements absetzen und dadurch auf die von MySQL verwalteten Datenbanken zugreifen. Die Möglichkeiten richten sich dabei natürlich danach, was Ihnen als Benutzer erlaubt ist. Falls Sie sich nach einer Neuinstallation als *root* angemeldet haben, sind Sie der Superadministrator von MySQL und dürfen alles.

Unter MS Windows müssen Sie den MySQL-Client in einem DOS-Fenster ausführen. Glücklicherweise bieten neuere Windows-Versionen ja zumindest rudimentäre Möglichkeiten, ein solches Fenster halbwegs erträglich aussehen zu lassen (z.B. schwarze Schrift auf weißem Grund, Schriftartenwahl bis zu einem gewissen Grad, Anpassung der Fenstergröße). Am einfachsten starten Sie den Client, indem Sie START/AUSFÜHREN wählen und das Programm *mysql.exe* im Programmverzeichnis von MySQL suchen und starten.

Um den MySQL-Client zu beenden, geben Sie am Prompt als Kommando einfach `quit` oder `exit` ein.

Fehlermeldungen des MySQL-Clients

Bei Syntaxfehlern in eingegebenen SQL-Statements oder in anderen problematischen Situationen gibt der MySQL-Client eine Fehlermeldung oder zumindest eine Warnung aus. Erscheint anstelle eines Ergebnisses wieder der Prompt, so liegt ein Fehler vor, meistens ein Syntaxfehler. Studieren Sie in diesem Fall die Fehlermeldung genau. Der MySQL-Interpreter versucht in der Regel auch, die Stelle in einem SQL-Statement anzugeben, die aus seiner Sicht zu einem Fehler führt.

Statement-History im MySQL-Client

Mithilfe der Pfeiltasten können Sie im MySQL-Client in bisher eingegebenen SQL-Statements blättern. Das ist praktisch, um ein Statement zu wiederholen oder um es als Basis einer leichten Abänderung zu verwenden.

Erste Einsichten in MySQL-Datenbanken

Geben Sie am Prompt des MySQL-Clients ein:

```
show databases;
```

Wichtig ist, dass Sie am Ende von SQL-Statements stets ein Semikolon notieren. Schicken Sie SQL-Statements genauso wie Kommandos auf der Shell mit Enter ab.

Mit dem obigen Statement erhalten Sie eine Liste von aktuell eingerichteten Datenbanken. Auf jeden Fall dabei sein sollte eine Datenbank mit dem Namen *mysql*. In dieser Datenbank verwaltet MySQL seine Benutzerkonfiguration. Werfen wir einen Blick in diese Datenbank:

```
use mysql;
```

Damit wechseln Sie zur Datenbank mit dem Namen *mysql*. Das Statement ist nicht zwingend nötig. Es ist dem cd-Kommando bei Verzeichnissen vergleichbar. Sie können entweder vor dem Arbeiten in einem Verzeichnis mit `cd` dorthin wechseln (dies tut `use` mit Datenbanken) oder Sie müssen bei jeder Dateioperation den Pfadnamen mit angeben (dies wäre auch in SQL nötig, wenn zuvor nicht zu einer bestimmten Datenbank gewechselt wird).

Um einen Überblick zu erhalten, welche Daten in einer Datenbank gespeichert werden, empfiehlt es sich, ihre Tabellen auflisten zu lassen:

```
show tables;
```

Damit erhalten Sie eine Liste von Tabellen, die zur ausgewählten Datenbank gehören. Das Ergebnis sollte so aussehen:

```
+--------------+
|    Tables    |
+--------------+
| columns_priv |
| db           |
| func         |
| host         |
| tables_priv  |
| user         |
+--------------+
```

Eine relationale Datenbank speichert ihre Daten in Tabellen. Jeder Datensatz entspricht einer Zeile darin und jedes Feld einer Spalte. Eine Datenbank besteht in der Regel aus mehr als einer Tabelle und »relational« wird eine Datenbank überhaupt erst, wenn mehrere Tabellen mit Feldern existieren, die das Herstellen einer tabellenübergreifenden Verknüpfung erlauben.

Auf die Datenbank *mysql* und ihre Tabellen werden wir in ***Abschnitt 13.2.7*** in Zusammenhang mit dem wichtigen Thema der Benutzerverwaltung zurückkommen. An dieser Stelle soll lediglich vermittelt werden, wie Sie sich Informationen und Einblick in eine Mysql-Datenbank verschaffen können.

Um die Inhalte einer Tabelle aufzulisten, können Sie das wohl bekannteste SQL-Statement `select` verwenden. Um beispielsweise alle Datensätze der Tabelle *user* aufzulisten, geben Sie ein:

```
select * from user;
```

Die Tabelle *user* in der Datenbank mysql enthält relativ viele Felder pro Datensatz. Daher fällt die Darstellung im textorientierten mysql-Client etwas chaotisch aus.

Das select-Statement von SQL ist vermutlich das komplexeste der Sprache überhaupt. Es erlaubt selbstverständlich auch Einschränkungen, so etwa die exklusive Auflistung von bestimmten Feldern oder Datensätzen, die bestimmten Kriterien genügen. Das select-Statement wird uns in ***Abschnitt 13.2.4*** noch eingehender beschäftigen.

Manchmal sind es jedoch nicht die Tabelleninhalte, die von Interesse sind, sondern Informationen darüber, welche Art von Daten in einer Tabelle gespeichert werden, also, aus welchen Feldern die Tabelle besteht. Auf die Tabelle *user* in der Datenbank *mysql* angewendet, können Sie diese Informationen wie folgt ermitteln:

```
describe user;
```

Damit erhalten Sie eine Auflistung aller Felder dieser Tabelle, mitsamt Angaben zum Feldtyp und weiteren Eigenschaften. Die Spalte *Fields* enthält die Feldnamen, die Spalte *Type* den Datentyp des Feldes, die Spalte *Null* eine Angabe darüber, ob das Feld leer sein darf oder nicht, die Spalte *Key* besagt, ob ein Index für die Spalte, die dieses Feld darstellt, eingerichtet wurde, und die Spalte *Extra* enthält weitere Informationen.

13.2.2 Datenbanken und Tabellen anlegen, ändern und löschen

Um den Umgang mit SQL und MySQL zu erlernen, werden wir nun eine Übungsdatenbank erstellen, und zwar eine Datenbank, die später als Basis zur dynamischen Seitenerzeugung dienen kann. Die Datenbank soll folgende Struktur haben:

Tabelle *pages*	**Tabelle** *authors*	**Tabelle** *styles*	**Tabelle** *templates*
table_key	*table_key*	*table_key*	*table_key*
get_name	*given_name*	*css_file*	*template_file*
title	*family_name*	*record_timestamp*	*record_timestamp*
keywords	*mail*	*description*	*description*
description	*phone*		
charset	*record_timestamp*		
author_key	*address*		
last_mod			
style_key			
template_key			
record_timestamp			
content			

Tabelle 13.1: Entwurf einer Übungsdatenbank

Die Datenbank soll den Titel *website* erhalten. Die dargestellten Spalten entsprechen den geplanten Tabellen und die Listen darin den geplanten Feldnamen der jeweiligen Tabellen. Die grundsätzliche Überlegung ist, dass jede Seite aus allgemeinen Daten besteht, die als Basis für HTML-Kopfdaten dienen können. Einige dieser Daten sind seitenindividuell,

wie z. B. Titel (*title*) oder Beschreibung (*description*). Andere Daten kommen bei vielen Seiten vor, etwa das HTML-Template, das für diese Seite verwendet werden soll, das Stylesheet, das referenziert werden soll, oder der Autor der betreffenden Seite. Deshalb werden diese Daten in der Tabelle *pages* nur über ein Feld referenziert, das eine Verknüpfung zu einem bestimmten Datensatz in den Tabellen *templates*, *styles* oder *authors* herstellt. Diese Art, Informationen auf Tabellen zu verteilen und zu verknüpfen, ist typisch für relationale Datenbanken.

Für umfangreichere SQL-Arbeiten: MySQL-Client im Batch-Betrieb

Bevor wir unsere Beispieldatenbank anlegen und Inhalte einfügen, ist es besser, den MySQL-Client mit `quit` oder `exit` zu beenden und die SQL-Statements in einem Editor nach Wahl einzugeben. Sie können z. B. auch lokal auf Ihrem PC mit einem Editor Ihrer Wahl arbeiten und die damit erzeugten Dateien später auf den Server-Rechner hochladen. Der Grund ist einfach, dass der MySQL-Client zur Bearbeitung komplexerer Statements nicht sehr komfortabel ist. Er erlaubt jedoch das Einlesen von SQL-Statements aus Dateien. Und genau davon machen wir nun Gebrauch.

Im Editor unserer Wahl legen wir eine neue Datei an mit Namen *website_create.sql*. In dieser Datei werden wir alle SQL-Statements notieren, die zur Erzeugung der Datenbank *website* erforderlich ist. Dies hat auch den Vorteil, dass die Datei wiederverwendbar ist, also z. B. auf einem anderen Rechner das Anlegen der gleichen Datenbank erlaubt.

Die Dateinamenerweiterung *.sql* ist zwar nicht vorgeschrieben, aber sinnvoll, um Dateien dieser Art zu kennzeichnen.

Datenbank anlegen

Um die gewünschte Datenbank neu anzulegen, können wir folgendes SQL-Statement in die neue SQL-Datei schreiben:

```
create database website;
```

Damit wird die Datenbank in MySQL erzeugt, sie ist jedoch ansonsten völlig leer.

Falls die Datenbank bereits existiert, würde dieses Statement allerdings einen Fehler erzeugen. Professioneller ist es daher, folgendes Statement zu verwenden:

```
create database if not exists website;
```

Damit wird eine eventuell existierende Datenbank jedoch nicht überschrieben, sondern es passiert in diesem Fall einfach gar nichts. Eine neue Datenbank wird erst dann angelegt, wenn eine vorher existierende gleichnamige Datenbank gelöscht wird.

Datenbank löschen

Um eine angelegte Datenbank zu löschen, geben Sie ein:

```
drop database [Datenbankname];
```

Auch hier lassen sich eventuelle Fehler vermeiden:

```
drop database if exists [Datenbankname];
```

Tabellen anlegen

Als Nächstes müssen wir in der neu angelegten Datenbank die gewünschten Tabellen einrichten. Das SQL-Statement dazu ist `create table`. Das Statement gehört jedoch zu denen, die etwas komplexer werden können, da mit einem einzigen Statement eine gesamte Tabelle mit all ihren Feldern und deren Eigenschaften erzeugt wird. Beginnen wir in unserem Beispiel mit der Tabelle *pages*: Im Editor notieren wir:

```
create table if not exists website.pages (
   table_key int unsigned auto_increment,
   get_name varchar(50) not null,
   title varchar(150) not null,
   keywords blob default '',
   description blob default '',
   charset varchar(30) default 'iso-8859-1' not null,
   author_key int unsigned default 0,
   last_mod datetime default '2005-01-01 00:00:00' not null,
   style_key int unsigned default 0,
   template_key int unsigned default 0,
   content blob default '',
   record_timestamp timestamp(14),
   primary key(table_key),
   unique(get_name),
   key(title)
);
```

Das Statement hat die folgende allgemeine Form:

```
create table [Tabellenname] ( [Felddefinitionen] );
```

Auch in diesem Fall kann durch den Zusatz `if not exists` verhindert werden, dass MySQL einen Fehler meldet, falls die Tabelle bereits existiert. Falls die Tabelle bereits existiert, wird sie nicht neu angelegt. Vorhandene gleichnamige Tabellen müssen erst gelöscht werden, bevor neue angelegt werden können.

Tabellen löschen

Um eine Tabelle zu löschen, steht folgendes SQL-Statement zur Verfügung:

```
drop table [Tabellenname];
```

Syntax der Felddefinitionen bei create table

Im obigen Beispiel wird die Tabelle *pages* angelegt. Die Adressierungsform `Datenbankname.Tabellenname` ist übrigens nötig, wenn zuvor nicht mit `use` zu einer bestimmten Datenbank gewechselt wird.

Innerhalb der runden Klammern von `create table()` werden die einzelnen Felder der Tabelle sowie Indexe für bestimmte Felder definiert. Felddefinitionen haben folgendes Format:

```
Feldname Feldtyp Optionen
```

Die einzelnen Feld- und Indexdefinitionen werden durch Kommata getrennt, d.h., jede Feld- oder Indexdefinition wird mit einem Komma abgeschlossen, mit Ausnahme der letzten. Im obigen Beispiel werden Felder mit den Namen `table_key`, `get_name`, `title`, `keywords`, `description`, `charset`, `author_key`, `last_mod`, `style_key`, `template_key`, `content` und `record_timestamp` definiert.

Namenskonventionen

Die Namenskonventionen für Datenbank- und Tabellennamen richten sich nach dem verwendeten Betriebssystem, da MySQL für jede Datenbank ein Verzeichnis anlegt und für jede Tabelle Dateien, deren »Vornamen« den vergebenen Tabellennamen tragen. Damit MySQL-Datenbanken portabel sind, empfiehlt es sich also, sich bei der Namensvergabe auf Buchstaben aus dem ASCII-Zeichensatz, auf Ziffern und auf den Unterstrich zu beschränken. Auch die Unterscheidung von Groß-/Kleinschreibung richtet sich nach dem Betriebssystem. Unter MS Windows ist *Website* und *website* das Gleiche, unter Linux/Unix nicht.

Bei Feldnamen akzeptieren neuere MySQL-Versionen so ziemlich alle Zeichen. Auch hier empfiehlt es sich jedoch, sich an die älteren Standards zu halten. Danach sind alle Buchstaben aus dem voreingestellten Zeichensatz (wenn nichts anderes festgelegt wurde, ist das ISO-8859-1), alle Ziffern, der Unterstrich (_) und das Dollarzeichen ($) erlaubt.

Namen von Datenbanken, Tabellen und Feldern dürfen maximal 64 Zeichen enthalten.

Feldtypen

Beim Feldtyp, also der Angabe hinter dem Feldnamen, bestimmen Sie, welche Art von Daten in dem Feld gespeichert werden sollen. Nachfolgende Tabelle listet die gängigen Datentypen von MySQL auf, die hierbei verwendet werden können:

Feldtyp	Varianten	Bedeutung
bigint	*unsigned* *zerofill*	Feldtyp für große Ganzzahlen (8 Byte Speicherbedarf). Ohne die Zusatzangabe *unsigned* beträgt der Wertebereich -9.223.372.036.854.775.807 bis 9223372036854775808, mit unsigned beträgt er 0 bis 18.446.744.073.709.551.615. Mit der Zusatzangabe *zerofill* wird je nach Zahlengröße von links her mit 0 aufgefüllt.
blob		Feldtyp für Freitext oder Binärdaten mit einer Gesamtlänge von bis zu 65.535 Byte.

Tabelle 13.2: Gängige Feld- bzw. Datentypen in MySQL

Feldtyp	Varianten	Bedeutung
char(x)	*binary*	Zeichenkette mit fester Länge. die Länge wird mit *x* angegeben. *x* kann zwischen 0 und 255 betragen. Inhalte mit weniger Zeichen werden rechts mit Leerzeichen aufgefüllt. MySQL speichert so viele Bytes wie bei *x* angegeben sind. Wenn die Zeichenkette länger als *x* ist, wird sie rechts abgeschnitten.
date		Datum im Format *YYYY-MM-DD*. Der Wertebereich ist *1000-01-01 bis 9999-12-31*. 3 Byte Speicherbedarf.
datetime		Datum im Format *YYYY-MM-DD HH:MM:SS*, z.B. *2005-03-21 12:10:45*. Der Wertebereich ist *1000-01-01 00:00:00* bis *9999-12-31 23:59*. 8 Byte Speicherbedarf.
decimal[(x,y)]	*zerofill*	Feldtyp für Fließkommazahlen, bei denen die Genauigkeit erhalten bleiben soll. Die Zahlen werden zu diesem Zweck als Zeichenkette abgespeichert. x ist die gewünschte Anzahl Vorkommastellen und y die gewünschte Anzahl Nachkommastellen. Wird *y* weggelassen, so wird es auf 0 gesetzt (keine Nachkommastellen). Wird auch *x* weggelassen, wird es auf 10 gesetzt. Mit der Zusatzangabe *zerofill* wird je nach Zahlengröße von links her mit 0 aufgefüllt.
double[(x,y)]	*zerofill*	Fließkommazahl mit doppelter Genauigkeit im Wertebereich von -3.402823466E+38 bis -1.175494351E-38, 0 und 1.175494351E-38 bis 3.402823466E+38 (8 Byte Speicherbedarf). *x* und *y* sind optional. *x* bestimmt die Anzeigebreite, *y* die Anzahl von Nachkommastellen. Mit der Zusatzangabe *zerofill* wird je nach Zahlengröße von links her mit 0 aufgefüllt.
enum(...)		Aufzählungstyp für feste mögliche Werte, z.B. *enum(»j«, »n«)*. Es sind bis zu 65.535 Werte möglich.
float[(x,y)]	*zerofill*	Fließkommazahl mit einfacher Genauigkeit im Wertebereich von -1.79769313486231 57E+308 bis -2.2250738585072014E-308, 0 und 2.2250738585072014E-308 bis 1.79769313486231 57E+308 (4 Byte Speicherbedarf). *x* und *y* sind optional. *x* bestimmt die Anzeigebreite, *y* die Anzahl von Nachkommastellen. Mit der Zusatzangabe *zerofill* wird je nach Zahlengröße von links her mit 0 aufgefüllt.
int	*unsigned* *zerofill*	Feldtyp für normalgroße Ganzzahlen (4 Byte Speicherbedarf). Ohne die Zusatzangabe *unsigned* beträgt der Wertebereich -2.147.483.648 bis 2.147.483.647, mit *unsigned* beträgt er 0 bis 4.294.967.295. Mit der Zusatzangabe *zerofill* wird je nach Zahlengröße von links her mit 0 aufgefüllt.
longblob		Feldtyp für Freitext oder Binärdaten mit einer Gesamtlänge von bis zu 4.294.967.295 Byte.
mediumblob		Feldtyp für Freitext oder Binärdaten mit einer Gesamtlänge von bis zu 16.777.215 Byte.
mediumint	*unsigned* *zerofill*	Feldtyp für mittelgroße Ganzzahlen (3 Byte Speicherbedarf). Ohne die Zusatzangabe *unsigned* beträgt der Wertebereich -8.388.608 bis 8.388.607, mit *unsigned* beträgt er 0 bis 16.777.215. Mit der Zusatzangabe *zerofill* wird je nach Zahlengröße von links her mit 0 aufgefüllt.

Tabelle 13.2: Gängige Feld- bzw. Datentypen in MySQL (Fortsetzung)

Feldtyp	Varianten	Bedeutung
set(...)		Aufzählungstyp für feste mögliche Werte, z.B. *set(»j«, »n«)*. Es sind bis zu 64 Werte möglich.
smallint	*unsigned zerofill*	Feldtyp für kleinere Ganzzahlen (2 Byte Speicherbedarf). Ohne die Zusatzangabe *unsigned* beträgt der Wertebereich -32.768 bis 32.767, mit *unsigned* beträgt er 0 bis 65.535. Mit der Zusatzangabe *zerofill* wird je nach Zahlengröße von links her mit 0 aufgefüllt.
time		Uhrzeit im Format *HH:MM:SS*. 3 Byte Speicherbedarf.
timestamp		Zeitstempel der Erzeugung oder Änderung eines Datensatzes im Format *YYYYMMDDHHMMSS*. 4 Byte Speicherbedarf.
tinyblob		Feldtyp für Freitext oder Binärdaten mit einer Gesamtlänge von bis zu 255 Byte.
tinyint	*unsigned zerofill*	Feldtyp für kleine Ganzzahlen (1 Byte Speicherbedarf). Ohne die Zusatzangabe *unsigned* beträgt der Wertebereich -128 bis 127, mit *unsigned* beträgt er 0 bis 255. Mit der Zusatzangabe *zerofill* wird je nach Zahlengröße von links her mit 0 aufgefüllt.
varchar(x)		Zeichenkette mit variabler Länge. Die maximale Länge wird mit *x* angegeben. *x* kann zwischen 0 und 255 betragen. MySQL speichert so viele Bytes wie die Zeichenkette tatsächlich hat, plus ein Byte für die Länge der Zeichenkette. Wenn die Zeichenkette länger als *x* ist, wird sie rechts abgeschnitten.
year		Jahreszahl zwischen 1901 und 2155. 1 Byte Speicherbedarf.

Tabelle 13.2: Gängige Feld- bzw. Datentypen in MySQL (Fortsetzung)

In unserer Beispieltabelle *pages* haben wir die Datentypen `int unsigned`, `varchar`, `blob`, `datetime` und `timestamp` benutzt.

Feldoptionen

Beim ersten Feld (`table_key`) wurde in unserem Beispiel die Option `auto_increment` notiert. Damit wird bei jedem neuen Datensatz automatisch eine neue höchste Nummer vergeben. Auf diese Weise erhält jeder Datensatz eine eindeutige Nummer. Bei allen Tabellen, deren Datensätze aus irgendeinem Grund eindeutig identifizierbar sein sollen, z.B. für relationale Verknüpfungen zwischen Tabellen, empfiehlt es sich, ein Feld mit dieser Option zu definieren.

Eine weitere Option ist `default`. Damit können Sie einen Vorbelegungswert definieren. Sollte ein Datensatz beim Anlegen keinen expliziten Wert für dieses Feld erhalten, erhält er den Default-Wert. Zeichenkettenwerte oder Datums-/Zeitangaben werden in einfache oder doppelte hohe Anführungszeichen gesetzt, numerische Werte nicht. Bei Fließkommazahlen ist das Dezimalzeichen der Punkt.

Die Option `not null` bewirkt, dass MySQL keine Datensätze speichert, ein so definiertes Feld leer ist, also keinen Wert hat. Alles, was also ein Pflichtfeld sein soll, sollte mit dieser Option definiert werden.

Indexe (Schlüssel)

In den Definitionen zu unserer Beispieltabelle *pages* werden nicht nur die Felder der Tabelle definiert, sondern für ein Feld auch ein `primary key` sowie für zwei weitere Felder je ein `key`. Wenn solche Angaben zu einer Tabelle notiert werden, dann legt MySQL einen Index für dieses Feld an. Indexe erlauben einen schnelleren Zugriff auf gesuchte Daten. Existiert zu einem Feld kein Index und wird in dem Feld nach einem Wert gesucht, so muss MySQL einen so genannten *Full Table Scan* durchführen, also der Reihe nach in allen Datensätzen nach dem Wert im gewünschten Feld suchen. Ein Index über ein Feld ermöglicht MySQL den gezielteren Zugriff auf einen gesuchten Wert.

- Ein *Primärindex* (`primary key`) darf nur einmal pro Tabelle vergeben werden.
- Ein *Unique-Index* (`unique`) darf nur für Felder vergeben werden, bei denen jeder Datensatz einen unterschiedlichen Wert hat.
- Ein *Sekundärindex* oder *gewöhnlicher Index* (`key`) kann prinzipiell für alle Felder vergeben werden.

Hinter dem Schlüsselnamen, also `primary key`, `unique` oder `key`, wird in runden Klammern der Name des gewünschten Feldes angegeben. Beispiel:

```
create table employees (
   table_key int unsigned auto_increment,
   given_name varchar(50) not null,
   family_name varchar(50) not null,
   personal_number int unsigned not null,
   primary key(table_key),
   unique(personal_number),
   key(family_name)
);
```

Tabellenstrukturen ändern

Eine Tabellenstruktur, die mit `create table()` angelegt wurde, sollte zwar vorher gut durchdacht sein. Doch in der Praxis kommt es auch bei guter Planung immer einmal vor, dass ursprüngliche Definitionen den Erfordernissen nicht oder nicht mehr gerecht werden. Dann sind Änderungen bei Tabellenstrukturen erforderlich. Das ist so weit kein Problem, auch dann nicht, wenn die Tabellen bereits Daten enthalten. Allerdings können Strukturänderungen je nach Art Konsequenzen für bisher gespeicherte Daten haben.

Zum Ändern einer Tabellenstruktur steht das SQL-Statement `alter table()` zur Verfügung.

Wenn Sie eine Spalte, also ein Feld hinzufügen möchten, lautet das Schema:

```
alter table <Tabellenname> add <Feldname> <Definition> [first | after Feldname];
```

Ein Beispiel:

```
alter table contacts add age tinyint default 0 after homepage_url;
```

Damit wird in der Tabelle *contacts* hinter dem Feld *homepage_url* ein neues Feld namens *age* eingefügt, das vom Typ her einen tinyint-Wert enthält und den Defaultwert 0 haben soll. Zu `[Definition]` gehören also die gewünschten Angaben zu Feldtyp und zu eventuellen Feldoptionen, wobei die gleichen Angaben möglich sind wie bei `create table()`.

Wenn Sie eine Spalte, also ein Feld löschen wollen, lautet das Schema:

```
alter table <Tabellenname> drop <Feldname>;
```

Ein Beispiel:

```
alter table contacts drop telex_number;
```

Alle in diesem Feld vorhandenen Daten werden dabei gelöscht!

Wenn Sie nur den Feldtyp eines Feldes oder die Optionen ändern wollen, lautet das Schema:

```
alter table <Tabellenname> modify <Feldname> <Definition>;
```

Ein Beispiel:

```
alter table contacts modify mail_address varchar(120) not null;
```

Beachten Sie, dass je nach Art der Änderung Daten verloren gehen, verkürzt oder unsinnig werden können!

Wenn Sie ein Feld umbenennen möchten, lautet das Schema:

```
alter table <Tabellenname> change <alter Feldname> <neuer Feldname>
<Definition>;
```

Ein Beispiel:

```
alter table contacts change mail_address mail varchar(120) not null;
```

In diesem Beispiel wird das Feld *mail_address* in *mail* umbenannt. Um den bisherigen Datentyp beizubehalten, müssen Sie die gleichen Definitionen zu Feldtyp und Optionen angeben wie bereits vorhanden. Sie haben jedoch die Möglichkeit, das Feld sowohl umzubenennen als auch Änderungen bei Feldtyp und Optionen vorzunehmen.

Beim reinen Ändern von Feldnamen bleiben alle darin gespeicherten Werte erhalten.

Um eine ganze Tabelle umzubenennen, steht folgende Syntax zur Verfügung:

```
rename <Tabellenname> to <neuer Tabellenname>;
```

Ein Beispiel:

```
rename contacts to contact_partners;
```

Nicht nur Tabellenfelder, sondern auch zur Tabelle gehörige Indexe können geändert werden. Beispiele:

```
alter table contacts add primary key(id_number);
alter table contacts drop primary key;
alter table contacts add index(age);
alter table contacts drop index(phone_number);
```

Weitere Beispieltabellen

Unsere Beispieldatenbank *website* erhält folgende weiteren Tabellendefinitionen in der Datei *website_create.sql*:

```
create table if not exists website.authors (
   table_key int unsigned auto_increment,
   given_name varchar(50) not null,
   family_name varchar(50) not null,
   mail varchar(120) default '',
   phone varchar(30) default '',
   address varchar(255) default '',
   primary key(table_key),
   record_timestamp timestamp(14),
   key(given_name)
);

create table if not exists website.styles (
   table_key int unsigned auto_increment,
   css_file varchar(255) not null,
   description blob default '',
   record_timestamp timestamp(14),
   primary key(table_key)
);

create table if not exists website.templates (
   table_key int unsigned auto_increment,
   template_file varchar(255) not null,
   description blob default '',
   record_timestamp timestamp(14),
   primary key(table_key)
);
```

In allen Tabellen benutzen wir also ein Auto-Increment-Feld zur eindeutigen Datensatzidentifizierung sowie ein Feld namens `record_timestamp`, in das beim Einfügen oder Ändern von Datensätzen automatisch der Zeitstempel eingefügt wird.

13.2.3 Datensätze einfügen, ändern und löschen

Während Datendefinitionen, wie sie im vorherigen Abschnitt gezeigt wurden, in aller Regel von Entwicklerseite über den MySQL-Client und Batch-Dateien vorgenommen werden, gehört das Bearbeiten von Datensätzen typischerweise zur Benutzerschnittstelle.

Auch das lässt sich jedoch nicht verallgemeinern. In unserem Beispiel könnte ein einfaches, webbasiertes CMS (Content-Management-System) dafür sorgen, dass autorisierte Benutzer über Webbrowser und HTML-Formulare die angelegten Tabellen mit Daten füllen können. Das CMS müsste dabei dafür sorgen, dass, bevor Daten für die Tabelle *pages* eingegeben werden können, wenigstens je ein Datensatz für die Tabellen *authors*, *styles* und *templates* eingegeben wurde, da bei *pages* jeweils ein existierender Autor, ein existierendes Stylesheet und ein existierendes Template angegeben werden muss. »Existierend« bedeutet dabei: Es muss in der Datenbank existieren.

Aber egal, ob Sie ein solches CMS in PHP selber entwickeln oder ob Sie die in der Datenbank gespeicherten Seiten auf Systemebene einfügen: In jedem Fall müssen Sie die SQL-Syntax kennen, mit deren Hilfe Datensätze verwaltet werden.

Neue Datensätze anlegen

Um in einer Tabelle einen Datensatz hinzuzufügen, steht das `insert`-Statement von SQL zur Verfügung. Wir demonstrieren das Statement anhand unserer Beispieltabelle *authors*:

```
insert into website.authors(
      given_name,
      family_name,
      mail,
      phone,
      address)
   values(
      "Rainer",
      "Zufall",
      "rz@example.org",
      "01003-423834",
      "Schicksalsweg 7, 90001 Gibtsnich"
);
```

Das Schlüsselwort `into` darf auch weggelassen werden. Wenn beim Einfügen nur bestimmte Felder ausgefüllt werden sollen, müssen die Feldnamen in runden Klammern genannt werden, und zwar durch Kommata getrennt in der Reihenfolge, in der anschließend ihre Werte angegeben werden. Danach folgt in jedem Fall das Schlüsselwort `values` und dahinter, in runden Klammern, wieder durch Kommata getrennt, die gewünschten Werte für die Felder. Achten Sie darauf, dass hinter den jeweils letzten Einträgen in einer Klammer kein Komma mehr steht!

Es gibt auch noch eine alternative Schreibweise. Obiges Beispiel würde danach lauten:

```
insert into website.authors set
      given_name="Rainer",
      family_name="Zufall",
      mail="rz@example.org",
      phone="01003-423834",
      address="Schicksalsweg 7, 90001 Gibtsnich";
```

Direkt hinter dem Statement-Einleitungswort `insert` und vor `into` können Sie außerdem folgende Optionsschlüsselwörter notieren:

- `low_priority` oder `delayed:` Nur eines dieser beiden Schlüsselwörter kann angegeben werden. Mit `low_priority` geben Sie an, dass mit dem Einfügen des Datensatzes gewartet werden soll, bis kein anderer Client-Prozess mehr aus der betroffenen Tabelle liest. Bei intensiv genutzten Datenbanken kann dies lange dauern. Der Client wartet mit seiner Statusmeldung so lange, bis der Datensatz eingefügt werden konnte. Die Angabe `delayed` bewirkt ebenfalls, dass der Datensatz erst eingefügt wird, wenn kein Leseprozess mehr stattfindet, jedoch mit dem Unterschied, dass der Client nicht so lange warten muss, bis dies der Fall ist, sondern sofort für das nächste SQL-Statement bereit ist.
- `ignore` bewirkt, dass der Datensatz nicht eingefügt wird, falls ein Primary- oder Unique-Schlüssel auf ein Feld gelegt ist und der eingegebene Feldwert im neuen Datensatz einem bereits existierenden Feldwert entspricht.

Notation von Zeichenketten und Zahlen

Zeichenkettenwerte und Datums-/Zeitangaben müssen in doppelten oder einfachen hohen Anführungszeichen notiert werden, Zahlen dagegen nicht. Innerhalb von Zeichenketten müssen einige Zeichen mithilfe des Zeichens `\` maskiert werden:

Zeichen	Notation
'	\'
"	\"
\	\\
%	\%
_	_

Tabelle 13.3: Maskierung von Zeichen in MySQL-Zeichenketten

Daneben können Sie mithilfe des Backslash auch bestimmte Steuerzeichen notieren:

Notation	Bedeutung
\0	Zeichen mit Binärwert 0
\n	Steuerzeichen für Zeilenumbruch
\r	Steuerzeichen für Wagenrücklauf
\t	Steuerzeichen für Tabulator
\b	Steuerzeichen für Backspace

Tabelle 13.4: Maskierung von Steuerzeichen in MySQL-Zeichenketten

Zahlen werden ohne Tausendertrennzeichen notiert. Dezimalzeichen ist der Punkt. In Feldern, die nicht mit unsigned gekennzeichnet sind, dürfen auch negative Werte mit vorangestelltem Minuszeichen vorkommen. Die Notationsform mit Exponentialschreibweise (z. B. 1.234e+8) ist bei Fließkommazahlen erlaubt.

Bestehende Datensätze ändern

Um einen oder mehrere Datensätze in einer Tabelle zu ändern, können Sie das `update`-Statement von SQL benutzen. Wir demonstrieren das Statement wieder anhand unserer Beispieltabelle ***authors***:

```
update website.authors set
      phone="++49 1003 423834",
      address="Schicksalsweg 7, 90001 Gibtsnich"
   where given_name="Rainer" and family_name="Zufall";
```

Ähnlich wie bei der alternativen `insert`-Syntax wird hinter `update <Tabellenname> set` notiert und daran anschließend die gewünschten Änderungen in der Form `Feldname=Wert`.

Entscheidende Bedeutung hat allerdings die where-Klausel im hinteren Teil des Statements. Durch diese Klausel müssen Sie festlegen, welche Datensätze geändert werden sollen. Es werden alle Datensätze geändert, die durch die where-Klausel gefunden werden. Wenn also z. B. nur notiert wäre `where family_name="Zufall"` und es gäbe mehrere gespeicherte Einträge mit diesem Wert, dann würden alle entsprechenden Datensätze mit den angegebenen neuen Werten für die Felder ***phone*** und ***address*** überschrieben.

Wenn also nur ein bestimmter Datensatz geändert werden soll, muss er möglichst eindeutig identifiziert werden. Statt die where-Klausel mit Angaben zu `given_name` und `family_name` zu gestalten, hätten wir, um sicherer zu gehen, auch die folgende Variante wählen können:

```
update website.authors set
      phone="++49 1003 423834",
      address="Schicksalsweg 7, 90001 Gibtsnich"
   where table_key=1;
```

Auf where-Klauseln gehen wir in ***Abschnitt 13.2.4*** noch näher ein.

Da das Feld `table_key` in unserer Beispieltabelle ein Autoincrement-Feld ist, das bei jedem Datensatz garantiert einen anderen Wert hat, lässt sich ein Datensatz darüber natürlich eindeutig identifizieren. Der `table_key` eines gewünschten Datensatzes muss natürlich bekannt sein. Im Zweifelsfall muss er vorher ermittelt werden durch eine Abfrage wie:

```
select table_key from website.authors where given_name="Rainer" and
family_name="Zufall";
```

Hinter dem Schlüsselwort `update` können Sie die Optionen `low_priority` und `ignore` notieren, mit der gleichen Wirkung wie bei `insert`.

Datensätze löschen

Das Löschen von Datensätzen funktioniert nach dem gleichen syntaktischen Muster wie das Ändern. Ein Beispiel:

```
delete from website.authors
   where given_name="Rainer" and family_name="Zufall";
```

Die Löschaktion wird durch `delete from` eingeleitet. Dahinter folgt die Benennung/Adressierung der gewünschten Tabelle. Der oder die Datensätze werden durch eine where-Klausel bestimmt.

Hinter dem Schlüsselwort `delete` sind die optionalen Angaben `low_priority` oder `quick` möglich – `low_priority` mit der bereits bekannten Wirkung und `quick` mit der Wirkung, dass der Löschvorgang sich nicht auf Indexe erstreckt, was den Vorgang beschleunigt.

Werden Datensätze aus Tabellen mit Autoincrement-Feld gelöscht, so wird beim nächsten Neuanlegen eines Datensatzes per Default keine »frei gewordene« Nummer vergeben, sondern die nächsthöhere Nummer des intern von MySQL verwalteten Autoincrement-Zählers. Durch häufiges Löschen und Neuanlegen von Datensätzen werden solche Zähler also schnell erhöht.

Neben dem Löschen einzelner Datensätze besteht manchmal auch der Wunsch, eine komplette Tabelle in ihren jungfräulichen Zustand zurückzuversetzen, also alle Datensätze zu löschen und Autoincrement-Felder ebenfalls auf 0 zurückzusetzen. Dazu dient folgendes Statement (Beispiel):

```
truncate website.authors;
```

Befüllen der Beispieldatenbank

Nachdem wir nun mit den SQL-Statements zum Verwalten von Datensätzen vertraut sind, befüllen wir unsere Übungsdatenbank *website*, indem wir in der Datei *website_create.sql* die Daten von zwei Autoren, einem Stylesheet, einem Template und einer Seite eingeben:

```
insert into website.authors(
      given_name, family_name, mail, phone, address)
   values(
      "Rainer", "Zufall", "rz@example.org", "01003-423834",
      "Schicksalsweg 7, 90001 Gibtsnich");
insert into website.authors(
      given_name, family_name, mail, phone, address)
   values(
      "Anna", "Lyse", "anna@example.org", "010012-123009",
      "Schicksalsweg 7, 90002 Gibtsauchnich");
insert into website.styles(
      css_file, description)
   values(
      "/src/standard.css", "Standard-Layout");
insert into website.templates(
      template_file, description)
   values(
      "/src/standard.tpl", "Standardseiten-Template");
insert into website.templates(
      template_file, description)
   values(
      "/src/error.tpl", "Fehlerseiten-Template");
```

```
insert into website.pages(
      get_name, title, keywords, description, charset,
      author_key, last_mod, style_key, template_key,
      content)
   values(
      "main", "LinkWeb", "Links, Linkverzeichnis, Weblinks",
      "Das Verzeichnis der Linkverzeichnisse", "ISO-8859-1",
      2, "2005-03-22 12:18:00", 1, 1,
      "<h1>LinkWeb</h1>\n<p>Bevor es große Suchmaschinen wie
      Google gab, waren es Linkverzeichnisse, welche die
      zentrale Vernetzung im Web besorgten. Auch heute noch,
      im Zeitalter der Suchmaschinen, gibt es nach wie vor
      gepflegte Linkverzeichnisse, meist auf einen bestimmten
      Themenbereich beschränkt. Dazu kommen durchsuchbare
      Spezialverzeichnisse, etwa für Adressen und
      Telefonnummern, aber auch für Internet-Inhalte jenseits
      des World Wide Web.</p>
      <p><b>LinkWeb</b> ist das Verzeichnis solcher
      Verzeichnisse.</p>
      <h2>Die Kategorien</h2>
      <ul>
      <li>Themenverzeichnisse sind Linkverzeichnisse zu
      bestimmten Fachgebieten.</li>
      <li>Branchenverzeichnisse sind nach Branchen sortierte
      Verzeichnisse von Unternehmen und Geschäften.</li>
      <li>Auskunft-Services sind durchsuchbare Verzeichnisse
      wie Telefonbücher.</li>
      <li>usw.</li>
      </ul>");
```

Die Datei *website_create.sql* ist damit fertig und kann dem MySQL-Client zur Bearbeitung übergeben werden. Das entsprechende Shell-Kommando oder Kommando auf DOS-Ebene lautet:

```
mysql -u [Benutzername] -p < website_create.sql
```

In dem Beispiel wird vorausgesetzt, dass sich *website_create.sql* im aktuellen Verzeichnis befindet und dass *mysql* im Pfad liegt, also von überall aus aufrufbar ist. Sollte das nicht der Fall sein, müssen Sie entsprechende Pfadangaben notieren. Für `[Benutzername]` müssen Sie *root* oder einen anderen Benutzernamen eingeben, der ausreichend Rechte hat, um neue Datenbanken, Tabellen und Datensätze anzulegen.

Wichtig ist der Umleitungsoperator <. Er bewirkt, dass der MySQL-Client, also das Programm mysql, nicht von der Standardeingabe liest, sondern aus der angegebenen Datei. Zuvor wird jedoch zur Eingabe des Passworts für den angegebenen Benutzernamen aufgefordert.

Wir empfehlen Ihnen, weitere Datensätze in die Übungsdatenbank einzufügen, um etwas mehr Material zum Ausprobieren der nachfolgend beschriebenen Datenabfragen zur Verfügung zu haben.

13.2.4 Daten abfragen

Das Suchen und Abfragen von Daten ist wohl die häufigste Datenbankaktion. Dementsprechend mächtig und komplex ist das einzige dafür zuständige SQL-Statement, nämlich `select`. Das Statement bietet auch noch viel mehr als nur das Durchsuchen von Tabellen. Im Grunde ist es ein eigener kleiner Script-Interpreter, der Ausdrücke erkennen und bewerten kann, Funktionen kennt und auch Konstrukte wie einfache Kontrollstrukturen. Es beginnt damit, dass das select-Statement rechnen kann. Wenn Sie den MySQL-Client starten und an seinem Prompt eingeben:

```
select 7*4;
```

Dann erhalten Sie als Ergebnis:

```
+-----+
| 7*4 |
+-----+
|  28 |
+-----+
```

Das select-Statement kennt auch etliche Funktionen. Ein Beispiel:

```
select hex(1000);
```

Damit wird die Dezimalzahl 1000 in einen Hexadezimalwert umgerechnet. Die Ausgabe lautet `3E8`.

Ein anderes Beispiel:

```
select length("Stefan Münz");
```

Das erzeugt die Ausgabe 11, da `length` eine Funktion zum Ermitteln der Länge einer Zeichenkette ist.

Noch ein Beispiel:

```
select if(weekday(current_date()) = 1, 'Dienstag', 'kein Dienstag');
```

In diesem Beispiel wird gleich dreifach verschachtelt: Eine if-Funktion enthält ein einfaches if-else-Konstrukt. Der erste Parameter ist die Bedingung (im Beispiel: `weekday(current_date()) = 1`), der zweite Parameter das, was geschehen soll, wenn die Bedingung wahr ist, und der dritte Parameter das, was geschehen soll, wenn die Bedingung falsch ist. Die Bedingung im Beispiel fragt, ob der Wochentag (ermittelbar durch die Funktion `weekday()`) des aktuellen Datums (ermittelbar durch die Funktion `current_date()`) den Wert 1 hat, was Dienstag entspricht. Abhängig davon wird Dienstag oder kein Dienstag ausgegeben.

Diese Beispiele möchten nur einen ersten Eindruck davon vermitteln, dass `select` weitaus mehr kann als nur einfache Tabellenabfragen. Dennoch möchten wir uns diesen als

Nächstes zuwenden, um davon ausgehend die Möglichkeiten des select-Statements nach und nach auszuloten.

Alle Daten einer Tabelle ermitteln

Dies ist die einfachste Form des select-Statements. Ein Beispiel:

```
select * from website.authors;
```

Durch den Platzhalter * werden alle Felder (Spalten) der Tabelle *authors* in der Datenbank *website* ausgegeben und da keine where-Klausel oder eine andere Beschränkung notiert ist, werden alle Datensätze der Tabelle gefunden und ausgegeben.

Alle Datensätze, aber nur bestimmte Felder ermitteln

Wenn zwar alle gespeicherten Datensätze einer Tabelle ausgegeben werden sollen, jedoch nicht alle Felder, dann können Sie angeben, welche Felder ausgegeben werden sollen:

```
select table_key, given_name, family_name from website.authors;
```

An die Stelle des Platzhalters *, der alle Felder bezeichnet, werden die gewünschten Felder durch Kommata getrennt angegeben.

Nur Datensätze mit bestimmtem Wert in einem Feld ermitteln

Damit beginnt die eigentliche Kunst des »Selektierens«, also genau diejenigen Datensätze zu ermitteln, die gewünscht werden. Im einfachsten Fall sollen alle Datensätze ermittelt und ausgegeben werden, die in einem Feld einen bestimmten Wert haben. Beispiel:

```
select * from website.authors where given_name = "Anna";
```

Zum Filtern bestimmter Datensätze dient die where-Klausel. Sollte der Wert in mehreren Feldern vorkommen, werden mehrere Datensätze gefunden.

Anzahl (gefundener) Datensätze ermitteln mit count(*)

In einigen Fällen möchte man gar keine Daten ermitteln, sondern nur wissen, wie viele Datensätze einem Auswahlkriterium entsprechen. Dies kann beispielsweise auch hilfreich sein, um im Vorfeld einer Datensuche herauszufinden, ob ein Kriterium genau einen Datensatz liefert. Beispiel:

```
select count(*) from website.authors where given_name = "Anna";
```

Anstelle einer Spezifikation, welche Felder ausgegeben werden sollen, wird die Funktion `count()` aufgerufen. Als Parameter wird ihr * übergeben, was so viel wie »Summe aller Ergebnisse« bedeutet.

Wenn Sie keine where-Klausel oder eine andere Einschränkung formulieren, können Sie auf diese Weise auch sehr einfach herausfinden, wie viele Datensätze in einer Tabelle insgesamt gespeichert sind. Beispiel:

```
select count(*) from website.authors;
```

Dies liefert die Gesamtzahl der Datensätze der Tabelle *authors* in der Datenbank *website*.

Möglichkeiten zur Beschränkung auf einen bestimmten Datensatz

Soll nur ein einziger bestimmter Datensatz gefunden werden, so gibt es mehrere Möglichkeiten, dies zu erreichen. Eine Möglichkeit besteht darin, durch eine Und-Verknüpfung von Einschränkungen die Auswahl zu verschärfen. Beispiel:

```
select * from website.authors
   where given_name = "Anna" and family_name = "Lyse";
```

Innerhalb einer where-Klausel können Sie Bedingungen durch `and` verknüpfen.

Optimal ist es jedoch, in einem Feld nach einem Wert zu suchen, der von vorneherein spaltenweit eindeutig ist, wie etwa Personalnummern, Produktschlüssel oder Benutzernamen. Das ist natürlich nur bei Tabellen möglich, die über ein entsprechendes Feld verfügen, und wo beim Befüllen der Datenbank sichergestellt wurde, dass jeder Eintrag in dem Feld eindeutig ist (z.B. mit `insert ignore ...` und Unique-Schlüssel zu der Spalte). Bei Tabellen, die häufig nach einem ganz bestimmten Datensatz durchsucht werden müssen, ist es daher dringend ratsam, ein solches Feld vorzusehen.

Ebenfalls nur einen Datensatz liefern select-Statements mit where-Klauseln, in denen nach einem bestimmten Wert in einem Autoincrement-Feld gesucht wird. In diesem Fall hat MySQL beim Anlegen von Datensätzen dafür gesorgt, dass der Wert spaltenweit eindeutig ist. Nachteil ist, dass die entsprechenden Werte normalerweise nicht in der Weise »bekannt« sind wie etwa ein Personenname oder ein Produktname. In der Regel ist eine solche Abfrage dann sinnvoll und möglich, wenn der Wert eines Autoincrement-Felds aus einem anderen Datensatz geholt wurde. In unserer Beispieldatenbank würde etwa folgendes Statement einen entsprechenden Wert liefern:

```
select style_key from website.pages where get_name = "main";
```

Da das Feld *get_name* mit einem Unique-Index belegt wurde, liefert die where-Klausel `get_name = "main"` auf jeden Fall maximal einen Datensatz. Ermittelt wird aus diesem Datensatz jedoch nur das Feld *style_key*. Dieser Wert kann in einem PHP-Script beispielsweise in einer Variablen gespeichert werden und anschließend zur Abfrage der Tabelle *styles* verwendet werden. Angenommen, das Ergebnis der obigen Abfrage liefert den Wert 1, dann kann *styles* entsprechend befragt werden:

```
select * from website.styles where table_key = 1;
```

Eine letzte Möglichkeit, die Anzahl gefundener Datensätze zu begrenzen, besteht einfach in der Bestimmung einer Ausgabegrenze. Beispiel:

```
select * from website.authors where given_name = "Anna" limit 1;
```

Damit wird nur der erste gefundene Datensatz ausgegeben.

Datensatzbereiche ermitteln mit limit

Der Zusatz `limit` erlaubt es auch, bestimmte Zeilenbereiche (Datensatzbereiche) aus einer Tabelle zu extrahieren. Beispiel:

```
select * from website.authors limit 0,5;
```

Damit werden die ersten fünf Datensätze der Tabelle ermittelt. Die erste Zahl hinter `limit` gibt den Offset an, ab dem wievielten Datensatz ausgegeben werden soll, während die zweite Zahl die Anzahl auszugebender Datensätze von dort nennt. Die Offset-Zählung beginnt bei 0 für den ersten Datensatz, 1 für den zweiten usw.

Limits beziehen sich übrigens nur dann auf den Originalspeicherzustand einer Tabelle, wenn nichts anderes angegeben ist. Wird eine Tabelle jedoch beispielsweise gleichzeitig sortiert, so bezieht sich die `limit`-Angabe auf die sortierte Version der Tabelle und nicht auf die Reihenfolge, in der die Datensätze tatsächlich in der Tabelle stehen.

Sortierte Datenausgabe mit order by

Um Datensätze zu sortieren, können Sie eine Sortierspalte angeben. Es ist nicht zwingend erforderlich, dass für die Sortierspalte ein Index festgelegt wurde. Bei Tabellen mit sehr vielen Datensätzen ist es allerdings deutlich schneller, wenn ein Index für eine Sortierspalte existiert. Beispiel einer einfachen Sortierung:

```
select * from website.pages order by get_name;
```

In diesem Fall werden die Datensätze der Tabelle *pages* aus der Datenbank *website* nach der Spalte `get_name` aufsteigend sortiert. Um eine absteigende Sortierung zu erzwingen, müssen Sie zusätzlich `desc` (für *descending* = absteigend) notieren:

```
select * from website.pages order by get_name desc;
```

Die Sortierungsform wird durch den Spalteninhalt festgelegt. In Zeichenkettenspalten wird alphanumerisch sortiert, in numerischen Spalten numerisch.

Sie können auch nach mehreren Spalten sortieren. Notieren Sie die gewünschten Spalten einfach durch Kommata getrennt. Beispiel:

```
select * from website.pages order by author_key, get_name;
```

Zuerst wird dann nach der erst genannten Spalte sortiert (im Beispiel `author_key`). Gleiche Einträge in der ersten Spalte werden dann nach dem jeweiligen Eintrag in der zweitgenannten Spalte sortiert.

Vergleichsoperatoren für Bedingungen in where-Klauseln

Bislang haben wir nur mit dem Istgleich-Operator verglichen. Beispiel:

```
select * from website.authors where given_name = "Anna";
```

Weitere Vergleichsoperatoren sind < (kleiner als), <= (kleiner oder gleich), > (größer als), >= (größer oder gleich), <> (ungleich). Beispiel:

```
select * from website.pages where table_key >= 10;
```

Oder:

```
select * from website.authors where family_name >= "M";
```

Bei Zeichenkettenfeldern wird dann alles gefunden, was im Feld `family_name` mit M oder einem im Alphabet weiter hinten liegenden Buchstaben beginnt.

Bereichsangaben in where-Klauseln

Bereiche lassen sich zwar auch durch kombinierte Vergleiche wie `>= 10 and < 20` formulieren, doch SQL stellt hierfür auch eine eigene Syntax zur Verfügung. Ein Beispiel:

```
select index_name, mobile_phone from contacts
   where mobile_phone between "+49 170" and "+49 179";
```

Damit werden aus einer Kontakte-Tabelle alle Datensätze gefunden, die im Feld *mobile_phone* einen Eintrag haben, der mit +49 170, +49 171 usw. bis +49 179 beginnt. Der Bereich wird durch `between <Wert> and <Wert>` definiert.

Eine andere Möglichkeit, einen Wertebereich zu bestimmen, ist die Angabe expliziter möglicher Werte. Beispiel:

```
select index_name, mobile_phone from contacts
   where mobile_phone in ("+49 170", "+49 175", "+49 178");
```

Die Abfrage liefert alle Datensätze, bei denen die Einträge im Feld *mobile_phone* mit +49 170, +49 175 oder +49 178 beginnen. Die Syntax einer solchen Werteauswahl beginnt mit `in`, gefolgt von einer beliebigen Anzahl gewünschter Werte in runden Klammern, getrennt durch Kommata.

Beiden Schlüsselwörtern, also `between` und `in`, kann ein `not` vorangestellt werden. Damit wird der Bereich verneint und es werden alle Datensätze gefunden, die nicht im angegebenen Bereich sind.

Abfragen auf leere oder gefüllte Felder

Um alle Datensätze zu ermitteln, die in einem bestimmten Feld keinen oder einen Wert haben, können Sie auf null prüfen. Beispiel:

```
select * from website.pages where description is null;
```

Damit werden alle Datensätze gefunden, bei denen das Feld *description* leer ist. Umgekehrt geht es ebenfalls:

```
select * from website.pages where description is not null;
```

Damit werden alle Datensätze gefunden, bei denen das Feld *description* einen Inhalt hat.

Abfragen mit Wildcards

Es gibt Fälle, in denen nicht nach einem bestimmten Wert oder nach einer Bedingung gesucht werden soll, sondern nach einem Muster. Beispiele:

```
select * from tasks where begin_date like "2005-06-__";
select index_name, work_phone from contacts where work_phone like "089%";
```

In beiden Fällen wird in der where-Klausel anstelle des Gleichheitsoperators das Schlüsselwort `like` notiert. Im abgefragten Wert kann dann mit Wildcards gearbeitet werden. Als Wildcardzeichen stehen zur Verfügung der Unterstrich (_) als Platzhalter für ein einzelnes Zeichen und das Prozentzeichen (`%`) für eine beliebige Anzahl von Zeichen an der betreffenden Stelle, inklusive kein Zeichen.

Im ersten der obigen Beispiele werden alle Datensätze einer Aufgabenliste ermittelt, deren Startdatum im Juni 2005 liegt. Durch zwei Unterstriche an der Stelle, wo die zweistellige Tageszahl steht, werden alle dort stehenden Werte gefunden.

Im zweiten Beispiel werden alle gespeicherten Telefonnummern gefunden, die mit 089 beginnen.

Die Sonderbedeutung von Prozentzeichen und Unterstrich ist auch der Grund, weswegen diese beiden Zeichen maskiert werden müssen, falls sie als normale Zeichen gesucht werden sollen.

Das Arbeiten mit Wildcards kann auch sinnvoll sein, um Schreibweisen von Namen zu berücksichtigen. So findet `M%r` neben *Meier* auch *Meyer*, *Mayer* und *Mayr*, allerdings auch *Mindelsheimer*.

Das Arbeiten mit `like` und Wildcards ist zwar bequem, verlangsamt allerdings die Ausführungsgeschwindigkeit der Suche, vor allem dann, wenn eine Spalte durchsucht wird, auf der kein Index liegt.

Abfragen mit regulären Ausdrücken

Auch das Suchen mit regulären Ausdrücken ist möglich. Ein Beispiel:

```
select * from contacts where index_name regexp ".*, .* \(.*\)"
```

Dieses Statement findet alle Datensätze, bei denen der Eintrag im Feld *index_name* dem Muster *Irgendwas, Irgendwas (Irgendwas)* entspricht, also z.B. *Meier, Julia (Superleben GmbH)*.

Anstelle des Gleichheitsoperators wird in der where-Klausel das Schlüsselwort `regexp` notiert. Innerhalb der Hochkommas folgt dann der reguläre Ausdruck (ohne Begrenzerzeichen wie /). Einzelheiten zu regulären Ausdrücken siehe ***Abschnitt 12.3.9***.

Auch für Abfragen mit regulären Ausdrücken gilt: Sie sind sehr präzise, doch bei großen Tabellen auch eher langsam.

Ausgereifte Suchen durch logische Verknüpfungen in where-Klauseln

Die Verknüpfung zweier Bedingungen durch `and` haben wir bereits kennen gelernt. Dabei werden – wie bei logischem Und üblich – nur Datensätze gefunden, die beiden Bedingungen genügen. Sie können jede Art von Vergleich, Bereichsdefinition usw. auf diese Weise verknüpfen und Sie können selbstverständlich auch größere Ketten von and-Verknüpfungen bilden. Beispiel:

```
select * from contacts
   where given_name < "M" and
         gender = "W" and
         mobile_phone is not null;
```

Diese Abfrage ermittelt aus einer Tabelle mit Kontaktdaten alle Kontakte, bei denen der Familienname in der ersten Hälfte des Alphabets vorkommt (`given_name < "M"`), die vom Geschlecht her weiblich sind (`gender = "W"`) und zu denen eine Handynummer gespeichert ist (`mobile_phone is not null`).

Verknüpfungen durch `and` schränken die Auswahl tendenziell ein. Um die Auswahl tendenziell zu erweitern, steht die Verknüpfung durch `or` (inklusives Oder) zur Verfügung. Dabei werden alle Datensätze gefunden, die wenigstens einer der verknüpften Bedingungen genügen. Beispiel:

```
select * from contacts
   where family_name < "E" or family_name >= "S";
```

Damit werden alle Datensätze in den Kontaktdaten gefunden, bei denen der Nachname mit A bis D beginnt oder mit S bis Z.

Sie können auch `and` und `or` mischen. Dabei hat `and` eine stärkere Bindung. Ein Beispiel:

```
select * from contacts
   where family_name < "E" and length(family_name) < 6
         or family_name >= "S";
```

Damit werden alle Datensätze gefunden, bei denen der Nachname mit A bis D beginnt und weniger als 6 Zeichen lang ist, sowie alle Nachnamen, die mit S bis Z beginnen, bei denen die Länge aber egal ist. Grund ist, dass die and-Verknüpfung vorrangig ausgewertet wird.

Gruppierte Datenausgabe mit Gruppierungsfunktionen und group by

Die Funktion `count()` haben wir bereits kennen gelernt. Sie zählt gefundene Datensätze. Angenommen, es wird Folgendes notiert:

```
select gender, count(*) from contacts   # nicht vollständig!
```

Es soll das Geschlecht ausgewählt werden und es soll alles gezählt werden – eine logisch nicht sinnige Zusammenstellung von Wünschen. Daher ist das Statement in dieser Form auch noch nicht vollständig. Es bietet sich jedoch an, die Angabe so zu interpretieren, dass gezählt werden soll, wie oft jeder im Feld *gender* vorkommende Eintrag insgesamt vorkommt. Da das Feld *gender* sinnvollerweise ein enum-Feld ist, also z. B. nur die beiden Einträge »M« und »W« zulässt, würde ermittelt, wie viele Männer und wie viele Frauen in der Kontaktetabelle gespeichert sind. Diese Form der Zusammenfassung wird als Gruppierung bezeichnet und `count()` ist eine der Funktionen, die zum Erzeugen von gruppierten Ergebnissen verhelfen können. Damit das Statement diesen Wunsch erfüllt, muss es im Beispiel um folgende Angabe erweitert werden:

```
select gender, count(*) from contacts group by gender;
```

Durch die group-by-Klausel wird angewiesen, nach welchem Feld gruppiert werden soll. Der zugewiesene Wert einer jeden ermittelten Gruppe wird durch eine gruppenbildende Funktion ermittelt. Ähnlich ist das folgende Statement:

```
select gender, count(distinct gender) from contacts
       group by gender;
```

Die Variante `count(distinct gender)` zählt im Unterschied zu `count(*)` keine Leerfelder.

Weitere gruppierungsfähige Funktionen, die wie `count()` verwendbar sind, aber vor allem auf numerische Werte anwendbar sind, sind `sum()` für Summenwerte, `avg()` für Mittelwerte, `min()` für Minimalwerte, `max()` für Maximalwerte oder `std()` für Standardabweichungen. Wenn Sie diese Funktionen ohne group-by-Klausel verwenden, beziehen sie sich auf alle Einträge einer Spalte. Beispiel:

```
select avg(points) from exercise_results;
```

Dieses Statement ermittelt aus einer angenommenen Tabelle mit Übungsergebnissen den Mittelwert aus allen Werten der Spalte *points*.

Hinter einer group-by-Klausel können Sie einschränkende Bedingungen ähnlich wie in einer where-Klausel formulieren. Beispiel:

```
select given_name, count(*) from contacts
       group by given_name having length(given_name) < 6;
```

Damit werden alle Vornamen mit der Anzahl ihres Vorkommens aufgelistet, wobei die Auswahl jedoch auf Vornamen eingeschränkt wird, die weniger als sechs Buchstaben haben. Die zusätzliche Einschränkungsklausel wird durch `having` eingeleitet. Verwenden Sie `having` jedoch nur in Zusammenhang mit `group by` und nicht an Stelle von `where`.

Aliasnamen für Ausgaben

Angenommen, Sie möchten die Ausgabe des obigen Statements nach dem Anzahl des Vorkommens sortieren. Die Spaltenüberschrift der Spalte mit den gezählten Datensätzen heißt `count(*)`. Dies ist jedoch kein gültiger Name, nach dem sortiert werden kann. Um das Problem zu lösen, können Sie mit Aliasnamen arbeiten. Das Statement kann folgendermaßen erweitert werden, um eine Sortierung wie gewünscht zu erreichen:

```
select given_name, count(*) as count from contacts
       group by given_name having length(given_name) < 6
       order by count desc;
```

Dabei wurde `count(*)` zu `count(*) as count` erweitert. Zweck dieser Erweiterung ist es, der durch `count(*)` erzeugten Spalte in der Ausgabetabelle einen ordentlichen Namen zu geben – er wird als Aliasname bezeichnet. Im Beispiel lautet er `count`. Genauso gut könnte er aber auch `Anzahl` oder sonst irgendwie lauten.

Am Ende des Statements erweist sich der Aliasname als nützlich. Dort wird er verwendet, um die Ausgabetabelle absteigend nach der Spalte mit dem Namen `count` zu sortieren. Das Beispiel ermittelt zu allen in der Tabelle vorkommenden Vornamen, wie oft diese vorkommen, beschränkt die ganze Geschichte auf Vornamen mit weniger als sechs Buchstaben und sortiert das Ergebnis nach Anzahl des Vorkommens absteigend.

13.2.5 Datenabfragen über mehrere Tabellen

In unserer Beispieldatenbank *website* haben wir einige Felder vorgesehen, die feste Verknüpfungen zwischen einzelnen Datensätzen speichern können. Angenommen, wir wählen die Daten einer Seite mit folgendem SQL-Statement aus:

```
select * from website.pages where get_name = "main";
```

Dann erhalten wir unter anderem Werte zu den Feldern *author_key*, *style_key* und *template_key*. Diese Felder speichern numerische Werte, die »harte Links« auf Felder des Namens *table_key* in den Tabellen *authors*, *styles* und *templates* darstellen. Im Beispiel des Datensatzes, den uns die where-Klausel `get_name = "main"` liefert, sind folgende Werte gespeichert:

author_key = 2

style_key = 1

template_key = 1

Das bedeutet etwa im Fall *author_key*: Autor dieser Seite ist diejenige Person, die in der Tabelle *authors* im Feld *table_key* den Wert 2 hat. Von diesem Zusammenhang weiß MySQL allerdings nichts. Er existiert lediglich in unserem Datenbankkonzept.

Um autorenspezifische Daten zur ausgewählten Seite zu ermitteln, könnte ein zweites select-Statement notiert werden:

```
select * from authors where table_key = 2;
```

Angenommen jedoch, es sollen zu jeder gespeicherten Webseite deren get-Name, ihr Titel sowie Vor- und Zuname des Autors dieser Seite ausgegeben werden. Der Abfragewunsch ist ja durchaus legitim, da jede Webseite einen Autor hat. Die zugehörigen Daten stehen jedoch in zwei verschiedenen Tabellen. SQL erlaubt es, Tabellen zu verknüpfen, sofern Felder existieren, die solche Verknüpfungen erlauben.

In unserem Beispiel könnte dies wie folgt aussehen:

```
select pages.get_name, pages.title, authors.given_name, authors.family_name from
pages, authors
   where pages.author_key = authors.table_key;
```

Da wir nun aus zwei Tabellen selektieren, müssen die Tabellen bei jedem gewünschten Feldnamen mit angegeben werden, also `pages.get_name` statt einfach nur `get_name`. Hinter dem Schlüsselwort `from` wird nicht mehr nur eine Tabelle genannt, sondern, durch Komma getrennt, beide betroffenen Tabellen.

Fehlt allerdings noch die Anweisung, wie die Zuordnung zwischen beiden Tabellen aussehen soll. Dies geschieht in der where-Klausel. Bedingung zur Auflistung eines tabellenübergreifenden Datensatzes mit den vier angegebenen Feldern soll sein, dass `pages.author_key` und `authors.table_key` übereinstimmen.

Um alles weitere kümmert sich der SQL-Interpreter. Die Ausgabetabelle enthält wie gewünscht die vier Felder aus den beiden verschiedenen Tabellen.

Left Joins

Im obigen Beispiel wurde die where-Klausel bemüht, um die Verknüpfungsbedingung zwischen beiden Tabellen zu formulieren. Da Verknüpfungsbedingungen jedoch vom Typ her etwas anderes sind als Selektionsbedingungen innerhalb einer Tabelle, gibt es eine syntaktische Variante, um Verknüpfungen herzustellen. Das gleiche Ergebnis wie das oben vorgestellte SQL-Statement erzielt das folgende Statement:

```
select pages.get_name, pages.title, authors.given_name, authors.family_name from
pages
   left join authors on pages.author_key = authors.table_key;
```

Hinter `from` werden nun nicht mehr beide betroffenen Tabellen aufgezählt. Stattdessen wird zunächst eine Basistabelle bestimmt. In unserem Fall ist das die Tabelle *pages*, denn wir möchten ja Daten zu jeder gespeicherten Seite ausgeben (und nicht zu jedem gespei-

cherten Seitenautor). Mit `left join` wird dieser Basistabelle die Zweittabelle zugewiesen, aus der zusätzliche Daten geholt werden sollen. Hinter dem Schlüsselwort `on` folgt abschließend die Verknüpfungsbedingung.

Hinter der Verknüpfungsbedingung kann eine where-Klausel folgen, in der die Datensatzauswahl eingeschränkt wird.

Im obigen Beispiel wird bei der Verknüpfungsbedingung auf den Wert in zwei Feldern namens *author_key* und *table_key* geprüft. Die Datenbank hätte jedoch auch so entworfen werden können, dass das Feld *table_key* jeweils den Namen der Tabelle trägt, also bei *pages* etwa *page_key* und bei *authors author_key*. In der Tabelle *pages* könnte das Bezugsfeld für die Tabelle *authors* dann durchaus ebenfalls *author_key* benannt werden. Die zu vergleichenden Felder hätten also in beiden Tabellen die gleichen Namen. Für diesen Fall stellt SQL die using-Klausel bereit. Beispiel für diese Variante:

```
select pages.get_name, pages.title, authors.given_name, authors.family_name from
pages
   left join authors using author_key;
```

Neben `left join` gibt es noch diverse andere Formen von Joins. Doch die meisten davon sind in MySQL nur Varianten der hier beschriebenen beiden Formen.

13.2.6 Tipps für effiziente Datenbankabfragen

Jede Datenbankabfrage benötigt Ressourcen wie Festplattenzugriffe, Arbeitsspeicher und CPU-Zeit. Basiert eine öffentliche Website wesentlich auf Datenbankinhalten, so entsteht bei vielen gleichzeitigen Benutzern schnell eine enorme Last am Server-Rechner. Viele einfache Root-Server, die nicht mit mehreren Gigabyte Arbeitsspeicher ausgestattet sind und ansonsten auch nur mit handelsüblicher PC-Ausrüstung, gehen deshalb bei etwas größer und beliebter werdenden Projekten schnell in die Knie.

Von der ausreichenden Hardwaredimensionierung abgesehen, können Sie als Webentwickler jedoch einiges tun, um den Ressourcenverbrauch günstig zu beeinflussen und Performance zu gewinnen. Gerade bei häufig benutzten Datenbankabfragen empfiehlt es sich, Tuning-Maßnahmen zu ergreifen. Ein paar davon stellen wir hier vor.

Arbeiten mit Indexen

Wenn eine Tabelle sehr viele Datensätze enthält, z.B. Einzelspielergebnisse in einer Online-Spiele-Site, und wenn häufig einzelne dieser Datensätze gesucht werden müssen (intensives Arbeiten mit where-Klauseln), dann sollten alle Spalten, die in Bedingungen vorkommen, einen Index erhalten. Es ist durchaus möglich, nachträglich einen Index zu erstellen:

```
alter table game_results add index(life_points);
```

Damit fügen Sie der Tabelle *game_results* einen Index für das Feld bzw. die Spalte *life_points* hinzu. Anstelle von `add index` können Sie auch `add primary` oder `add unique`

angeben, um einen der beiden anderen Indextypen hinzuzufügen. Ein Index vom Typ *primary* darf jedoch nur für eine Spalte der Tabelle existieren und ein Index vom Typ *unique* ist nur auf Spalten anwendbar, bei denen jeder Datensatz garantiert einen anderen Wert hat.

Indexe können MySQL allerdings auch ausbremsen – nämlich dann, wenn tendenziell viele oder alle Datensätze aus einer Tabelle ermittelt werden. Während der Zugriff auf einen bestimmten Datensatz etwa bei 100.000 Datensätzen mithilfe eines Indexes ungleich schneller ist, als wenn die Tabelle Datensatz für Datensatz durchsucht werden müsste, so verhält es sich in dem Fall, dass alle 1000 Datensätze einer Tabelle gelesen werden sollen, genau umgekehrt. In diesem Fall geht das sequentielle Lesen schneller, als wenn jeder einzelne Datensatz über die Leselogik der Indexierung ermittelt wird.

Indexe sind also für Tabellenspalten angesagt, die

- in Vergleichsoperationen in einer where-Klausel für eine starke Selektion von Datensätzen sorgen sollen,
- mit `order by` sortiert werden sollen,
- nach Minimal- oder Maximalwerten durchsucht werden sollen, also mit den Funktionen `min(Spaltenname)` oder `max(Spaltenname)`,
- für Joins mit anderen Tabellen benötigt werden.

Mehrspaltenindexe

Angenommen, Sie benötigen folgendes SQL-Statement häufig und arbeiten innerhalb davon einer where-Klausel mit logischem Und:

```
select * from exercise_results where user_name = "fab_mei_002" and exercise =
"A17b";
```

In diesem Fall kann die Abfrage durch einen so genannten Mehrspaltenindex über beide betroffenen Spalten beschleunigt werden. Die Definition des betreffenden Indexes lautet im Beispiel:

```
alter table exercise_results add index(user_name, exercise);
```

Der Index speichert dann alle Kombinationen beider Spalten und beschleunigt dadurch den Zugriff bei der obigen select-Abfrage.

Teilindexe

Manchmal genügt es, nur die ersten n Zeichen von Zeichenketten zu indexieren. Dadurch wird der Index kleiner und schneller. Beispiel:

```
alter table contacts add index(given_name(6));
```

Damit wird ein Index für eine Spalte namens *given_name* erzeugt, wobei aber nur die ersten sechs Zeichen indexiert werden. Für die meisten Vornamen genügt dies bereits, auch wenn das Feld selber wegen einiger langer oder zusammengesetzter Namen deutlich länger definiert werden musste.

Optionen für select-Statements

Hinter dem einleitenden Schlüsselwort `select` können eine Reihe von Optionen notiert werden, die das Verhalten des DBMS beeinflussen und für bessere Performance sorgen können:

- `high_priority` weist dem Abfrageprozess den Vorrang zu, falls andere Prozesse gleichzeitig andere Aktionen mit der Tabelle durchführen, wie etwa Schreiboperationen.
- `sql_buffer_result` bewirkt, dass das Abfrageergebnis in eine temporäre Tabelle geschrieben wird. Dadurch kann MySQL die Tabellensperre für andere Abfragen schneller wieder lösen. Wenn viele Clients gleichzeitig eine Tabelle abfragen, wie es bei einer Website mit Datenbankanbindung oft der Fall ist, kann dies insofern zu einer Performance-Verbesserung führen, als ein Client-Prozess nicht so lange warten muss, falls die Tabelle gerade durch einen anderen Client-Prozess gesperrt ist.
- `sql_big_result` kann bei select-Statements angegeben werden, bei denen bereits von vorneherein feststeht, dass eine Menge Ausgabezeilen erzeugt werden. Die Option hilft dem MySQL-eigenen Optimizer, die temporäre Zwischenspeicherung von Daten so zu organisieren, wie es für große Ergebnismengen am besten ist.

  MySQL unterstützt daneben auch die gegenteilige Option `sql_small_result`, die für sehr kleine Ergebnismengen (z. B. nur ein Datensatz) beim Optimieren hilft.
- `straight_join` kann in manchen Fällen von Abfragen über mehreren Tabellen helfen, die Verknüpfung zu optimieren.

Ergebnismengen reduzieren

Viel Performance wird häufig einfach dadurch verschenkt, dass Daten abgefragt werden, die gar nicht benötigt werden. Wenn Sie aus einer Tabelle mit Kontakten nur Namen und Telefonnummer benötigen, dann listen Sie hinter `select` auch nur die betreffenden Felder auf, statt das bequemere `select * ...` zu notieren.

Umfangreiche Tabellen teilen

Websites, die Bereiche für registrierte Benutzer bereithalten und die registrierten Benutzer in einer Datenbank speichern, halten die Anmeldedaten der Benutzer, also Benutzername und Passwort, oftmals in der gleichen Tabelle vor wie die Kontaktstammdaten der Benutzer, also Adressen, Telefonnummern, Geburtstage usw. Das ist zwar insofern logisch, als Benutzername und Passwort zu den Benutzerdaten gehören. Es ist jedoch nicht sehr performant, da bei jedem Anmeldevorgang eines Benutzers die große Kontaktetabelle bemüht werden muss. In diesem Fall kann es die Geschwindigkeit erhöhen, aus der einen Tabelle zwei zu machen, nämlich eine für Benutzernamen und Kennwort und eine andere

für die Kontaktdaten. Da der Benutzername normalerweise eindeutig sein muss, kann er in beiden Tabellen als Spalte mitgeführt werden und als Verknüpfungsfeld zwischen den beiden Tabellen dienen.

Abfrageergebnisse MySQL-unabhängig cachen/speichern

MySQL hat seine eigenen Techniken, um Abfragen zu cachen. Dennoch ist es manchmal performanter, von MySQL unabhängige Möglichkeiten zu nutzen. Denn so kann beispielsweise die Anzahl erforderlicher Abfragen schon im Vorfeld reduziert werden.

Viele Websites, die über PHP auf MySQL-Datenbanken zugreifen, arbeiten beispielsweise mit Sessions. Da ist es eine gute Idee, gleich zu Beginn einer Session einige benötigte Daten aus der Datenbank zu sammeln und in den Sessiondaten zu speichern. So lässt sich vermeiden, dass bei jedem Seitenwechsel die gleichen Datenbankabfragen erneut gestartet werden müssen. Die Sessiondaten werden dadurch zwar etwas umfangreicher, doch in der Regel ist das günstiger als ein Vielfaches an Datenbankzugriffen.

Bei Tabellen, die sowohl mit Schreib- als auch mit Lesevorgängen stark frequentiert werden, kann es unter Umständen sinnvoll sein, in regelmäßigen Abständen eine temporäre Kopie zu erzeugen und die häufigen Lesezugriffe nur auf diese Kopie zu lenken. Das regelmäßige Erzeugen der Kopie kann durch einen cron-Job übernommen werden.

Als Beispiel nehmen wir eine Tabelle namens *exercise_results* an, in der alle Übungsergebnisse von Benutzern eines Online-Learning-Systems gespeichert werden. Bei vielen fleißigen Benutzern müssen laufend, also fast im Sekundentakt, neue Ergebnisse in diese Tabelle geschrieben werden. Andererseits wird die Tabelle auch häufig ausgelesen, da laufend bislang erreichte Übungsergebnisse verlangt werden. Da die Schreibvorgänge viel Performance schlucken und Tabellensperren verursachen, werden die Abfragen stark ausgebremst. Um dafür mit einer regelmäßig aktuellen Lesekopie der Tabelle zu arbeiten, müssen Sie folgende Schritte ausführen:

1. Erzeugen Sie mit `create table()` die Lesetabelle. Verwenden Sie dazu exakt den gleichen SQL-Code wie beim Erzeugen der Originaltabelle. Vergeben Sie einen sinnvollen Namen. Wenn die Originaltabelle *exercise_results* heißt, bietet sich ein Tabellenname wie *exercise_results_for_select* an.

2. Probieren Sie, auf Systemebene, also je nach Umgebung auf einer Linux-Shell oder in einem DOS-Fenster, den MySQL-Client im Batch-Betrieb nacheinander mit folgenden beiden Kommandos zu starten:

```
mysql - e "truncate online_learning_db.exercise_results_for_select";
mysql - e "insert into online_learning_db. exercise_results_for_select select
* from online_learning_db. exercise_results";
```

   In den Beispielen ist `online_learning_db` der Name der betroffenen Datenbank. Das erste Statement löscht alle Datensätze aus der Lesetabelle und setzt diese zurück. Das zweite Statement benutzt die von MySQL unterstützte Syntax `insert into <Tabelle>`

`select ... from ...`. Damit lassen sich Tabellen oder Teile davon in andere Tabellen kopieren.

Überprüfen Sie anschließend, ob die beiden Batch-Kommandos für den MySQL-Client erfolgreich waren. Werfen Sie dazu einen Blick in die erzeugte Lesetabelle.

3. Notieren Sie die beiden Batch-Aufrufe des MySQL-Clients entweder direkt in einer crontab-Datei (unter Linux/Unix) oder notieren Sie die beiden Aufrufe in einem Shellscript, das Sie dann in der crontab-Datei angeben.

   Die Ausführfrequenz können Sie in der crontab-Datei minutengenau justieren. Je nach benötigter Aktualität der Daten sind sinnvolle Werte »alle 5 Minuten« bis »alle 60 Minuten«. In jedem Fall entlastet das jedoch die Originaltabelle erheblich.

Tabellen reorganisieren

Wenn aus Tabellen häufig Daten gelöscht und wieder neue Daten hinzugefügt werden, ist es von Zeit zu Zeit sinnvoll, MySQL die Tabelle reorganisieren zu lassen. Dies ist ein Optimierungsvorgang, der künftige Zugriffe beschleunigen hilft. Das SQL-Statement dazu lautet einfach:

```
repair <Tabellenname>
```

Bei ständig weiterlaufendem Betrieb können Sie dieses Kommando ebenfalls regelmäßig als cron-Job ausführen lassen. Rufen Sie dazu als auszuführendes Kommando wie weiter oben gezeigt den MySQL-Client im Batch-Betrieb auf. Beispiel:

```
mysql - e "repair website.pages";
```

Bei einer Reorganisation genügt ein täglicher bis wöchentlicher Lauf, am besten zu betriebsarmen Zeiten, etwa in den frühen Morgenstunden.

13.2.7 InnoDB-Tabellen

Der klassische Tabellentyp ISAM ist die Standardmethode, die MySQL unterstützt, allerdings in der erweiterten Variante ***MyISAM***. ISAM steht für ***Indexed Sequential Access Method*** und bedeutet einfach, dass ein Zugriff auf gespeicherte Datensätze sowohl sequentiell als auch über Indexe möglich ist.

Die Daten von eingerichteten Datenbanken verwaltet MySQL unterhalb eines konfigurierbaren Verzeichnisses, meistens ***data*** mit Namen. Für jede Datenbank wird ein Unterverzeichnis mit dem Datenbanknamen angelegt. Für jede MyISAM-Tabelle einer Datenbank existieren unterhalb davon zumindest eine Datei ***Tabellenname.frm*** mit den Tabellendefinitionen sowie eine Datei ***Tabellenname.myd*** mit den Daten der Tabelle und eine Datei ***Tabellenname.myi*** mit Indexen.

Seit der Zwischenversion 3.23.34a unterstützt MySQL jedoch auch den neueren InnoDB-Tabellentyp. Dieser hat gegenüber MyISAM folgende Vorteile:

- ***InnoDB ist transaktionssicher.*** Das bedeutet, jeder Vorgang beim Zugriff auf die Tabelle wird protokolliert. Wird eine Transaktion nicht abgeschlossen, sei es durch Benutzerabbruch oder durch einen Systemabsturz, wird der ursprüngliche Tabellenzustand wiederhergestellt (diese Wiederherstellung wird auch als ***Rollback*** bezeichnet).
- ***InnoDB kann Tabellen auf Zeilenebene sperren.*** Das ist ein großer Vorteil, wenn Tabellen intensiv bearbeitet werden. Zwei Client-Prozesse können gleichzeitig Daten in die Tabelle schreiben, solange es sich um unterschiedliche Zeilen handelt. Bei MyISAM-Tabellen muss ein Client-Prozess warten, bis die gesamte Tabelle wieder zum Schreiben freigegeben ist.

Die Datensicherheit wird also erhöht, ebenso die Performance bei häufigen Schreibvorgängen. Es sollen jedoch auch nicht die Nachteile verschwiegen werden:

- ***InnoDB kennt keine 1:1-Zuordnung zwischen Datei und Tabelle.*** Die Datenhaltung von InnoDB erfolgt völlig anders. Es gibt reservierte Speicherdateien, die zudem sehr groß werden können. Darin verwaltet InnoDB seine Tabellen nach einer internen Logik, ähnlich wie ein Betriebssystem sein Swap-File verwendet. Es lassen sich daher auch keine Dateioperationen mehr auf bestimmte Tabellen anwenden (z. B. Datensicherungen von Hand).
- ***InnoDB benötigt mehr Performance beim Lesen.*** Das Auslesen von Tabellendaten ist Erfahrungsberichten zufolge langsamer als bei MyISAM-Tabellen auf der gleichen Maschine.

MyISAM eignet sich also bei Website-Datenbanken eher, wenn Tabellen vorwiegend ausgelesen werden müssen und nur selten Schreibvorgänge stattfinden. Wenn dagegen Tabellen durch Lösch-, Update- und Hinzufüge-Operationen ständig verändert werden, ist InnoDB die geeignetere Wahl. Beide Tabellentypen lassen sich übrigens auch mischen.

Damit InnoDB mit MySQL verfügbar ist, muss MySQL unter Linux/Unix beim Installieren mit der mit `./configure --with-innodb` konfiguriert worden sein. In der Konfigurationsdatei ***my.ini*** oder ***my.cnf***, die bei Linux häufig unter ***/etc*** und bei Windows im Windows-Verzeichnis liegt, sollten einige InnoDB-spezifische Direktiven eingetragen und konfiguriert werden, um das Verhalten von InnoDB zu optimieren. Die entsprechenden Direktiven werden in der MySQL-Dokumentation genau beschrieben. Besonders darauf achten müssen Sie, dass genügend Arbeitsspeicher und Festplattenspeicher für InnoDB reserviert wird, da es andernfalls zu Problemen kommen kann.

Eine InnoDB-Tabelle kann, wenn alle Voraussetzungen erfüllt sind, mit folgender Syntax angelegt werden:

```
CREATE TABLE <Tabellenname> (<Felddefinitionen>) TYPE = InnoDB;
```

Die Angabe zum Tabellentyp wird am Ende der Definition notiert. Da MyISAM der Default-Typ ist, ist nur die abweichende Angabe `type = InnoDB` nötig.

Falls Sie bestehende MyISAM-Tabellen in InnoDB-Tabellen überführen möchten, gehen Sie wie folgt vor:

1. Erzeugen Sie mit `create table neuer_name() type = InnoDB` die gewünschte neue Tabelle im InnoDB-Format. Verwenden Sie dazu exakt den gleichen SQL-Code wie beim Erzeugen der bestehenden MyISAM-Originaltabelle. Vergeben Sie für die Tabelle einen anderen Namen als den der bestehenden Tabelle.
2. Kopieren Sie den Inhalt der bestehenden MyISAM-Tabelle in die neue InnoDB-Tabelle:

```
insert into neuer_name select * from alter_name;
```

3. Überprüfen Sie mithilfe geeigneter select-Statements auf die neue InnoDB-Tabelle, ob die Daten dort wie gewünscht verfügbar sind.
4. Löschen Sie die MyISAM-Tabelle:

```
drop table alter_name;
```

5. Benennen Sie die neue Tabelle so um, dass sie den Namen der früheren Tabelle hat:

```
rename neuer_name to alter_name;
```

Auf InnoDB-Tabellen sind ansonsten die gleichen SQL-Statements anwendbar wie auf MyISAM-Tabellen. Solange sich MySQL im voreingestellten Auto-Commit-Modus befindet, wird jedes einzelne SQL-Statement als eine Transaktion betrachtet. Das InnoDB-Modell verfolgt jede einzelne Transaktion und kann mithilfe des so genannten ***Rollback*** fehlgeschlagene Transaktionen rückgängig machen. Falls Sie mehrere SQL-Statements innerhalb einer Transaktion zusammenfassen möchten, ist das wie im folgenden Beispiel möglich:

```
begin;
select dies, das from tabelle_1 where dies = 1 and das = 2;
update tabelle_2 set dort = 2 where hier = 1;
commit;
```

Durch das SQL-Kommando `begin` werden alle nachfolgenden SQL-Kommandos bis zum Kommando `commit` als eine Transaktion behandelt.

13.2.8 Benutzer- und Rechteverwaltung

Ein MySQL-System auf einem Server-Rechner sollte so eingerichtet werden, dass die Benutzer des Systems auch innerhalb des DBMS die Rechte haben, die sie benötigen, aber nicht so viele Rechte, dass sie Schaden an Stellen anrichten können, mit denen sie nichts zu tun haben.

Nach einer ordentlichen Anfangseinrichtung von MySQL (bei Windows durch den Installer, bei Linux/Unix durch Ausführung des Shellscripts ***mysql_install_db***) existiert ein Benutzer root. Die Anmeldung am MySQL-Client ist möglich durch:

```
mysql -u root -p
```

Es erfolgt die Passwortabfrage für den Benutzer *root*, sofern bereits ein Passwort eingerichtet wurde.

Als Benutzer *root* verfügen Sie über sämtliche Rechte innerhalb von MySQL. Als Benutzer mit diesen Rechten können Sie andere Benutzer einrichten, ändern und löschen. Für jeden Benutzer können Sie sehr detailliert festlegen, welche Rechte dieser Benutzer haben soll. All das findet in der mitgelieferten Datenbank *mysql* statt. Wechseln wir zu dieser Datenbank, um damit zu arbeiten:

```
use mysql;
```

Diese Datenbank hat folgenden Aufbau: Es gibt eine zentrale Tabelle namens *user*, in der gespeichert ist, welche Benutzer der hier laufende MySQL-Server kennt und welche datenbankübergreifenden, globalen Rechte diese Benutzer haben. Ferner gibt es die Tabellen *db*, *host*, *tables_priv* und *columns_priv*, in denen für jeden in *user* definierten Benutzer weitere Detailrechte bis hinunter auf Tabellen- und Spaltenebene definiert werden können.

Betrachten wir zunächst die Tabelle *user*, um zu sehen, welche Benutzer eingerichtet sind, welche Rechte es in dieser Tabelle gibt und welche Rechte welchen Benutzern zugeordnet sind:

```
select * from user;
```

Jeder Datensatz der Tabelle repräsentiert einen Benutzer.

Das erste Feld, *Host*, gibt den Hostnamen des Rechners an, auf dessen MySQL-Installation sich der Benutzereintrag bezieht. Das zweite Feld, *User*, enthält den Benutzernamen, mit dem der Benutzer z. B. den MySQL-Client aufrufen kann (`mysql -u Benutzername`). Das dritte Feld, *Password*, enthält eine hash-verschlüsselte Form des Benutzerpassworts. Das Passwort ist aus dem Hash-Wert nicht ableitbar, wohl aber ist die neuerliche Anwendung des Hash-Algorithmus auf ein vom Benutzer eingegebenes Passwort dazu geeignet, um zu überprüfen, ob das eingegebene Passwort korrekt ist.

Die nachfolgenden Felder (Spalten) der Tabelle *user* stellen jeweils eine Berechtigung dar. Das entsprechende Recht ist einem Benutzer entweder erteilt (*Y*) oder nicht (*N*). Nachfolgende Tabelle listet die Rechte und ihre Bedeutung auf:

Spalte	Recht	Bedeutung
Select_priv	`select`	Der Benutzer darf Datensätze aus Tabellen auslesen (*Y*) oder nicht (*N*).
Insert_priv	`insert`	Der Benutzer darf Datensätze in Tabellen einfügen (*Y*) oder nicht (*N*).
Update_priv	`update`	Der Benutzer darf Datensätze in Tabellen ändern (*Y*) oder nicht (*N*).

Tabelle 13.5: Ein-/ausschaltbare Berechtigungen für Benutzer

Spalte	Recht	Bedeutung
Delete_priv	delete	Der Benutzer darf Datensätze aus Tabellen löschen (*Y*) oder nicht (*N*).
Create_priv	create	Der Benutzer darf Datenbanken, Tabellen oder Indexe neu anlegen (*Y*) oder nicht (*N*).
Drop_priv	drop	Der Benutzer darf Datenbanken oder Tabellen löschen (*Y*) oder nicht (*N*).
Reload_priv	reload	Der Benutzer darf den MySQL-Server anweisen, die Berechtigungstabellen neu einzulesen (*Y*) oder nicht (*N*).
Shutdown_priv	shutdown	Der Benutzer darf den MySQL-Server stoppen (*Y*) oder nicht (*N*).
Process_priv	process	Der Benutzer darf sich über laufende MySQL-Serverprozesse informieren (*Y*) oder nicht (*N*).
File_priv	file	Der Benutzer darf Daten aus Tabellen in externe Dateien exportieren bzw. Daten aus externen Dateien in Tabellen importieren (*Y*) oder nicht (*N*).
Grant_priv	grant	Der Benutzer darf seine eigenen Rechte an andere Benutzer weitergeben (*Y*) oder nicht (*N*).
Index_priv	index	Der Benutzer darf Indexe anlegen und löschen (*Y*) oder nicht (*N*).
Alter_priv	alter	Der Benutzer darf Tabellenstrukturen ändern (*Y*) oder nicht (*N*).

Tabelle 13.5: Ein-/ausschaltbare Berechtigungen für Benutzer (Fortsetzung)

Alle Berechtigungen aus dieser Liste, die auf einzelne Datenbanken anwendbar sind, finden sich in der Tabelle *db* wieder. Hier können diese Rechte abweichend von den globalen Einstellungen für bestimmte Datenbanken und bestimmte Benutzer gespeichert werden. Ähnlich ist es in der Tabelle *host*, wo nicht datenbankspezifische, sondern hostspezifische Abweichungen zu den Default-Rechten gespeichert werden können. In der Tabelle *tables_priv* können benutzerspezifisch abweichende Rechte für einzelne Tabellen und in *columns_priv* für einzelne Tabellenspalten gespeichert werden.

Die Rechteeinstellung ist also sehr fein justierbar, wenn man sich die Mühe macht, sie zu nutzen. Nun hat eine MySQL-Installation auf einem Server-Rechner für Website-Datenanbindung andere Aufgaben als etwa eine zentrale Firmeninstallation, in der direkt SQL-fähige Clients auf die Datenbanken zugreifen und wo festgelegt werden muss, welche Mitarbeiter welche Daten sehen, ändern oder löschen dürfen.

Doch für jeden Benutzer, der auf Systemebene eingerichtet wird, sollte es auch innerhalb von MySQL ein eigenes Benutzerkonto geben. Das betrifft sowohl menschliche Benutzer als auch wichtige Benutzer aus Bits und Bytes. Wenn etwa ein als Apache-Modul laufendes PHP-Script mit MySQL arbeiten will, ist es sinnvoll, den Apache-Server als Benutzer einzurichten. Über dessen MySQL-Zugangsdaten kann das PHP-Script dann eine Datenbankverbindung herstellen.

Das Einrichten eines Benutzers für Apache-Prozesse erlaubt auch, die Rechte auf MySQL-Ebene so einzustellen, dass kein Missbrauch möglich ist. Ein Missbrauch kann nämlich leicht dadurch entstehen, dass ein PHP-Script einen Inhalt aus einem HTML-Formularfeld in ein SQL-Statement dynamisch einbaut. Hacker können diese Möglichkeit nutzen, um den Formularfeldinhalt so zu gestalten, dass das SQL-Statement, welches das Script daraus bastelt, etwas ganz anderes liefert als das, was sich der Programmierer vorgestellt hat.

Benutzer anlegen und Rechte gewähren

Als Benutzer *root* haben Sie durchaus alle Rechte, um die Tabellen der Datenbank *mysql* mithilfe entsprechender SQL-Statements mit Daten zu befüllen und zu verwalten. Auf diese Weise können Sie Benutzer anlegen, ändern und löschen. Dennoch ist das nicht der von MySQL empfohlene Weg. Stattdessen gibt es spezielle, allein diesem Zweck dienende SQL-Statements. Diese Statements bewirken, dass MySQL die damit formulierten Benutzerrechte in die Tabellen der Datenbank *mysql* einträgt.

Angenommen, Sie möchten einen Benutzer namens *Apache* einrichten. Dieser soll aus der Datenbank *website* Tabellen lesen und ändern dürfen, nicht aber ganze Tabellen löschen dürfen. Außerdem soll er keine anderen Datenbanken anlegen und löschen und keine Indexe anlegen und löschen dürfen. Administrative Statements soll er ebenfalls nicht absetzen können. Eine solche Restriktion ist für viele Webanwendungen ausreichend und verhindert Missbrauchmöglichkeiten.

Das geeignete SQL-Statement lautet beispielsweise:

```
grant select, insert, update, delete
      on website.*
      to Apache@localhost
      identified by "L23Ov7F";
```

Damit wird ein Benutzer `Apache` eingerichtet, falls er noch nicht vorhanden ist. Falls bereits ein Benutzer mit diesem Namen existiert, werden seine Rechte entsprechend geändert. Es wird festgelegt, dass der Benutzer nur vom aktuellen Hostrechner (`localhost`) aus auf den MySQL-Server zugreifen kann. Als Passwort für diesen Benutzer wird `L23Ov7F` festgelegt. Es wird festgelegt, dass der Benutzer die Rechte zum Einfügen (`insert`), Ändern (`update`) und Löschen (`delete`) von Datensätzen hat, und zwar in allen Tabellen der Datenbank *website* (`website.*`).

Das typische Schema des grant-Statements lautet also:

```
grant <Rechteliste> on <Bereich> to <Benutzer@Hostname> identified by
"Passwort";
```

Ein Benutzername darf derzeit nicht mehr als 16 Zeichen haben.

Anstelle einer kommagetrennten Aufzählung einzelner Rechte können Sie auch das Schlüsselwort `all` angeben. Damit gewähren Sie dem Benutzer alle Rechte. Welche Rechte Sie im Einzelnen vergeben können, entnehmen Sie Tabelle 13.5: Ein-/ausschaltbare Berechtigungen für Benutzer (mittlere Spalte).

Hinter dem Schlüsselwort `on` bestimmen Sie, für welchen Bereich die Rechte gelten sollen. Wenn Sie `*.*` angeben, bedeutet das: für alle Tabellen in allen Datenbanken. Wenn Sie beispielsweise `website.pages` angeben, bedeutet das: nur für die Datenbank *website* und dort für die Tabelle *pages*, für andere Datenbanken und Tabellen nicht. Im obigen Beispiel wird mit `website.*` angewiesen: Die genannten Rechte gelten für alle Tabellen der Datenbank *website*, nicht jedoch für andere Datenbanken oder deren Tabellen.

Hinter dem Schlüsselwort `to` werden der Benutzer und der Host angegeben, für den die Rechte gelten. Benutzername und Hostname werden durch ein @-Zeichen getrennt. Wenn Sie `localhost` angeben, kann sich der Benutzer nur vom aktuellen Hostrechner aus anmelden. Für einen Benutzer wie im Beispiel den Apache Webserver ist das die richtige Wahl, sofern sich der Apache ja auf dem gleichen Hostrechner befindet. Wenn ein Benutzer von überall aus zugreifen dürfen soll, z.B. auch bei Verwendung dynamischer IP-Adressen, dann können Sie `Benutzername@"%"` notieren. Das Wildcardzeichen erlaubt einen beliebigen Hostnamen. Ebenso können Sie den Hostnamen mithilfe von Wildcardzeichen zu Teilen spezifizieren, um den Zugriff aus bestimmten Namensräumen oder IP-Adressbereichen zu erlauben, z.B. `benutzername@"%.domaene.de"`.

Am Ende eines grant-Statements können Sie optional den Zusatz `with grant option` notieren. Damit erlauben Sie dem Benutzer, seine eigenen Rechte an andere Benutzer durch Aufruf des grant-Statements weitergeben zu können.

Benutzerrechte entziehen

Zum gezielten Entziehen einzelner Rechte steht das revoke-Statement zur Verfügung. Ein Beispiel:

```
revoke delete on website.* from Apache@localhost;
```

Damit wird dem Benutzer *Apache* von Host *localhost* das Recht entzogen, aus irgendwelchen Tabellen der Datenbank *website* Datensätze zu löschen.

Passwörter ändern

So wie für Benutzer auf Betriebssystemebene gilt auch für MySQL-Benutzer: Regelmäßiges Ändern von Passwörtern erhöht die Sicherheit. Das Passwort des Benutzers, als der Sie selber angemeldet sind, können Sie durch folgendes SQL-Statement (Beispiel) ändern:

```
set password = password("M989nan01");
```

Das Passwort für einen anderen Benutzer können Sie nur ändern, wenn Sie mit dem Benutzer, als der Sie selber angemeldet sind, Zugriff auf die Tabelle *mysql* haben. In diesem Fall lautet das Statement (Beispiel):

```
set password for stefan@localhost = password("lopi.taz77!");
```

Auch in diesem Fall können Sie beim Hostnamen mit Wildcards arbeiten.

Der Statement-Teil hinter dem zuweisenden = ist übrigens ein Funktionsaufruf. Die Funktion `password()` sorgt automatisch für eine Hash-Verschlüsselung des eingegebenen Passworts. Nur in dieser verschlüsselten Form wird das Passwort in *mysql.user* abgespeichert.

13.3 Praxisbeispiel: Entwurf Webshop-Datenbank

Das Ziel einer relationalen Datenbank ist es, Daten eines bestimmten Bereichs der Realität in tabellarischer Form so zu speichern, dass die Daten nach verschiedenen Kriterien filterbar und ermittelbar sind. Da die Daten einem begrenzten Bereich angehören, hängen sie auch irgendwie zusammen. Wie sie genau zusammenhängen und wie sich ein Datenbereich durch Überlegungen in eine relationale Tabellenstruktur umsetzen lässt, behandelt dieser Abschnitt. Er demonstriert anhand eines Beispiels, wie eine Datenbank in der Praxis entsteht.

13.3.1 Schritt 1: Datenaufnahme

Der erste Schritt besteht darin, sich genau zu überlegen, welche Daten benötigt werden. Je geübter Sie im Entwurf von Datenbanken sind, desto eher werden Sie schon damit beginnen, in Tabellen, Datentypen, Feldnamen usw. zu denken. Das ist jedoch nicht erforderlich und sogar eher hinderlich. Die Datenaufnahme sollte noch nichts mit der Umsetzung in eine MySQL-Tabellenstruktur zu tun haben. Gut geeignet zur Datenaufnahme ist etwa ein Brainstorming, bei dem einfach nur »Datenfelder« notiert werden, egal, zu welchem Bereich sie gehören. Das Ergebnis eines solchen Brainstorming kann dann anschließend geordnet und ergänzt werden.

Angenommen, für eine Website soll ein Shop-System eingerichtet werden. Dazu gehören mindestens folgende Daten:

- Daten zu Produkten, die über die Website angeboten und verkauft werden sollen
- Daten zu Kunden, die angebotene Produkte kaufen
- Daten zu Verkaufsvorgängen

Mithilfe der Produktdaten sollen die Produkte auf der Website vorgestellt und beschrieben werden können. Außerdem soll gespeichert werden, wie viele Exemplare eines Produkts noch auf Lager sind, damit entscheidbar ist, ob ein Produkt derzeit überhaupt angeboten werden kann oder nicht.

Jeder Kunde, der etwas kauft, muss gewisse Daten hinterlassen, damit ihm ein Produkt zugestellt werden kann. Diese Daten sollen gespeichert werden, damit der Status jedes Verkaufsvorgangs über einen längeren Zeitraum verfolgbar ist (also z. B. für Mahnwesen, Buchhaltung oder Steuern), aber auch, damit langfristig Kundenprofile erstellt werden können.

Zum Verkaufsvorgang gehört, wann welcher Kunde welches Produkt zu welchen Konditionen bestellt hat, welchen Status die Auslieferung hat und welchen Status die Bezahlung hat. Diese Daten sollen für den jeweiligen Kunden auch einsehbar sein.

Im Hintergrund gehören zu einem Shop natürlich noch andere Daten, wie Finanz- und Lohnbuchhaltung, Wareneinkaufs- und Lagerverwaltungsdaten usw. Es ist jedoch keineswegs zwingend erforderlich und sogar eher bedenklich, diese in der gleichen Datenbank zu halten wie diejenigen Daten, die für den Web-Shop benötigt werden. In der Praxis ist es sogar meist so, dass die Daten ohnehin alle in einer Datenbank im Hintergrund vorgehalten werden, die auf einem anderen Rechner als dem Webserver-Rechner läuft. Die Datenbank des Web-Shops muss dann lediglich regelmäßig mit Daten aus der Hintergrunddatenbank versorgt werden und umgekehrt müssen Daten, die über die Webdatenbank neu einzufügen sind, regelmäßig in die Datenbank im Hintergrund geschrieben werden. Mithilfe entsprechender Scripts auf Betriebssystemebene, SQL-Statements und cron-Jobs ist so etwas automatisierbar.

Bei der Datenaufnahme ist natürlich eine eventuelle Hintergrunddatenbank wichtig, da die anzulegende Datenbank mit den Daten der Hintergrunddatenbank korrespondieren muss. In unserem Fall sparen wir uns diese Auseinandersetzung und versuchen, unsere Datenaufnahme ohne Rücksicht auf andere Datenbanken zu konkretisieren.

Folgende Daten sollen zu einem Produkt gehören:

- Produktnummer
- Produktbezeichnung
- Produktkurzbeschreibung
- Link zu einer Adresse für eine Produktlangbeschreibung (Datenblatt)
- Aktueller Nettoverkaufspreis
- Aktuelle Lagerstückzahl

Zu jedem Kunden, der ein Produkt von der Website kauft, sollen folgende Daten gespeichert werden:

- Kundennummer
- Anrede, Vorname, Zuname und gegebenenfalls Firma/Organisation
- Anschrift
- Telefonnummer für eventuelle Rückfragen
- E-Mail-Adresse für Auftragsbestätigung
- Benutzernamen und Passwort für spätere Registrierungsmöglichkeit

Zu den einzelnen Verkaufsvorgängen sollen es folgende Daten sein:

- Bestellvorgangsnummer
- Kunde, der etwas bestellt/gekauft hat
- Produkt, das gekauft/bestellt wurde
- Zeitpunkt der Bestellung

- Nettoverkaufspreis
- Mehrwertsteuer
- Lieferstatus
- Bezahlstatus

13.3.2 Schritt 2: Ablaufbeschreibungen

Um eine Datenbank so zu strukturieren, dass sie den Anforderungen gerecht wird, ist es nicht nur wichtig zu ermitteln, welche Daten gespeichert werden sollen, sondern auch, in welcher Weise die Daten typischerweise erzeugt und benötigt werden. Deshalb sollten Sie im Vorfeld versuchen, typische Abläufe zu beschreiben. Und zwar – wie schon bei der Datenaufnahme – nicht aus Sicht des PHP-Programmierers, sondern aus Sicht der Realität.

In unserem Shop sind etwa die im Folgenden dargestellten Abläufe denkbar.

Kunde sucht nach Produkt

Der Kunde soll in den Produkten stöbern oder danach suchen können. Beim Suchen soll er verschiedene Kriterien eingeben können. Anschließend erhält er eine Auswahl der Produkte, die den Kriterien entsprechen. Außerdem soll er seine Auswahl wieder rückgängig machen und auf Wunsch neu formulieren können.

Die Kriterien sollen dem Kunden entgegenkommen. Dieser sucht vielleicht ein bestimmtes Produkt und weiß, wie es heißt, oder er sucht ein bestimmtes Produkt und weiß nicht, wie es heißt, oder er sucht ein bestimmtes Produkt, will aber nicht mehr als einen bestimmten Betrag dafür ausgeben, oder er sucht im Rahmen eines Budgets nach einem Produkt in einem bestimmten Produktbereich.

Es muss möglich sein, ihm eine Produktauswahl abhängig von all diesen Kriterien anzubieten.

Kunde benutzt Einkaufskorb

In jedem Shop ist es üblich, dass ein Kunde mehrere Produkte auf einmal kaufen kann. Er sammelt die gewünschten Produkte in der Hand, in einem Einkaufskorb oder einem Einkaufswagen. Das sollte auch im geplanten Web-Shop so sein, denn andernfalls müsste der Kunde für jedes Produkt einen neuen Bestellvorgang abwickeln. So etwas ist nicht gerade kundenfreundlich und soll daher vermieden werden.

Kunde bestellt Produkt

Zunächst muss der Kunde die für die Lieferung nötigen Daten eingeben. Es ist sinnvoll, ihn dabei auch gleich zu bitten, Benutzername und Kennwort mit anzugeben. Falls er mit solchen Daten bereits gespeichert ist, kann er sich die Eingabe der restlichen Daten sparen, weil diese aus der Datenbank geholt werden. Er bekommt sie jedoch angezeigt und kann sie ändern, falls sie z. B. nicht mehr stimmen.

Im nächsten Schritt wird dem Kunden in einer Übersicht angezeigt, welche Produkte er zur Bestellung ausgewählt hat und welche Bruttosumme sich durch die Nettopreise der Einzelartikel plus eventuell Mehrwertsteuer ergibt. Die zuvor angegebenen Daten zur Lieferanschrift werden ebenfalls zur Kontrolle mit angezeigt.

Bei der Mehrwertsteuer ist anzumerken, dass diese nur für inländische Bestellungen berechnet werden soll. Das bedeutet, dass Bestellungen aus dem Ausland erkannt und gespeichert werden sollen. Es bietet sich an, bei der Kundenanschrift ein Feld für das Land hinzuzufügen, das diese Aufgabe übernimmt.

Am Ende der Bestellung soll der Kunde auswählen, wie er bezahlen möchte: mit Einzugsermächtigung, mit Kreditkarte oder per Rechnung. Durch eine endgültige Bestätigung macht er seine Bestellung dann wirksam. Neben der Sofortbestätigung im Browser wird auch eine E-Mail an seine angegebene Mail-Adresse versendet, welche alle Bestelldaten noch einmal auflistet.

Konsequenzen für die Überlegungen zum Datenbankentwurf

Die drei beschriebenen Vorgänge beeinflussen unsere bisherige Datenaufnahme. Bei den Produktdaten benötigen wir auf jeden Fall noch die Produktbereiche, zu denen ein Produkt gehört. Denn anders ist nicht jede Suche wie beschrieben möglich. Dabei soll ein Produkt durchaus auch mehreren Produktbereichen zugeordnet werden können.

Die Einkaufskorbfunktion bezieht sich zwar nur auf die aktuelle Sitzung und muss daher nicht dauerhaft in der Datenbank gespeichert werden. Jedoch wird durch die Einkaufskorbfunktion klar, dass keine 1:1-Beziehung zwischen Kaufvorgängen und Produkten besteht, da bei jedem Kaufvorgang ja mehrere Produkte gekauft werden können. »Produkt, das gekauft/bestellt wurde« muss daher geändert werden in »Produkte, die gekauft/bestellt wurden«.

Bei der Auswertung des beschriebenen Bestellvorgangs stellen wir fest, dass in der Datenaufnahme bislang keine Daten zur Art der Bezahlung enthalten waren. Dieses Datum sollte jedoch mit aufgenommen werden.

13.3.3 Relationen entwerfen

Auch in diesem Schritt müssen wir noch nicht in SQL-Statements denken. Allerdings müssen wir jetzt die Art der Datenspeicherung in einer relationalen Datenbank berücksichtigen. Es gilt zu ermitteln, welche Tabellen benötigt werden und welche Verknüpfungsfelder zu anderen Tabellen darin erforderlich sind.

Produkte

Es ist klar, dass wir im Beispiel eine Tabelle für Produktdaten benötigen, mit den ermittelten Daten. Die Daten, die zunächst aufgenommen wurden, gehören zu jedem einzelnen Produkt und können daher prima in einer Tabelle gespeichert werden. Bei den Produktbereichen müssten wir allerdings die Bereiche, denen ein Produkt zugeordnet ist, in einer Liste speichern, z. B. kommagetrennt.

Immer, wenn Sie bei den Entwurfsüberlegungen auf ein solches Erfordernis stoßen (»da muss ich ja mehrere Daten in einer Liste speichern«), ist es aus Sicht relationaler Datenbanken meist sinnvoller, eine zusätzliche Tabelle zu spendieren. In unserem Fall würden wir also eine zusätzliche Tabelle spendieren, in der wir Produkte und Produktbereiche zuordnen.

Die Tabellenstruktur sieht also folgendermaßen aus:

Produkte	**Produktnummer**
	Produktbezeichnung
	Produktkurzbeschreibung
	Link zu Datenblatt
	Aktueller Nettopreis
	Aktuelle Lagerstückzahl

Tabelle 13.6: Entwurf der Beispieltabelle für Produkte

Produktbereichszuordnungen	**Produktnummer**
	Produktbereich

Tabelle 13.7: Entwurf der Beispieltabelle für Produktbereichszuordnungen

Fett dargestellte Felder dienen als Verknüpfungsfelder und sind tabellenübergreifend am gleichen Namen erkennbar.

Kunden

Bei den ermittelten Kundendaten ist es zunächst sinnvoll, ordentlich in »atomare Felder« zu untergliedern, vor allem bei der Anschrift. Wie weit die Atomarisierung gehen soll, ob etwa Straße und Hausnummer zwei Felder werden sollen oder ob die Telefonnummer in Vorwahl und Durchwahl getrennt werden soll, hängt von den Erfordernissen ab. Da wir in unserem Fall jedoch nicht beabsichtigen, Abfragen der Sorte »zeige mir alle Kunden mit Vorwahlnummer 089« durchzuführen, können wir uns diese Atomisierung sparen.

Die ermittelten Kundendaten lassen sich zwar gut in einer Tabelle zusammenfassen. Doch es gibt Unterschiede bei den Daten. Benutzername und Passwort sind zwar kundenspezifische Daten, doch sie sollen freiwillig sein. Das bedeutet, dass es nicht zu jedem Kunden einen Benutzernamen und ein Kennwort geben wird. Außerdem haben die site-internen Anmeldedaten funktional nichts mit der Lieferanschrift zu tun. Es erscheint daher durchaus sinnvoll, zwei Tabellen zu spendieren, eine mit Kundenstammdaten und eine mit Kundenzugangsdaten.

Auch die Angabe einer Firma/Organisation ist kein Pflichtfeld und wird voraussichtlich sogar nur in einer Minderheit von Datensätzen vorkommen. Würden wir eine strenge Normalisierung von Daten anstreben, müssten wir dieses Feld ebenfalls in eine Extra-Tabelle auslagern, die nur Kundennummern und Firmennamen speichert. Doch so weit wollen wir es nicht treiben. Wir müssen nicht an jedem Byte sparen und können es uns

leisten, dass die Mehrheit der Felder in dieser Spalte leer bleiben wird. In der Praxis werden Sie häufig solche Kompromisse bei der Datennormalisierung finden.

Wir entscheiden uns für folgende Tabellenstruktur:

Kundenstammdaten	**Kundennummer**
	Anrede
	Vorname
	Zuname
	Firma/Organisation
	Anschrift
	Land
	PLZ
	Ort
	Telefonnummer
	E-Mail

Tabelle 13.8: Entwurf der Beispieltabelle für Kundenstammdaten

Kundenanmeldedaten	**Kundennummer**
	Benutzername
	Passwort

Tabelle 13.9: Entwurf der Beispieltabelle für Kundenanmeldedaten

Bestellungen

Das logische Bindeglied zwischen Produkten und Kunden sind die Bestellvorgänge. Hierbei haben wir festgestellt, dass wegen der gewünschten Warenkorbfunktion eine 1:n-Beziehung zwischen Bestellung und bestellten Produkten berücksichtigt werden muss. Es bietet sich an, eine Tabelle für Bestellungskerndaten bereitzustellen und eine Zuordnungstabelle zwischen Bestellvorgängen und dabei bestellten Produkten. Die Bestellvorgangsnummer können wir in beiden Tabellen verwenden, um die Zuordnung herzustellen.

Im zunächst angedachten Feld für den Nettoverkaufspreis müssten wir die Summe der bestellten Produkte speichern. Zur Normalisierung von Daten gehört es jedoch auch, keine errechneten Werte zu speichern, da diese sich ja jederzeit wieder errechnen lassen. In unserem Fall ist es allerdings so, dass Produktpreise sich ändern können. Bei einer späteren Summenbildung über bestellte Produkte auf Basis des Feldes für den Nettopreis in der Tabelle für Produkte könnte dann ein anderes Ergebnis herauskommen als bei der seinerzeit getätigten Bestellung.

Da die Nettopreissumme einer Bestellung im Nachhinein nachvollziehbar bleiben soll, bleibt ohnehin nur, in die Zuordnungstabelle für Bestellvorgänge und Produkte deren bei der Bestellung aktuellen Preis mit aufzunehmen. Es ergibt sich also folgende Tabellenstruktur:

Bestellvorgänge	Bestellvorgangsnummer
	Kundennummer
	Datum der Bestellung
	Uhrzeit der Bestellung
	Mehrwertsteuer (Prozent)
	Lieferstatus
	Bezahlstatus

Tabelle 13.10: Entwurf der Beispieltabelle für Bestellvorgänge

Produktbestellungen	Bestellvorgangsnummer
	Produktnummer
	Produktnettopreis

Tabelle 13.11: Entwurf der Beispieltabelle für Produktbestellungen

Fazit

Es haben sich insgesamt sechs Tabellen ergeben, die nicht nur in der Lage sind, alle erforderlichen Daten zu speichern, sondern dabei auch die Relationen zwischen Daten berücksichtigen und gleichzeitig so weit normalisiert sind, dass die Spalten jeder Tabelle einen funktionalen Zusammenhang ergeben.

13.3.4 Felder und Schlüssel definieren

Nachdem die Tabellen, ihre Felder und deren Relationen feststehen, müssen als Nächstes die Feldnamen, Feldeigenschaften wie Datentyp, Feldlänge usw. festgelegt werden. Außerdem muss festgelegt werden, welche Felder welche Indextypen (Primärindex, Unique-Index oder »normaler Index«) erhalten sollen.

Es ist sinnvoll, auch diese Definitionen code-unabhängig zu dokumentieren. Dennoch ist dies auch der Punkt, ab dem es sinnvoll ist, direkt mit SQL weiterzuarbeiten. Eine Möglichkeit, beides unter einen Hut zu bringen, besteht darin, eine Datei mit den SQL-Statements zum Erzeugen der Datenbank und ihrer Tabellen zu erstellen und die Tabellendefinitionen darin auszukommentieren. Kommentarzeilen in solchen Dateien können Sie einfach durch ein voranstehendes Gatterzeichen (#) erzeugen.

Unsere Beispieltabellen setzen wir in einer Datei namens ***shop_create.sql*** wie folgt um:

Listing 13.1: SQL-Batchdatei shop_create.sql

```
# ------------------------------------------------
# SQL-Batch-Datei zum Erzeugen der Web-Shop-Datenbank
# ------------------------------------------------

create database if not exists webshop;

create table if not exists webshop.products (
```

```
    number int unsigned not null,
    name varchar(80) not null,
    description blob,
    datasheet_link varchar(80),
    netto_price decimal(4,2),
    on_hand smallint not null,
    primary key(number),
    unique(name),
    key(netto_price)
);

create table if not exists webshop.product_assignments (
    product_number int unsigned not null,
    assignment varchar(80) not null,
    key(product_number),
    key(assignment)
);

create table if not exists webshop.customers (
    number int unsigned not null,
    given_name varchar(30) not null,
    family_name varchar(30) not null,
    organization varchar(40) not null,
    address varchar(40) not null,
    country char(2) not null,
    postal_code varchar(10) not null,
    location varchar(40) not null,
    phone varchar(40) not null,
    mail varchar(60) not null,
    primary key(number),
    key(given_name)
);

create table if not exists webshop.registered_customers (
    customer_number int unsigned not null,
    user_name varchar(20) not null,
    user_password varchar(20) not null,
    primary key(customer_number),
    key(user_name)
);

create table if not exists webshop.orders (
    number int unsigned not null,
    customer_number int unsigned not null,
    order_date date not null,
    order_time time not null,
    vat_percent tinyint not null,
    delivery_state enum("ordered", "underway", "sent") not null,
    payment_state enum("not_paid", "remindered", "paid") not null,
    primary key(number),
    key(customer_number)
);
```

```
create table if not exists webshop.ordered_products (
   order_number int unsigned not null,
   product_number int unsigned not null,
   netto_price decimal(4,2) not null,
   key(order_number),
   key(product_number)
);
```

Bei den Tabellen- und Feldnamen haben wir uns an eine durchgängig englische Namenskonvention gehalten. Auch andere Regelmäßigkeiten, zu denen SQL niemanden zwingt, gehören dazu, wie etwa der einheitliche Feldname `number` für Primärindex-Felder. Bezugsfelder lassen den Tabellenbezug eindeutig erkennen, wie `product_number` oder `order_number`.

Fast alle Felder sind mit der Option `not null` markiert, was verdeutlicht, dass keine wirklich überflüssigen Felder vorkommen.

Die Feldtypen wurden mit Bedacht gewählt, z.B. `tinyint` für den Mehrwertsteuer-Prozentsatz, da dieser auf jeden Fall keine höhere Zahl ist. Preise wurden mit `decimal(4,2)` definiert, da in der Konzeption des Shops davon ausgegangen wird, dass keine fünfstelligen Preise vorkommen und dass zu jedem Preis stets ein Cent-Betrag gehört. Zeichenkettenfelder wurden vorwiegend mit `varchar()` definiert und nach realistischen Erfahrungswerten ausreichend dimensioniert. Nur das Feld `country` wurde fix mit `char(2)` definiert. Es soll zur besseren Eindeutigkeit ein zweistelliges Länderkürzel nach ISO 3166 aufnehmen, also etwa DE, AT, CH usw.

Neben den Primärindexen auf die number-Felder bei Produkten, Kunden und Bestellungen wurden Indexe auf Felder gesetzt, die bei Abfragen in where-Klauseln auftauchen können. Das sind Produktbezeichnungen, Preise, Kundennamen, Kundenbenutzernamen, bei Bestellungen die Kundennummern und bei bestellten Produkten die Produkt- und Bestellvorgangsnummern.

Fazit

Die Datenbank ist damit fertig erstellt und kann mit Daten befüllt werden. Dies kann je nach Umgebung durch ein Script geschehen, das Daten aus anderen Datenbanken in diese Datenbank überträgt, oder auch durch eine webbasierte Oberfläche im Browser, z.B. als geschützte Seiten innerhalb der Site, die es Ihnen als Shop-Betreiber ermöglichen, Produktdaten einzugeben.

Die Tabellen für Kunden und Bestellungen füllen sich, indem Kunden etwas bestellen. Die zugehörige Webanwendung muss in einer Script-Sprache wie PHP geschrieben sein, welche die Kommunikation mit MySQL erlaubt und die Daten aus einem Bestellformular nach ausreichender Prüfung in die Datenbank schreibt.

13.4 MySQL-Datenbankzugriffe mit PHP

Damit sind wir wieder bei PHP angekommen. Mit ausreichenden Hintergrundkenntnissen zur Funktionsweise von MySQL ausgestattet, sind Sie nun in der Lage, die PHP-Funktionen zur Kommunikation mit MySQL sinnvoll anzuwenden.

PHP und MySQL-Unterstützung

Damit PHP mit MySQL kommunizieren kann, muss es entsprechend eingerichtet sein. In Version 4.x hat PHP über eigene MySQL-Client-Bibliotheken verfügt. In PHP 5.x hat man sich davon verabschiedet, da die PHP-eigenen MySQL-Client-Bibliotheken mit denen von MySQL selbst wegen Versionsunterschieden in Konflikt geraten konnten. Während die MySQL-Unterstützung in PHP 4.x also von vorneherein funktioniert, muss sie in PHP 5.x explizit eingerichtet werden. Unter Linux/Unix muss dazu PHP über das Script *.configure*, das nach Entpacken der Installationsdateien von PHP verfügbar ist, mit MySQL-Unterstützung kompiliert werden. Geben Sie dazu auf Shell-Ebene im Verzeichnis des .configure-Scripts an:

```
./configure --prefix=/usr/local/bin/php --with-config-file-path=/etc --with-
mysql=/pfad/zu/mysql
```

Der Beispielaufruf bereitet die Kompilierung von PHP mit den beiden typischsten Standardoptionen und mit MySQL-Unterstützung vor, wobei */pfad/zu/mysql* durch den Pfad zu ersetzen ist, den das Shell-Kommando `which mysql` liefert. Das weitere Vorgehen einer Kompilierung wird in ***Abschnitt 12.1.2*** beschrieben.

Unter MS Windows müssen Sie in der *php.ini* von PHP 5.x das führende Semikolon bei der Zeile `extension=php_mysql.dll` entfernen. Außerdem müssen Sie die Datei *php_mysql.dll* aus dem Verzeichnisbereich von PHP ins Windows-Systemverzeichnis (z.B. *winnt\system32* oder *windows\system32*) kopieren. Bei PHP 4.x brauchen Sie nichts weiter zu tun.

MySQL-Laufzeitoptionen in php.ini

In der *php.ini* gibt es einige Direktiven speziell für MySQL, die unterhalb der Sektionsüberschrift *[MySQL]* zu finden sind:

Option	Default-Wert	Bedeutung	Berechtigung
mysql.allow_persistent	*On*	Wenn *On*, sind persistente Verbindungen zu MySQL möglich, wenn *Off*, nicht. Persistente Verbindungen ermöglichen keine Session-Funktionalität oder Ähnliches. Sie können lediglich bei sehr vielen gleichzeitigen Datenbankverbindungen helfen, besser mit den Ressourcen umzugehen.	*ALL*

Tabelle 13.12: MySQL-spezifische Optionen in der php.ini

Option	Default-Wert	Bedeutung	Berechtigung
mysql.connect_timeout	*60*	Anzahl Sekunden, die PHP auf Antworten vom MySQL-Server wartet.	*ALL*
mysql.default_host		Default-Hostname, falls im Script keiner explizit angegeben wird. Aus Sicherheitsgründen weniger zu empfehlen (leer lassen).	*ALL*
mysqli.default_host		Ab PHP 5.x: für MySQL 4.1 und höher.	*ALL*
mysql.default_password		Default-Benutzerpasswort, falls im Script keines explizit angegeben wird. Aus Sicherheitsgründen weniger zu empfehlen (leer lassen).	*ALL*
mysqli.default_pw		Ab PHP 5.x: für MySQL 4.1 und höher.	*ALL*
mysql.default_port		Abweichender Default-Port. Im Normalfall leer zu lassen, da PHP den MySQL-Default-Port über eine Umgebungsvariable ermitteln kann.	*ALL*
mysqli.default_port		Ab PHP 5.x: für MySQL 4.1 und höher.	*ALL*
mysql.default_socket		Default-Socketname, falls MySQL nicht wie meistens üblich über TCP/IP und Port, sondern über eine Socketverbindung kontaktiert wird.	*ALL*
mysqli.default_socket		Ab PHP 5.x: für MySQL 4.1 und höher.	*ALL*
mysql.default_user		Default-Benutzername, falls im Script keiner explizit angegeben wird. Aus Sicherheitsgründen weniger zu empfehlen (leer lassen).	*ALL*
mysqli.default_user		Ab PHP 5.x: für MySQL 4.1 und höher.	*ALL*
mysql.max_links	*-1*	Maximale Anzahl von MySQL-Verbindungen pro PHP-Prozess. *-1* bedeutet: keine Einschränkung.	*SYSTEM*
mysqli.max_links	*-1*	Ab PHP 5.x: für MySQL 4.1 und höher.	*SYSTEM*
mysql.max_persistent	*-1*	Ab PHP 5.x.	*SYSTEM*
mysql.trace_mode	*Off*	Wenn *On*, werden SQL-Fehler und -Warnungen im Browser ausgegeben, wenn *Off*, nicht.	*ALL*
mysqli.reconnect	*Off*	Ab PHP 5.x: für MySQL 4.1 und höher. Wenn *On*, wird automatisches Neuverbinden zum MySQL-Server erlaubt, wenn *Off*, dann nicht.	*SYSTEM*

Tabelle 13.12: MySQL-spezifische Optionen in der php.ini (Fortsetzung)

In der Regel sind keine Änderungen an den Voreinstellungen erforderlich. Bei knappen Server-Ressourcen kann es sinnvoll sein, die maximale Anzahl von MySQL-Verbindungen zu beschränken. Allerdings sollte ein PHP-Script in diesem Fall auch eine entsprechende Fehlerbehandlung enthalten. Auch das Heruntersetzen des Werts für den Verbindungs-Timeout ist verantwortbar, denn 60 Sekunden wartet ohnehin kein gewöhnlicher Anwen-

der auf die Antwort eines Servers. Angaben zum MySQL-Benutzernamen, Passwort und Host sollten, wenn keine besonderen Gründe dagegen sprechen, nicht in der *php.ini* stehen, sondern stets im PHP-Script. Wie, das werden wir im nächsten Abschnitt sehen.

13.4.1 MySQL-Datenbankverbindungen mit PHP

Um zu testen, ob bei Ihnen die MySQL-Unterstützung in PHP funktioniert, können Sie nachfolgendes Testscript benutzen. Beachten Sie, dass Sie die Wertzuweisungen in den ersten beiden Zeilen, also die Werte für `$user` und `$password`, anpassen müssen. Unter MySQL sollten Sie, bevor Sie mit PHP darauf zugreifen, einen geeigneten Benutzer eingerichtet und mit den gewünschten Rechten ausgestattet haben (siehe auch ***Abschnitt 13.2.8***).

Der Quelltext des Scripts lautet:

```
<?php
   $user = "Apache";
   $password = "w34t90u";
   $host = "localhost";
   $sql = "select curdate() as today";
   $dbh = mysql_connect($host, $user, $password);
   $res = mysql_query($sql, $dbh);
   if($res)
      $row = mysql_fetch_assoc($res);
   mysql_close($dbh);
   echo $row['today'];
?>
```

Das Script gibt, wenn alles funktioniert, das aktuelle Datum in der Form JJJJ-MM-TT aus. In diesem Fall steht der PHP-Programmierung für MySQL auf Ihrem Rechner nichts mehr im Wege!

Um mit PHP eine Verbindung zu MySQL aufzubauen, benötigen Sie normalerweise einen Benutzernamen, ein Passwort und eine Angabe zum Hostnamen. Der Hostname ist, wenn PHP und MySQL sich auf dem gleichen Rechner befinden, in aller Regel `localhost`. Das PHP-Script sollte sich aus Sicherheitsgründen nicht mit dem MySQL-Benutzer *root* anmelden, sondern es sollte ein anderer, nicht ganz so privilegierter Benutzer dafür eingerichtet werden.

Möglicherweise wundern Sie sich, warum das MySQL-Zugangspasswort im PHP-Quellcode steht. Das Script muss jedoch das Passwort kennen, anders kann es sich nicht bei MySQL anmelden. Zu einer Sicherheitslücke kann das werden, wenn ein PHP-Script vom Webserver auf Grund einer Fehlkonfiguration nicht ausgeführt, sondern als Textdatei an den Browser übertragen wird. In diesem Fall könnte jeder Seitenbesucher den PHP-Quelltext studieren und damit auch die MySQL-Benutzerdaten einsehen. Solange der Benutzer unter MySQL allerdings nur für den Zugang über localhost eingerichtet wurde, müsste sich der erfreute Quelltextbetrachter erst einmal auf Shell-Ebene in den Server-Rechner hacken, bevor er den Zugang nutzen könnte.

In unserem Beispielscript werden die drei für den MySQL-Verbindungsaufbau nötigen Daten in den Variablen `$user`, `$password` und `$host` am Beginn des Scripts definiert, damit die Werte leicht an andere Umgebungen anpassbar sind.

In der Variablen `$sql` speichert das Script ein SQL-Statement als normale Zeichenkette. Das SQL-Statement bewirkt, wenn man es im MySQL-Client so eingeben würde, z. B. folgende Ausgabe:

```
+------------+
| today      |
+------------+
| 2005-03-25 |
+------------+
```

Die Ausgabe besteht also aus einer Tabelle mit einer Spalte (einem Feld) namens ***today*** und einem Datensatz, der in dieser Spalte einen entsprechenden Wert enthält. Dass die Spalte ***today*** heißt, liegt an der Aliasdefinition mit `as today` innerhalb des SQL-Statements.

Unser Testscript tut im weiteren Verlauf nichts anderes, als sich mit MySQL zu verbinden, das in `$sql` definierte SQL-Statement abzusetzen, die Ergebnistabelle wie oben gezeigt in Form eines assoziativen Arrays zu empfangen und dessen einzigen Wert am Ende auszugeben.

Zum Verbindungsaufbau mit dem MySQL-Server dient die Funktion `mysql_connect()`. Dieser werden als Parameter der Hostname, der MySQL-Benutzername sowie das zugehörige Passwort übergeben. Kann die Verbindung erfolgreich hergestellt werden, gibt die Funktion einen Handler zurück, vergleichbar den Handlern, die Sie bereits im Zusammenhang mit Funktionen zum Öffnen, Lesen, Schreiben und Schließen von Dateien kennen gelernt haben. Den Rückgabewert sollte das Script in einer Variablen speichern, da es diese im weiteren Verlauf für die übrigen MySQL-Funktionen benötigt.

Funktion:	`mysql_connect()`
Aufgabe:	Stellt eine Verbindung zum MySQL-Server her.
Parameter:	1. (optional, wenn in ***php.ini*** festgelegt) Hostname, auf dem der MySQL-Server läuft, zu dem die Verbindung hergestellt werden soll. Bei Verbindungen zu entfernten Rechnern kann dies z. B. auch eine IP-Adresse sein. 2. (optional, wenn in ***php.ini*** festgelegt) MySQL-Benutzername, mit dem sich das PHP-Script beim MySQL-Server anmelden soll. 3. (optional, wenn in ***php.ini*** festgelegt) Zugehöriges Passwort des MySQL-Benutzers. 4. (optional) Boolesche Angabe, ob ein Zweitaufruf mit den gleichen Benutzerdaten einen neuen Link erzeugen soll oder nicht. 5. (optional) eine oder mehrere der Zusatzoptionen `MYSQL_CLIENT_COMPRESS`, `MYSQL_CLIENT_IGNORE_SPACE` oder `MYSQL_CLIENT_INTERACTIVE`.
Rückgabewert:	Handler für die Datenbankverbindung oder `false` im Fehlerfall.

Tabelle 13.13: Steckbrief Funktion »mysql_connect()«

Nachdem die MySQL-Verbindung hergestellt ist, kann das PHP-Script ein SQL-Statement absetzen. Dazu dient die Funktion `mysql_query()`. Als erster Parameter wird ihr das gewünschte SQL-Statement übergeben, in unserem Beispiel die zuvor definierte Variable `$sql`. Als zweiter Parameter sollte, um mögliche Probleme zu vermeiden, der von `mysql_connect()` erhaltene Datenbank-Handler übergeben werden.

Funktion:	`mysql_query()`
Aufgabe:	Setzt ein SQL-Statement an den MySQL-Server ab.
Parameter:	1. SQL-Statement als Zeichenkette. 2. (optional) Datenbank-Handler. Wenn nicht angegeben, wird der zuletzt von einem mysql_connect()-Aufruf erhaltene Handler verwendet.
Rückgabewert:	Bei den Statements `select`, `show`, `describe` oder `explain` wird im Erfolgsfall eine Ressource auf die Ergebnisse zurückgegeben, die dann von weiteren speziellen Funktionen zum »Holen« der Ergebnisse verwendet werden kann. Bei Statements wie `delete`, `update` oder `drop` wird im Erfolgsfall `true` zurückgegeben. Im Fehlerfall wird in allen Fällen `false` zurückgegeben.

Tabelle 13.14: Steckbrief Funktion »mysql_query()«

Was `mysql_query()` bei einem Statement wie `select` zurückgibt, sind noch keine Daten aus der Datenbank, sondern lediglich wieder ein Handler für diese Daten. Da der Handler im weiteren Verlauf noch benötigt wird, um auf die einzelnen Ergebnisdaten zuzugreifen, sollte der Aufruf von `mysql_query()` einer Variablen zugewiesen werden. In unserem Testscript ist dies die Variable `$res`.

Das Testscript holt sich die Daten durch Aufruf der Funktion `mysql_fetch_assoc()`. Auf die Verwendungsweise dieser Funktion und ähnlicher PHP-Funktionen gehen wir im nächsten Abschnitt ein. So wie im Beispiel aufgerufen, gibt die Funktion einen assoziativen Array zurück, bei dem die Spaltennamen der MySQL-Ergebnistabelle die Schlüssel sind und die Feldinhalte die zugehörigen Werte. Da unsere Ergebnistabelle eine Spalte namens *today* hat, kann das Script am Ende mit `$row['today']` auf den Wert des ersten Datensatzes dieser Spalte zugreifen.

Wenn eine Datenbankverbindung nicht mehr benötigt wird, sollte sie mit `mysql_close()` geschlossen werden. Die Verbindung wird, sofern sie nicht persistent ist, zwar ohnehin bei Script-Ende geschlossen. Doch vorzeitiges Schließen kann helfen, Ressourcen zu sparen.

Funktion:	`mysql_close()`
Aufgabe:	Schließt eine MySQL-Verbindung.
Parameter:	1. (optional) Datenbank-Handler. Wenn nicht angegeben, wird der zuletzt von einem mysql_connect()-Aufruf erhaltene Handler verwendet.
Rückgabewert:	Im Erfolgsfall `true` , ansonsten `false`.

Tabelle 13.15: Steckbrief Funktion »mysql_close()«

13.4.2 Datenbankabfragen in PHP

Die Funktion `mysql_query()` dient bei einer erfolgreichen Datenbankverbindung zum Absetzen eines SQL-Statements. Selbstverständlich können Sie bei geöffneter Datenbankverbindung auch mehrere solcher Funktionsaufrufe mit verschiedenen SQL-Statements starten. Bei Statements, die eine Ergebnismenge liefern, wie beispielsweise `select`, liefert `mysql_query()` einen Handler, über den der Zugriff auf die Ergebnisdaten möglich ist. Für den Zugriff auf die Ergebnisdaten stellt PHP gleich mehrere Funktionen zur Verfügung:

Eine Funktion, um letztlich an Ergebnisdaten heranzukommen, haben wir bereits kennen gelernt: `mysql_fetch_assoc()`. Diese Funktion liefert bei jedem Aufruf den jeweils nächsten Datensatz aus der Ergebnismenge – so lange, bis keine Datensätze mehr vorliegen.

Funktion:	`mysql_fetch_assoc()`
Aufgabe:	Liefert den jeweils nächsten Datensatz der Ergebnismenge z. B. eines select-Statements als assoziativen Array. Der Funktionsaufruf sollte einer Variablen zugewiesen werden, die anschließend der assoziative Array ist.
Parameter:	1. Ressourcen-Handler, der von `mysql_query()` geliefert wurde.
Rückgabewert:	Gibt den gefundenen Datensatz als assoziativen Array zurück. Feldnamen (Spaltenüberschriften) sind die Schlüssel. Im Fehlerfall wird `false` zurückgegeben.

Tabelle 13.16: Steckbrief Funktion »mysql_assoc()«

Wenn Sie lieber einen 0-index-basierten Array wünschen, also keine Feldnamen als Array-Schlüssel, können Sie stattdessen die Funktion `mysql_fetch_row()` verwenden.

Funktion:	`mysql_fetch_row()`
Aufgabe:	Liefert den jeweils nächsten Datensatz der Ergebnismenge z. B. eines select-Statements als 0-index-basierten Array. Der Funktionsaufruf sollte einer Variablen zugewiesen werden, die anschließend der Array ist.
Parameter:	1. Ressourcen-Handler, der von `mysql_query()` geliefert wurde.
Rückgabewert:	Gibt den gefundenen Datensatz als Array zurück. Das erste Feld des Datensatzes hat dann den Arrayindex 0, das zweite 1 usw. Im Fehlerfall wird `false` zurückgegeben.

Tabelle 13.17: Steckbrief Funktion »mysql_row()«

Wenn Sie beide Varianten wünschen, also assoziativen Array und 0-index-basierten Array, können Sie `mysql_fetch_array()` verwenden.

Funktion:	`mysql_fetch_array()`
Aufgabe:	Liefert den jeweils nächsten Datensatz der Ergebnismenge z. B. eines select-Statements sowohl als 0-index-basierten Array als auch als assoziativen Array. Der Funktionsaufruf sollte einer Variablen zugewiesen werden, die anschließend der Array ist.

Tabelle 13.18: Steckbrief Funktion »mysql_array()«

Parameter:	1. Ressourcen-Handler, der von `mysql_query()` geliefert wurde. 2. (optional): `MYSQL_ASSOC` für »nur assoziativen Array«, `MYSQL_NUM` für »nur 0-index-basierten Array« oder `MYSQL_BOTH` für »beide Array-Arten« (Voreinstellung).
Rückgabewert:	Gibt den gefundenen Datensatz als Array zurück. Im Fehlerfall wird `false` zurückgegeben.

Tabelle 13.18: Steckbrief Funktion »mysql_array()« (Fortsetzung)

Eine vierte Funktion namens `mysql_fetch_object()` wählt den objektorientierten Ansatz, um das gleiche Ergebnis zu liefern. In diesem Fall ist jeder Feldname eine Objekteigenschaft.

Funktion:	`mysql_fetch_object()`
Aufgabe:	Liefert den jeweils nächsten Datensatz der Ergebnismenge z. B. eines select-Statements als Objekt. Der Funktionsaufruf sollte einer Variablen zugewiesen werden, die anschließend die Objektvariable ist. Der Zugriff auf einzelne Felder ist dann über `$objvar->Feldname` möglich.
Parameter:	1. Ressourcen-Handler, der von `mysql_query()` geliefert wurde.
Rückgabewert:	Gibt den gefundenen Datensatz als Objekt zurück. Im Fehlerfall wird `false` zurückgegeben.

Tabelle 13.19: Steckbrief Funktion »mysql_object()«

All diese Funktionen liefern »den jeweils nächsten Datensatz«. Um an alle gefundenen Daten heranzukommen, bedarf es einer Schleife. Der folgende Code-Schnipsel zeigt, wie die Funktionen zum »Holen« eines Datensatzes typischerweise zum Einsatz kommen:

```
$user = "";
$password = "";
$host = "localhost";
$sql = "select given_name, family_name, mail from contacts;
$dbh = mysql_connect($host, $user, $password);
if(!$dbh)
   die("Keine Verbindung zur Datenbank");
$res = mysql_query($sql, $dbh);
if(!$res)
   die("Keine Ergebnisse");
while($row = mysql_fetch_assoc($res))
   // tu was mit dem aktuellen Datensatz
mysql_close($dbh);
```

Bei dieser Konstruktion wird eine while-Schleife benutzt. In der Durchlaufbedingung wird einfach der Funktionsaufruf von `mysql_fetch_assoc()` notiert. Da diese Funktion ebenso wie ihre Schwesternfunktionen `false` liefert, wenn kein Datensatz mehr vorliegt, wird die Schleife so lange durchlaufen, wie Datensätze vorhanden sind. Innerhalb der Schleife kann der jeweils aktuelle Datensatz verarbeitet werden, z. B. ausgegeben werden, in einem verschachtelten Array zwischengespeichert werden usw.

Analyse der Ergebnisdaten

Bei der while-Schleife müssen Sie nicht wissen, wie viele Datensätze insgesamt vorliegen. Manchmal kann es jedoch im Vorfeld sinnvoll sein zu wissen, wie viele Datensätze gefunden wurden. Die Funktion `mysql_num_rows()` liefert dieses Ergebnis. Beispiel eines Einsatzes:

```
$user = "";
$password = "";
$host = "localhost";
$sql = "select given_name, family_name, phone, mail from contacts;
$dbh = mysql_connect($host, $user, $password);
$res = mysql_query($sql, $dbh);
$count = mysql_num_rows($res);
if($count > 100)
   die("Zu viele Ergebnisse");
for($i = 0; $i < $count; $i++) {
   $row = mysql_fetch_assoc($res);
   // tu was mit dem aktuellen Datensatz
}
mysql_close($dbh);
```

In diesem Beispiel wird das Ergebnis von `mysql_num_rows()` dazu verwendet, um bei zu vielen gefundenen Datensätzen abzubrechen. Außerdem werden die gefundenen Datensätze in einer klassischen for-Schleife mit Zählervariable abgearbeitet, da nun bekannt ist, wie viele es insgesamt sind.

Funktion:	`mysql_num_rows()`
Aufgabe:	Ermittelt die Anzahl Datensätze einer Ergebnismenge z. B. eines select-Statement.
Parameter:	1. Ressourcen-Handler, der von `mysql_query()` geliefert wurde.
Rückgabewert:	Anzahl der gefundenen Datensätze.

Tabelle 13.20: Steckbrief Funktion »mysql_fetch_object()«

Falls Sie eine Abfrage auf eine Tabelle starten, deren Felder (Spalten) Ihnen gar nicht genau bekannt sind, z. B. mit `select * from ...`, dann müssen Sie möglicherweise erst einmal die genauen Feldnamen ermitteln. Dieses Problem könnten Sie durch ein describe-Statement in SQL lösen. Beispiel:

```
$dbh = mysql_connect($host, $user, $password);
$sql = "describe webshop.products";
$res = mysql_query($sql, $dbh);
echo "<table rules=\"rows\">";
while($row = mysql_fetch_assoc($res)) {
   echo "<tr>";
   foreach($row as $key => $value)
      echo "<th>$key:</th><td>$value</td>";
   echo "</tr>";
```

```
}
echo "</table>";
mysql_close($dbh);
```

Das Beispiel möchte die Felder und ihre Eigenschaften von `webshop.products` ermitteln und ausgeben. Dazu verwendet es das Statement `describe webshop.products`. Als Ergebnis liefert MySQL für jedes Feld der Tabelle einen Datensatz mit folgenden Feldern: ***Field*** (Feldname), ***Type*** (Datentyp, Feldlänge), ***Null*** (ja/nein, ob das Feld leer sein darf), ***Key*** (Indexinformationen zu diesem Feld), ***Default*** (vorbelegter Wert) und ***Extra*** (zusätzliche Informationen).

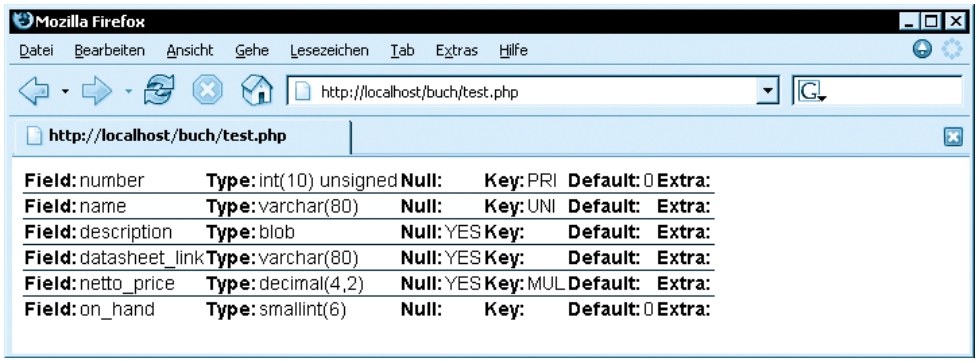


Abbildung 13.2: SQL-describe-Statement als HTML-Tabelle formatiert

Die Abbildung zeigt die Ausgabe des obigen Code-Schnipsels. Bezug ist die Tabelle ***products***, wie sie in ***Abschnitt 13.3.4*** definiert wurde.

Dieses Ergebnis bezieht sich jedoch auf die gesamte Tabelle, da es mit `describe` gewonnen wurde. PHP bietet jedoch auch eigene MySQL-Funktionen an, um Feldinformationen gezielter zu ermitteln. Interessante Informationen bietet beispielsweise die Funktion `mysql_fetch_field()` an, die aus einem aktuellen Datensatz Informationen zu einem bestimmten Feld liefert. Ein Code-Schnipsel zur Demonstration, wie die Funktion einsetzbar ist:

```
$dbh = mysql_connect($host, $user, $password);
$sql = "select number, name from webshop.products";
$res = mysql_query($sql, $dbh);
while($res_info = mysql_fetch_field($res)) {
  if($res_info->numeric == 1)
     echo "{$res_info->name} ist numerisch<br>";
  else
     echo "{$res_info->name} ist nicht numerisch<br>";
}
mysql_close($dbh);
```

Das SQL-Statement ermittelt Datensätze mit den Feldern ***number*** und ***name*** aus der Tabelle ***products***. In einer while-Schleife ermittelt das Beispiel für die einzelnen Felder (in

unserem Beispiel sind es zwei Felder, also *number* und *name*), ob diese numerisch sind oder nicht. Dazu wird `mysql_fetch_field()` aufgerufen. Die Funktion liefert ein Objekt mit Eigenschaften des Feldes zurück. So liefert die Eigenschaft `numeric` 0 oder 1, ob das Feld numerisch ist oder nicht, und die Eigenschaft `name` den Feldnamen. Da es sich um ein Objekt handelt, muss mit `$Variable->Eigenschaft` darauf zugegriffen werden.

Funktion:	`mysql_fetch_field()`
Aufgabe:	Ermittelt Feldeigenschaften der jeweils nächsten Ergebnisspalte oder einer bestimmten Ergebnisspalte. Der Funktionsaufruf sollte einer Variablen zugewiesen werden. Diese enthält anschließend ein Objekt mit den Feldeigenschaften.
Parameter:	1. Ressourcen-Handler, der von `mysql_query()` geliefert wurde. 2. (optional): 0-basierter Offset der Spalte, für die Informationen ermittelt werden sollen.
Rückgabewert:	Ein Objekt mit folgenden Eigenschaften: `name` (Feldname) `table` (Name der Tabelle, zu der das Feld gehört) `max_length` (maximale Länge des Feldes) `not_null` (0 / 1, ob das Feld leer sein darf) `primary_key` (0 / 1, ob das Feld einen Primärindex hat) `unique_key` (0 / 1, ob das Feld einen Unique-Index hat) `multiple_key` (0 / 1, ob das Feld einen normalen Index hat) `numeric` (0 / 1, ob das Feld numerisch ist) `blob` (0 / 1, ob das Feld ein BLOB-Feld ist) `type` (der Feldtyp) `unsigned` (0 / 1, ob das Feld vorzeichenlos ist) `zerofill` (0 / 1, ob das Feld führende Nullen erhält).

Tabelle 13.21: Steckbrief Funktion »mysql_fetch_field()«

Viele Datensätze »seitenweise« ausgeben

Aus Suchmaschinen ist die Funktionalität bekannt, bei Hunderten oder Tausenden von gefundenen Treffern z. B. immer nur 10 oder 50 Datensätze pro Webseite anzubieten und dem Anwender das Blättern zu ermöglichen. Dazu müssen Sie irgendwie speichern, welche »Seite« zuletzt ausgegeben wurde. Am einfachsten ist dies in einem GET-Parameter möglich. Nachfolgendes Beispiel zeigt einen Code-Ausschnitt, der die Blätterfunktionalität auf diese Weise realisiert:

```
$dbh = mysql_connect($host, $user, $password);
$records_per_page = 10;
$sql_all = "select index_name from marketing.contacts";
$res_all = mysql_query($sql_all, $dbh);
$total_records = mysql_num_rows($res_all);
$start = 0;
if(isset($_GET['start']))
   $start = $_GET['start'];
$sql = "select index_name from marketing.contacts limit $start,
        $records_per_page";
```

```
$res = mysql_query($sql, $dbh);
while($row = mysql_fetch_assoc($res)) {
   echo "{$row['index_name']}<br>";
}
if($start > 0) {
   if($start >= $records_per_page)
      $nextstart = $start - $records_per_page;
   else
      $nextstart = 0;
   echo "<a href=\"{$_SERVER['PHP_SELF']}?start=$nextstart\">[&lt;]</a> ";
}
$nextstart = $start + $records_per_page;
if($nextstart <= $total_records)
   echo " <a href=\"{$_SERVER['PHP_SELF']}?start=$nextstart\">[&gt;]</a> ";
mysql_close($dbh);
```

Um die Blätterfunktion sauber zu implementieren, arbeiten wir mit zwei SQL-Statements. Das erste, gespeichert in `$sql_all`, enthält die eigentliche Datenabfrage. Im Beispiel werden alle Indexnamen aus einer Tabelle namens `marketing.contacts` ermittelt. Das Ergebnis dieses Statements, das in `$res_all` gespeichert wird, benötigen wir jedoch nur dazu, um mit `mysql_num_rows()` die Gesamtzahl der zu erwartenden Datensätze zu ermitteln. Diese Zahl wird in `$total_records` gespeichert.

Als Nächstes wird ermittelt, welcher Ausschnitt aus den Gesamtdaten angezeigt werden soll. Wurde das Script mit dem GET-Parameter ***start=*** aufgerufen, wird die Variable `$start` auf den Wert aus dem GET-Parameter gesetzt, ansonsten bleibt ihr Wert 0.

Damit haben wir die Gesamtzahl erwarteter Datensätze (`$total_records`), den Offset-Start zum aktuellen Datensatz (`$start`) und die Anzahl jeweils auszugebender Datensätze, zu Beginn des Script-Ausschnitts definiert in `$records_per_page`.

Damit ausgerüstet starten wir ein zweites SQL-Statement, mit dessen Hilfe nur die nächsten n Datensätze ab einer bestimmten Position ermittelt werden. Das SQL-Statement, gespeichert in `$sql`, erreicht dies durch Anwendung der limit-Klausel von SQL. Als deren erster Wert (Offset, ab dem wievielten Datensatz ausgegeben werden soll) wird die Variable `$start` angegeben und als zweiter Wert (Anzahl maximal auszugebender Datensätze) wird `$records_per_page` angegeben. Damit erhalten wir im Beispiel immer zehn Datensätze. Nur bei den letzten zehn der Tabelle können es weniger sein, je nachdem, wie viele noch übrig sind.

Der Rest des Beispielscripts widmet sich dem Problem, welche Blätter-Links mit welchen GET-Parametern auszugeben sind. Der Rückwärts-Link (dargestellt als `[<]`) wird nur angezeigt, wenn `$start` größer als 0 ist, also im erwarteten Gesamtergebnis noch Datensätze vor der aktuellen Auswahl sind. Der Vorwärts-Link (dargestellt als `[>]`) wird nur dargestellt, wenn die nächsten zehn Datensätze nicht jenseits der insgesamt erwarteten Datensätze liegen, also kein Ergebnis mehr liefern würden.

Ressourcenfreigabe bei umfangreichen Ergebnismengen

Wenn Sie über ein select-Statement zahlreiche Datensätze mit großen Datenmengen selektiert und in PHP verarbeitet haben, ist es zweckmäßig, den dadurch belegten Speicher so früh wie möglich freizugeben, um den Ressourcenverbrauch geringer zu halten. Zwar wird der belegte Speicher nach Ablauf des PHP-Scripts ohnehin freigegeben. Doch wenn Ihr Script nach einer Datenbankabfrage noch diverse andere Aufgaben erledigt, sollte der Speicher zuerst freigegeben werden. Notieren Sie dazu, nachdem eine größere Ergebnismenge vollständig verarbeitet wurde, einfach:

```
mysql_free_result($res);
```

Dabei müssen Sie für `$res` den Namen der Variablen einsetzen, in dem Sie den Rückgabewert von `mysql_query()` gespeichert haben.

Kontrolle bei SQL-Statements ohne Ergebnismenge

Wenn Sie mittels PHP beispielsweise einen neuen Datensatz in eine Tabelle einfügen und dazu das insert-Statement von SQL benutzen, erhalten Sie im Gegensatz zu einem select-Statement keine »Ergebnismenge«. Ebenso wenig, wenn Sie Datensätze mit einem update-Statement ändern oder mit einem delete-Statement löschen oder wenn Sie mit einem alter table-Statement die Struktur einer Tabelle ändern. In einigen Fällen genügt es zu wissen, ob eine entsprechende Operation erfolgreich war oder nicht. In manchen Fällen soll jedoch auch ermittelt werden, wie viele Datensätze betroffen waren, z. B. bei `delete` oder `update` in Verbindung mit einer where-Klausel, die mehrere Datensätze betreffen kann.

Um SQL-Statements wie `insert`, `update`, `delete`, `alter table` usw. abzusetzen, genügt es, `mysql_query()` aufzurufen. Ein Beispiel für `insert`:

```
$dbh = mysql_connect($host, $user, $password);
$field_names = array("number", "name", "description",
                     "datasheet_link", "netto_price",
                     "on_hand");
$field_values = array(
   101234,
   "'Züstelmann Krawuse XLR 200/d Kombi-de luxe'",
   "'Top-Krawusenmodell vom Marktführer'",
   "'http://www.example.org/'",
   169.90,
   18);
$sql = "insert into webshop.products (";
$sql .= implode(",", $field_names);
$sql .= ") values (";
$sql .= implode(",", $field_values);
$sql .= ")";
$res = mysql_query($sql, $dbh);
if($res) echo "Daten eingefügt";
else echo "Fehler beim Einfügen der Daten";
mysql_close($dbh);
```

Beachtung verdient hierbei die Art, wie die Daten für das abzusetzende SQL-insert-Statement zusammengesetzt werden. Sowohl für Feldnamen als auch für Feldwerte wird je ein Array erzeugt. Bei den Feldwerten ist zu beachten, dass Zeichenketten nicht nur PHP-üblich in hohe Anführungszeichen gesetzt werden müssen, sondern innerhalb davon nochmals in Anführungszeichen für MySQL. Im Beispiel haben wir für die inneren Anführungszeichen einfache Anführungszeichen verwendet.

Der Wert der Variablen `$sql` wird in mehreren Anweisungen zusammengebaut (`.=` ist der Operator zum Erweitern einer Zeichenkette um den nachfolgenden Wert). Das Statement wird so zusammengebaut, dass es am Ende lautet:

```
insert into webshop.products
(number,name,description,datasheet_link,netto_price,on_hand) values
(101234,'Züstelmann Krawuse XLR 200/d Kombi-de luxe','Top-Krawusenmodell vom
Marktführer','http://www.example.org/',169.9,18)
```

Durch Aufruf von `mysql_query()` wird das Statement abgesetzt. Da es keine Ergebnismenge liefert, gibt `mysql_query()` nur `true` zurück, falls die Operation erfolgreich war, oder `false`, wenn Fehler auftraten. Im Beispiel fangen wir diesen Rückgabewert in `$res` auf und können mit `if($res)` abfragen, ob die Operation erfolgreich war.

Bei Statements wie `update` oder `delete` können auch mehrere Datensätze betroffen sein. Um herauszufinden, wie viele, steht die Funktion `mysql_affected_rows()` zur Verfügung. Ein Anwendungsbeispiel, das z.B. im Anschluss an ein update-Statement mit where-Klausel notiert werden könnte:

```
$res = mysql_query($sql, $dbh);
if($res) {
   $count = mysql_affected_rows($dbh);
   echo "$count Datensätze geändert";
}
else echo "Fehler beim Ändern der Daten";
```

In der Variablen `$count` wird bei diesem Beispiel gespeichert, wie viele Datensätze betroffen waren.

Datenbank festlegen

Bei unseren bisherigen Beispielen haben wir Tabellen stets über die Datenbank adressiert, also z.B. `webshop.products`. Wenn zuvor `mysql_select_db` () aufgerufen wird, um eine bestimmte Datenbank auszuwählen, ist dies nicht mehr nötig. Ein Beispiel:

```
$dbh = mysql_connect($host, $user, $password);
$db = mysql_select_db("webshop");
$sql = "select * from products";
$res = mysql_query($sql, $dbh);
if($res) {
   $count = mysql_num_rows($res);
   echo "$count Datensätze gefunden";
}
```

Das select-Statement hat in diesem Fall keine Datenbankadressierung mehr nötig, da die Datenbank zuvor ausgewählt wurde.

Funktion:	`mysql_select_db()`
Aufgabe:	Wechselt zu einer bestimmten Datenbank.
Parameter:	1. Name der gewünschten Datenbank. 2. (optional) Datenbank-Handler. Wenn nicht angegeben, wird der zuletzt von einem mysql_connect()-Aufruf erhaltene Handler verwendet.
Rückgabewert:	Bei Erfolg `true`, ansonsten `false`.

Tabelle 13.22: Steckbrief Funktion »mysql_select_db()«

Fehler-Handling

Wenn bei einer Datenbankkommunikation ein Fehler aufgetreten ist, können Sie die Original-Fehlernummern und -meldungen von MySQL in PHP »importieren«. Beispiel:

```
$dbh = mysql_connect($host, $user, $password);
$sql = "select product_name from webshop.products";
$res = mysql_query($sql, $dbh);
if(!$res)
  echo "Fehler: ", mysql_errno(), " (", mysql_error(), ")";
```

Funktion:	`mysql_errno()`
Aufgabe:	Ermittelt die MySQL-Fehlernummer des letzten aufgetretenen Fehlers.
Parameter:	1. (optional) Datenbank-Handler. Wenn nicht angegeben, wird der zuletzt von einem mysql_connect()-Aufruf erhaltene Handler verwendet.
Rückgabewert:	Gibt die Fehlernummer zurück.

Tabelle 13.23: Steckbrief Funktion »mysql_errno()«

Funktion:	`mysql_error()`
Aufgabe:	Ermittelt den MySQL-Fehlertext des letzten aufgetretenen Fehlers.
Parameter:	1. (optional) Datenbank-Handler. Wenn nicht angegeben, wird der zuletzt von einem mysql_connect()-Aufruf erhaltene Handler verwendet.
Rückgabewert:	Gibt den MySQL-Fehlertext zurück.

Tabelle 13.24: Steckbrief Funktion »mysql_error()«

13.5 Praxisbeispiel: Webbasierte Datenverwaltung

In diesem Abschnitt werden wir auf das Beispiel aus ***Abschnitt 13.3*** zurückgreifen. Der Shop-Anbieter soll die Produkte des Shops verwalten, also neue Produkte hinzufügen, oder bestehende Produkte ändern und löschen können. Die webbasierte Lösung einer solchen Datenverwaltung könnte durchaus an die Website mit dem Shop angeschlossen sein, jedoch durch ***htaccess***-Verzeichnisschutz oder einen anderen Mechanismus vor dem öffentlichen Zugriff sicher sein.

13.5.1 Das Handling bei der Datenverwaltung

Am Anfang müssen Überlegungen stehen, die aus Sicht des Benutzers erfolgen. Daten sollen neu angelegt, geändert und gelöscht werden können. Der Benutzer muss also auf jeden Fall die gewünschte Funktion (Neu, Ändern, Löschen) wählen können. Beim Ändern und Löschen muss zunächst ermittelt werden, welcher Datensatz überhaupt geändert oder gelöscht werden soll. Deshalb ist auf jeden Fall eine Datensatzfilterung nach einem Auswahlkriterium erforderlich. Beendete Aktionen sollen bestätigt werden: im Normalfall mit einer Erfolgsmeldung und im Fehlerfall mit einer entsprechenden Meldung.

Wir möchten dem Benutzer auch so viele Klicks wie möglich ersparen. Denn nicht das vom Webdesigner ausgeklügelte Hangeln durch etliche Seiten macht den Benutzer glücklich, sondern alles, was seiner natürlichen Bequemlichkeit entgegenkommt.

Die Oberfläche

Wir entscheiden uns für folgendes Template, das wir zunächst in HTML/CSS als normale HTML-Datei entwerfen. In den HTML-Code fügen wir an den Stellen, in denen PHP Text einfügen soll, selbst gebastelte Platzhalter der Art `[%Platzhaltername%]` ein. Auch diese Technik haben wir in früheren Kapiteln bereits kennen gelernt.

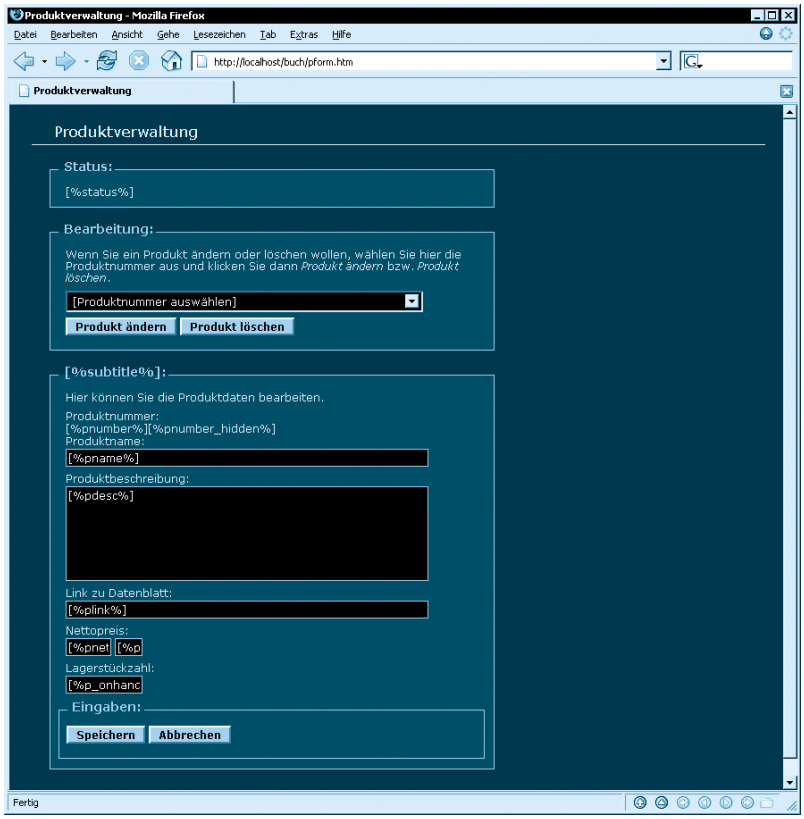


Abbildung 13.3: Geplante Formularseite für Produktverwaltung

Der Vorteil dieser Seite ist, dass es die einzige Seite ist, die der Benutzer kennen muss, d.h., es gibt keine Extraseiten für Statusmeldungen oder dergleichen. Der Bearbeitungsbereich ist in drei Blöcke aufgeteilt.

Der erste Block, *Status*, gibt nach beendeten Aktionen aus, ob die Aktion erfolgreich war oder nicht, und ansonsten eine schlichte Aufforderung.

Der zweite Block, *Bearbeitung*, ist für die Fälle gedacht, in denen der Benutzer ein Produkt ändern oder löschen will. In einer Auswahlliste werden ihm jederzeit alle vorhandenen Produktnummern angeboten. Diese müssen natürlich zur Laufzeit aus der Datenbank ermittelt werden. Nach Auswahl einer Produktnummer kann der Anwender auf *Produkt ändern* oder *Produkt löschen* klicken.

Der untere Block ist das Bearbeitungsformular für die Daten eines Produkts. Wenn der Benutzer zuvor bei *Bearbeitung* ein Produkt ausgewählt und *Produkt ändern* angeklickt hat, wird die Seite so aufgebaut, dass in diesem Formular die vorhandenen Daten des ausgewählten Produkts eingetragen sind. Der Benutzer kann die Daten bearbeiten.

Wurde kein Produkt zum Ändern ausgewählt, so dient das Formular zum Anlegen eines neuen Produkts. Die Felder sind in diesem Fall leer.

Variabel, also mit Platzhaltern belegt, sind der Statustext, die Überschrift des unteren Blocks (abhängig davon, ob er dazu dient, ein zum Ändern ausgewähltes Produkt zu bearbeiten oder ein neues Produkt hinzuzufügen) und die Feldinhalte im unteren Block, die ja beim Ändern eines vorhandenen Produkts mit bisherigen Werten vorbelegt werden müssen. Weitere Platzhalter, die in der Abbildung nicht sichtbar sind, dienen der Script-Steuerung.

Der HTML-Code des Template

Zum besseren Verständnis (z.B. was die Feldnamen betrifft) listen wir den HTML-Code des zuvor entworfenen Template hier auf. Auf das Listing des eingebundenen Stylesheets verzichten wir an dieser Stelle, da es nichts zum Thema beiträgt. Die Template-Datei benennen wir um in *pform.tpl*.

Listing 13.2: HTML-Template »pform.tpl« für Produktverwaltung

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Produktverwaltung</title>
<link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<h1>Produktverwaltung</h1>
<form action="produkte.php" method="post">
<fieldset><legend>Status:</legend>
```

```
<div>
[%status%]
</div>
</fieldset>
<fieldset style="margin-top:15px"><legend>Bearbeitung:</legend>
<div>
<p>Wenn Sie ein Produkt ändern oder löschen wollen, wählen
Sie hier die Produktnummer aus und klicken Sie dann
<em>Produkt ändern</em> bzw. <em>Produkt löschen</em>.</p>
<select name="pnumber_select" style="width:400px">
<option value="0">[Produktnummer auswählen]</option>
[%pnumber_options%]
</select>
<input type="submit" name="edit" class="button"
       value="Produkt ändern">
<input type="submit" name="delete" class="button"
       value="Produkt löschen">
</div>
</fieldset>
<fieldset style="margin-top:15px"><legend>[%subtitle%]:</legend>
<div>
<p>Hier können Sie die Produktdaten bearbeiten.</p>
Produktnummer:<br>
[%pnumber%][%pnumber_hidden%]
</div>
<div>Produktname:<br>
<input type="text" name="pname" class="text"
       style="width:400px" value="[%pname%]">
</div>
<div>Produktbeschreibung:<br>
<textarea name="pdesc" class="text" rows="5" cols="40"
       style="width:400px; height:100px">[%pdesc%]</textarea>
</div>
<div>Link zu Datenblatt:<br>
<input type="text" name="plink" class="text"
       style="width:400px" value="[%plink%]">
</div>
<div>Nettopreis:<br>
<input type="text" name="pnetto_euro" class="text"
       style="width:45px" value="[%pnetto_euro%]">
<input type="text" name="pnetto_cent" class="text"
       style="width:25px" value="[%pnetto_cent%]">
</div>
<div>Lagerstückzahl:<br>
<input type="text" name="p_onhand" class="text"
       style="width:80px" value="[%p_onhand%]">
</div>
<fieldset><legend>Eingaben:</legend>
<p>
<input type="submit" name="[%save%]" class="button"
       value="Speichern">
<input type="submit" name="cancel" class="button"
       value="Abbrechen">
```

```
</p>
</fieldset>
</form>
</body>
</html>
```

13.5.2 Aufgaben des PHP-Scripts

Wir möchten die gesamte Produktverwaltung der Einfachheit halber in einem einzigen PHP-Script unterbringen. Dieses Script muss folgende Abläufe beherrschen:

- Es muss überprüfen, ob POST-Daten vorliegen. Wenn ja, muss es herausfinden, welche Aktion auszuführen ist.
- Wenn neu eingegebene Produktdaten gespeichert werden sollen, muss das Script die eingegebenen Daten auslesen, auf Gültigkeit und Vollständigkeit prüfen und die Daten in die Datenbank schreiben.
- Wenn ein Produkt zum Ändern ausgewählt wurde, muss das Script das Formular zur Dateneingabe mit den Daten des ausgewählten Produkts vorbelegen.
- Wenn geänderte Daten gespeichert werden sollen, muss das Script die Daten wiederum auslesen, auf Gültigkeit und Vollständigkeit prüfen und in die Datenbank zurückschreiben.
- Wenn ein Produkt zum Löschen ausgewählt wurde, muss das Script den Datensatz für das Produkt aus der Datenbank löschen.
- In allen Fällen muss das Script das Template einlesen und die Platzhalter darin situationsabhängig mit korrekten Texten oder Werten füllen.

Die Script-Struktur

Damit das Script seine verschiedenen Aufgaben erfüllen kann, werden wir diese in Funktionen unterbringen, die Unterprogrammcharakter haben. Der zentral und außerhalb aller Funktionen ausgeführte Code dient lediglich zur Definition globaler Variablen und zur Entscheidung, welche Aufgabe zu erfüllen ist. Folgende Funktionen sind vorgesehen:

- Eine Funktion `read_template()` zum Einlesen der Template-Datei.
- Eine Funktion `set_vars()`, die im Template die *[%Platzhalter%]* befüllt.
- Eine Funktion `set_var(name, value)`, die einen einzelnen *[%Platzhalter%]* namens *name* mit einem Wert *value* belegt.
- Eine Funktion `read_db_products()`, die alle Produktnummern aus der Datenbank liest, um die Auswahlliste zum Ändern oder Löschen zu befüllen.
- Eine Funktion `read_db_product($pnumber)`, die alle Daten eines bestimmten, anhand der Produktnummer (*pnumber*) identifizierten Produkts ausliest, um die Bearbeitungsfelder zum Ändern der Produktdaten zu befüllen.

- Eine Funktion `check_data()`, die eingegebene Produktdaten auf Vollständigkeit und Richtigkeit überprüft.
- Eine Funktion `insert_db_data()`, die überprüfte Daten in die Datenbank schreibt.
- Eine Funktion `update_db_data()`, die überprüfte Änderungsdaten in die Datenbank zurückschreibt.
- Eine Funktion `delete_db_data()`, die einen gewünschten Datensatz aus der Datenbank löscht.
- Eine Funktion `show_page()`, welche die Seite am Ende ausgibt.

Das Script ruft sich nach Absenden des Formulars jeweils selbst wieder auf. Anhand der übergebenen Daten kann es erkennen, welche Aktion durchzuführen ist.

Der Script-Code

Nachfolgend das vollständige Listing der Script-Datei *produkte.php*:

Listing 13.3: Quelltext des Scripts produkte.php

```
<?php

# ================================================
# Globale Variablen:

$template_file = "pform.tpl";
$page_content = "";
$action = "none";
$error = "";
$errors = array(
  'template' =>
     "<b>Fehler beim Einlesen der Template-Datei!</b><br>",
  'db_connect' =>
     "<b>Fehler bei Datenbankverbindung!</b><br>",
  'db_read_data' =>
     "<b>Fehler beim Lesen aus der Datenbank!</b><br>",
  'db_insert_data' =>
     "<b>Fehler beim Einfügen in die Datenbank!</b><br>",
  'db_update_data' =>
     "<b>Fehler beim Ändern in der Datenbank!</b><br>",
  'db_delete_data' =>
     "<b>Fehler beim Löschen aus der Datenbank!</b><br>",
  'empty_pnumber' =>
     "<b>Eingaben unvollständig: Produktnummer!</b><br>",
  'no_pnumber' =>
     "<b>Eingaben fehlerhaft: Produktnummer!</b><br>",
  'empty_pname' =>
     "<b>Eingaben unvollständig: Produktname!</b><br>",
  'empty_pnetto_euro' =>
     "<b>Eingaben unvollständig: Nettopreis!</b><br>",
  'empty_pnetto_cent' =>
```

```
      "<b>Eingaben unvollständig: Nettopreis!</b><br>",
  'no_pnetto_euro' =>
      "<b>Eingaben fehlerhaft: Nettopreis!</b><br>",
  'no_pnetto_cent' =>
      "<b>Eingaben fehlerhaft: Nettopreis!</b><br>",
  'empty_p_onhand' =>
      "<b>Eingaben unvollständig: Lagerstückzahl!</b><br>",
  'no_p_onhand' =>
      "<b>Eingaben fehlerhaft: Lagerstückzahl!</b><br>",
);
$status = array(
  'none' => "Wählen Sie ein Produkt zum Ändern oder Löschen
             aus, oder geben Sie Daten für ein neues
             Produkt ein.",
  'new' => "<b>Produkt wurde gespeichert</b>.<br>Sie können
             Produktdaten ändern oder löschen, oder ein weiteres
             Produkt hinzufügen.",
  'delete' => "<b>Produkt wurde gelöscht</b>.<br>Sie können
             Produktdaten ändern oder löschen, oder ein Produkt
             hinzufügen.",
  'edit' => "Sie können andere Produktdaten ändern oder löschen,
             oder die Daten des ausgewählten Produkts bearbeiten.",
  'edited' => "<b>Produktdaten wurden geändert</b>.<br>Sie können
             Produktdaten ändern oder löschen, oder ein Produkt
             hinzufügen."
);
$pnumber_editable = "<input type=\"text\" name=\"pnumber\"
                     class=\"text\" style=\"width:200px\"
                     value=\"\">";
$pnumber_hidden = "<input type=\"hidden\"
                  name=\"pnumber\" value=\"[%n%]\">";


$db_host = "localhost";
$db_user = "";
$db_passwd = "";
$db_name = "webshop";

# ================================================
# Zentralscript:

read_template();
if(!empty($_POST)) {
  if(isset($_POST['edit'])) {
     $action = "edit";
  }
  if(isset($_POST['delete'])) {
     $action = "delete";
     delete_db_data();
  }
  if(isset($_POST['insert'])) {
     check_data();
     if(!empty($error))
```

```
            $action = "new_repeat";
         else {
            $action = "new";
            echo insert_db_data();
         }
      }
      if(isset($_POST['edited'])) {
         $action = "edited";
         check_data();
         if(!empty($error))
            $action = "edit_repeat";
         else {
            $action = "edited";
            update_db_data();
         }
      }
      if(isset($_POST['cancel'])) {
         $action = "none";
      }
   }
   set_vars();
   show_page();


   # ================================================
   # Funktion read_template():

   function read_template() {
      global $template_file, $page_content, $action,
             $error, $errors;
      $page_content = file_get_contents($template_file);
      if(!$page_content)
         $error = $errors['template'];
   }

   # ================================================
   # Funktion read_db_products():

   function read_db_products() {
      global $db_host, $db_user, $db_passwd, $db_name,
             $error, $errors;
      $dbh = mysql_connect($db_host, $db_user, $db_passwd);
      if(!$dbh) {
         $error = $errors['db_connect'];
         return;
      }
      $db = mysql_select_db($db_name);
      $sql = "select number from products order by number";
      $res = mysql_query($sql, $dbh);
      if(!$res) {
         $error = $errors['db_read_data'];
         return;
      }
```

```
    $options = "";
    while($row = mysql_fetch_row($res)) {
      $options .= "<option value=\"{$row[0]}\">";
      $options .= "{$row[0]}</option>\n";
    }
    mysql_close($dbh);
    return($options);
}

# ================================================
# Funktion read_db_product():

function read_db_product($pnumber) {
    global $db_host, $db_user, $db_passwd, $db_name,
           $error, $errors;
    $dbh = mysql_connect($db_host, $db_user, $db_passwd);
    if(!$dbh) {
       $error = $errors['db_connect'];
       return;
    }
    $db = mysql_select_db($db_name);
    $sql = "select * from products where number = ".$pnumber;
    $res = mysql_query($sql, $dbh);
    if(!$res) {
       $error = $errors['db_read_data'];
       return;
    }
    $row = mysql_fetch_assoc($res);
    mysql_close($dbh);
    return($row);
}

# ================================================
# Funktion set_vars(action, error):

function set_vars() {
    global $error, $status, $action,
           $pnumber_editable, $pnumber_hidden;
    $pdata = array('test' => "test");
    if(!empty($error))
        set_var('status', $error);
    else
        set_var('status', $status[$action]);
    set_var('pnumber_options', read_db_products());
    switch($action) {
       case 'none':
       case 'new':
       case 'delete':
       case 'edited':
          set_var('subtitle', "Neues Produkt");
          set_var('pnumber', $pnumber_editable);
          set_var('pnumber_hidden', "");
          set_var('pname', "");
```

```
            set_var('pdesc', "");
            set_var('plink', "");
            set_var('pnetto_euro', "0");
            set_var('pnetto_cent', "00");
            set_var('p_onhand', "0");
            set_var('save', "insert");
            break;
         case 'edit':
            set_var('subtitle', "Produkt bearbeiten");
            $pdata = read_db_product($_POST['pnumber_select']);
            set_var('pnumber', "<p>".$pdata['number']."</p>");
            set_var('pnumber_hidden',
            str_replace("[%n%]", $pdata['number'], $pnumber_hidden));
            set_var('pname', $pdata['name']);
            set_var('pdesc', $pdata['description']);
            set_var('plink', $pdata['datasheet_link']);
            $pnetto_euro = (int) $pdata['netto_price'];
            $pnetto_cent =
                ($pdata['netto_price'] - $pnetto_euro) * 100;
            set_var('pnetto_euro', $pnetto_euro);
            set_var('pnetto_cent', $pnetto_cent);
            set_var('p_onhand', $pdata['on_hand']);
            set_var('save', "edited");
            break;
         case 'edit_repeat':
         case 'new_repeat':
            set_var('subtitle', "Eingaben wiederholen");
            $pdata = read_db_product($_POST['pnumber_select']);
            set_var('pnumber', $_POST['pnumber']);
            set_var('pname', $_POST['pname']);
            set_var('pdesc', $_POST['pdesc']);
            set_var('plink', $_POST['plink']);
            set_var('pnetto_euro', $_POST['pnetto_euro']);
            set_var('pnetto_cent', $_POST['pnetto_cent']);
            set_var('p_onhand', $_POST['p_onhand']);
            if($action == 'new_repeat')
               set_var('save', "insert");
            else if($action == 'edit_repeat')
               set_var('save', "edited");
            break;
      }
   }

   # ================================================
   # Funktion set_var(name, value):

   function set_var($name, $value) {
      global $page_content;
      $search = "[%".$name."%]";
      $page_content = str_replace($search, $value, $page_content);
   }

   # ================================================
```

```
# Funktion show_page():

function show_page() {
   global $page_content;
   echo $page_content;
   exit(1);
}

# ================================================
# Funktion check_data():

function check_data() {
  global $errors, $error;
  if(empty($_POST['pnumber'])) {
     $error = $errors['empty_pnumber']; return;
  }
  if(empty($_POST['pname'])) {
     $error = $errors['empty_pname']; return;
  }
  if(empty($_POST['pnetto_euro'])) {
     $error = $errors['empty_pnetto_euro']; return;
  }
  if(empty($_POST['pnetto_cent'])) {
     $error = $errors['empty_pnetto_cent']; return;
  }
  if(empty($_POST['p_onhand'])) {
     $error = $errors['empty_p_onhand']; return;
  }
  if(is_nan($_POST['pnumber'])) {
     $error = $errors['no_pnumber']; return;
  }
  if(is_nan($_POST['pnetto_euro'])) {
     $error = $errors['no_pnetto_euro']; return;
  }
  if(is_nan($_POST['pnetto_cent'])) {
     $error = $errors['no_pnetto_cent']; return;
  }
  if(is_nan($_POST['p_onhand'])) {
     $error = $errors['no_p_onhand']; return;
  }
  if(strlen($_POST['pnetto_cent']) != 2) {
     $error = $errors['no_pnetto_cent']; return;
  }
  if(strlen($_POST['pnetto_euro']) > 4) {
     $error = $errors['no_pnetto_euro']; return;
  }
}

# ================================================
# Funktion insert_db_data():

function insert_db_data() {
   global $db_host, $db_user, $db_passwd, $db_name,
```

```
            $error, $errors;
    $dbh = mysql_connect($db_host, $db_user, $db_passwd);
    if(!$dbh) {
       $error = $errors['db_connect'];
       return;
    }
    $nettoprice = $_POST['pnetto_euro']."."$_POST['pnetto_cent'];
    $db = mysql_select_db($db_name);
    $sql = "insert into products";
    $sql .= "(number, name, description, datasheet_link, ";
    $sql .= "netto_price, on_hand) values ";
    $sql .= "(".$_POST['pnumber']."', '".$_POST['pname']."', '";
    $sql .= $_POST['pdesc']."', '".$_POST['plink']."', ";
    $sql .= $nettoprice.", ".$_POST['p_onhand'].")";
    $res = mysql_query($sql, $dbh);
    if(!$res)
       $error = $errors['db_insert_data'];
    mysql_close($dbh);
}

# ================================================
# Funktion update_db_data():

function update_db_data() {
    global $db_host, $db_user, $db_passwd, $db_name,
            $error, $errors;
    $dbh = mysql_connect($db_host, $db_user, $db_passwd);
    if(!$dbh) {
       $error = $errors['db_connect'];
       return;
    }
    $nettoprice = $_POST['pnetto_euro']."."$_POST['pnetto_cent'];
    $db = mysql_select_db($db_name);
    $sql = "update products ";
    $sql .= "set number = ". $_POST['pnumber']."', ";
    $sql .= "name = '". $_POST['pname']."', ";
    $sql .= "description = '". $_POST['pdesc']."', ";
    $sql .= "datasheet_link = '". $_POST['plink']."', ";
    $sql .= "netto_price = ". $nettoprice.", ";
    $sql .= "on_hand = ". $_POST['p_onhand']." ";
    $sql .= "where number = ".$_POST['pnumber'];
    $res = mysql_query($sql, $dbh);
    if(!$res)
       $error = $errors['db_update_data'];
    mysql_close($dbh);
}

# ================================================
# Funktion delete_db_data()

function delete_db_data() {
    global $db_host, $db_user, $db_passwd, $db_name,
            $error, $errors;
```

```
    $dbh = mysql_connect($db_host, $db_user, $db_passwd);
    if(!$dbh) {
       $error = $errors['db_connect'];
       return;
    }
    $db = mysql_select_db($db_name);
    $sql = "delete from products ";
    $sql .= "where number = ".$_POST['pnumber_select'];
    $res = mysql_query($sql, $dbh);
    if(!$res)
       $error = $errors['db_delete_data'];
    mysql_close($dbh);
}

?>
```

Globale Variablen

Daten, die möglicherweise austauschbar sind, werden am Beginn des Scripts als Variablen definiert. Das ist übrigens ein Programmierstil, der sich vor allem bei Scripts bewährt hat, die weitergegeben und von anderen Entwicklern an lokale Bedürfnisse angepasst werden.

In unserem Beispielscript werden Variablen definiert für die Template-Datei (`$template_file`), für Fehlermeldungen (Array `$errors`) und für Datenbankverbindungsdaten (`$db_host`, `$db_user`, `$db_passwd`, `$db_name`). Daneben werden noch ein paar weitere Variablen definiert, die zur Script-Steuerung dienen.

Die zentrale Script-Steuerung

Unterhalb der globalen Variablendefinitionen steht außerhalb aller Funktionen der Teil des Scripts, der bei jedem Aufruf ausgeführt wird. Darin wird zunächst die Funktion `read_template()` aufgerufen, um die Template-Datei mit dem HTML-Code und den darin enthaltenen Platzhaltern einzulesen.

Was zu tun ist, entscheidet das Script zunächst abhängig davon, ob POST-Daten vorliegen (`if(!empty($_POST))`). Wenn keine solchen Daten vorliegen, muss nichts weiter getan werden, als die Platzhalter durch ihre Inhalte zu ersetzen. Dazu wird `set_vars()` aufgerufen und anschließend kann die Seite mit `show_page()` ausgegeben werden.

Wenn POST-Daten vorliegen, muss unterschieden werden, welche Aktion auszuführen ist. Dazu haben wir uns im HTML-Code einer Variante bedient, die gar nicht so häufig zum Einsatz kommt, aber eigentlich sehr elegant ist: Wir arbeiten mit mehreren (insgesamt vier) Submit-Buttons innerhalb des Formulars. Die Submit-Buttons unterscheiden sich durch ihren Namen (name-Attribut):

- Der Submit-Button *Produkt ändern* hat das name-Attribut *edit*.
- Der Submit-Button *Produkt löschen* hat das name-Attribut *delete*.
- Der Submit-Button *Speichern* hat als name-Attribut einen Platzhalter *[%save%]*. Wir werden noch sehen warum.
- Der Submit-Button *Abbrechen* hat das name-Attribut *cancel*.

Sogar der Abbrechen-Button ist also als Submit-Button definiert und nicht, wie in HTML sonst üblich, als Reset-Button. Der Grund ist, dass bei einem Abbrechen-Vorgang nicht nur die Formularfelder zurückgesetzt werden müssen, sondern das Script auch wieder in den Zustand gebracht werden soll, in dem keine POST-Daten vorliegen. Dazu ist aber ein Neuaufruf des Scripts erforderlich, was durch die Submit-Funktionalität gewährleistet ist.

Mit Abfragen wie `if(isset($_POST['edit']))` kann die Zentralsteuerung des Scripts ermitteln, ob ein Submit-Button mit dem entsprechenden Namen (also z. B. ***edit***) angeklickt wurde. So weiß das Script, welche Aktion der Anwender ausgelöst hat und was zu tun ist. Intern merkt sich das Script die auszuführende Aktion in der Variablen `$action`.

Unter anderem fragt das Script die beiden Feldnamen `insert` und `edited` ab. Auch das sind Namen eines Submit-Buttons und zwar des Buttons mit dem Platzhalternamen *[%save%]*. Das Script versorgt den Platzhalter beim Aufruf zuvor mit einem der Werte `insert` oder `edited`, abhängig davon, ob über die Eingabefelder gerade ein neuer Datensatz eingegeben werden kann oder ob darin ein zuvor ausgewählter, vorhandener Datensatz bearbeitet werden kann.

Beide Fälle, also `insert` oder `edited`, bedeuten, dass der Anwender Daten eingegeben hat und auf den Button ***Speichern*** geklickt hat. Deshalb wird in den entsprechenden Abfragezweigen die Funktion `check_data()` aufgerufen, welche die Eingaben auf Vollständigkeit und Plausibilität überprüft. Falls diese Funktion einen Fehler findet, füllt sie die Variable `$error` mit einer entsprechenden Fehlermeldung aus dem Array `$errors`. Durch die Abfrage `if(!empty($error))` kann das Script deshalb ermitteln, ob die eingegebenen Daten fehlerhaft oder in Ordnung sind. Wenn sie fehlerhaft sind, werden `$action` die Werte `new_repeat` bzw. `edit_repeat` zugewiesen. Damit merkt sich das Script, dass die via POST erhaltenen Daten nochmals in die Formularfelder eingetragen werden sollen, um dem Anwender die Korrektur der Daten zu ermöglichen.

Sind die eingegebenen Daten in Ordnung, werden die Funktionen `insert_db_data()` bzw. `edit_db_data()` aufgerufen, welche die Daten in die Datenbank einfügen bzw. zurückschreiben. Wurde der Submit-Button ***Produkt löschen*** angeklickt, wird `delete_db_data()` aufgerufen, um den ausgewählten Datensatz aus der Datenbank zu löschen.

Die Funktionen set_vars() und set_var()

Die Funktion, welche die zentrale Script-Steuerung unmittelbar unterstützt, ist `set_vars()`. Mittelpunkt der Funktion ist ein switch-case-Konstrukt, das den Wert der zuvor in der Zentralsteuerung versorgten Variablen `$action` abfragt. Bei den Werten `none` (keine POST-Daten vorhanden), `new` (zuvor eingegebener Datensatz ist in der Datenbank einzufügen), `delete` (zuvor ausgewählter Datensatz ist zu löschen) oder `edited` (zuvor eingegebener Datensatz ist in der Datenbank zu ändern) kann das Formular so angezeigt werden, dass es zur Eingabe eines neuen Datensatzes geeignet ist. Auch die Legendenüberschriften usw. werden entsprechend gesetzt.

Wenn ein Datensatz geändert werden soll (`$action` hat in diesem Fall den Wert `edit`), müssen die Daten des gewünschten Datensatzes zunächst aus der Datenbank gelesen werden. Dazu wird `read_db_product()` aufgerufen. Als Parameter wird der via POST erhaltene

Wert aus der select-Liste zur Auswahl einer Produktnummer übergeben. Das Ergebnis der Datenbankabfrage, also des Aufrufs von `read_db_product()`, wird in der Variablen `$pdata` gespeichert. Es handelt sich dabei um einen assoziativen Array mit allen Feldnamen und Inhalten des Datensatzes. Die Formularfelder werden mit den entsprechenden Werten des Arrays gefüllt.

Einen dritten Fall stellen die Aktionszustände `new_repeat` und `edit_repeat` dar. In diesem Fall müssen nicht Daten aus der Datenbank, sondern die mit POST erhaltenen, jedoch als fehlerhaft erkannten Daten noch einmal in die Formularfelder eingefügt werden, um dem Anwender die Korrektur zu ermöglichen.

Alle Einfügevorgänge werden durch Aufrufe der Funktion `set_var()` abgewickelt. Diese sucht in der Variablen `$page_content`, in die das Template eingelesen wurde, nach einem Platzhalter mit dem Namen des ersten ihr übergebenen Parameters. Diesen Platzhalter ersetzt sie durch den zweiten ihr übergebenen Wert.

Die Datenbankfunktionen

In insgesamt fünf der Funktionen wird mit der Datenbank kommuniziert: wie schon an den Namen erkennbar in `read_db_products()`, `read_db_product()`, `insert_db_data()`, `update_db_data()` und `delete_db_data()`. Alle Funktionen stellen jeweils eine Datenbankverbindung her und schließen diese am Ende wieder. Bei Verbindungs- oder sonstigen Kommunikationsfehlern wird die Variable `$error` versorgt und der Vorgang wird abgebrochen. In der Funktion `set_vars()` wird abgefragt, ob `$error` einen Inhalt hat. Abhängig davon wird im Statusbereich der Seite, also im ersten Block, die Fehlermeldung gut sichtbar ausgegeben.

Alle fünf Datenbankfunktionen setzen nur ein einziges SQL-Statement an den MySQL-Server ab. In der Variablen `$sql` wird das Statement in den Funktionen jeweils zusammengebaut. Beachten Sie dabei, wie mit einfachen Anführungszeichen innerhalb der doppelten PHP-Zeichenkettenanführungszeichen gearbeitet wird, um Zeichenkettenwerte in SQL zu definieren. Beispielausschnitt:

```
$sql = "insert into products";
$sql .= "(number, name, description, datasheet_link, ";
$sql .= "netto_price, on_hand) values ";
$sql .= "(".$_POST['pnumber']."', '".$_POST['pname']."', '";
$sql .= $_POST['pdesc']."', '".$_POST['plink']."', ";
$sql .= $nettoprice.", ".$_POST['p_onhand'].")";
```

Im zweiten Teil des Statements, wo in runde Klammern eingeschlossen die Werte notiert werden, die in den neuen Datensatz einzufügen sind, wird mit lauter Zeichenkettenverknüpfungen (Punkt-Operator) von Werten des $_POST-Arrays, also der erhaltenen Formulardaten, mit literalen Verbindungszeichenketten (`"..."`) der Gesamtwert von `$sql` »errichtet«. Die einfachen Anführungszeichen sind dabei so gesetzt, dass SQL-Zeichenkettenwerte, wie etwa der Inhalt von `$_POST['pname']`, in solche Anführungszeichen eingeschlossen ist, weil das entsprechende Datenbankfeld *name* ein varchar-Feld für Zeichenketten ist. Versuchen Sie unbedingt zu verstehen, wie hier in der Variablen `$sql`

die Zeichenkette für das SQL-Statement so zusammengesetzt wird, dass es aus SQL-Sicht syntaktisch korrekt ist. Diese Art, ein SQL-Statement aus Formulardaten zusammenzusetzen, werden Sie bei der Kombination »Formulardaten für Datenbankinhalte« immer wieder benötigen.

13.5.3 Optimierungen für mehr Sicherheit

Das Praxisbeispiel zeigt, dass eine komplette Datenverwaltung, selbst wenn nur die Daten einer einzigen Datenbanktabelle davon betroffen sind, gar nicht so trivial ist, vor allem, je professioneller sie sein soll – also wenn z. B. auch die Überprüfung von Eingabedaten und ein entsprechendes Fehler-Handling dazu gehören soll. Dennoch haben wir im Beispiel noch keine spezielleren Sicherheitsvorkehrungen gegen böswillige Benutzer bedacht. Um dieses Thema soll es abschließend gehen.

SQL-Injection

Unter dem Begriff ***SQL-Injection*** versteht man den Versuch, über Schnittstellen wie Formularfelder, deren Inhalt in eine Datenbank eingefügt werden soll, Missbrauch zu treiben. Dabei wird versucht, in das Formularfeld einen Inhalt einzugeben, der beim script-gesteuerten Zusammensetzen etwa eines select-Statements letztlich ein SQL-Statement ergibt, das etwas ganz anderes bewirkt als das beabsichtigte.

Angenommen, wir hätten in unserem Beispiel anstelle der select-Liste mit den Produktnummern ein einfaches Eingabefeld, in dem der Anwender eine Produktnummer eingeben kann. Dann würde im PHP-Script an irgendeiner Stelle so etwas Ähnliches stehen wie:

```
$sql = "select * from poducts where number = ".$_POST['pnumber'];
```

Dabei wäre `pnumber` der Name des `<input>`-Feldes in HTML.

Gibt der Anwender in dem Feld so etwas wie 101234 ein, dann lautet das tatsächliche SQL-Statement am Ende:

```
select * from poducts where number = 101234;
```

Das ist in Ordnung und so gewünscht. Angenommen jedoch, der Anwender gibt in dem Feld Folgendes ein:

```
101234; truncate webshop.products;
```

Dann wird ohne weitere Prüfung folgendes SQL-Statement erzeugt:

```
select * from poducts where number = 101234; truncate webshop.products;
```

Ein zweites SQL-Statement wird also »heimlich mitgeliefert«, das alles tun darf, wozu der MySQL-Benutzer, unter dem das Script sich anmeldet, berechtigt ist. Dabei ist das Löschen einer Tabelle mit `truncate` noch eine der harmloseren Möglichkeiten. SQL erlaubt

über eine spezielle Syntax etwa auch das Ausführen von Kommandos und Programmen auf dem Server-Rechner. Mit etwas Geschick und Glück kann es einem Anwender so gelingen, das gesamte System lahm zu legen und Daten im großen Stil zu vernichten.

Ebenfalls beliebt ist das Einfügen eines inklusiven OR-Parts, der ein Statement immer wahr macht. In der PHP-Dokumentation wird das am Beispiel eines Passworts demonstriert. Beispiel:

```
select * from users where user='{$_POST['username']}' and
password='{$_POST['password']}'
```

Angenommen, der Anwender gibt im Feld *username* einen gültigen Benutzernamen wie `hansmeier` und im Feld *password* `' or 1=1` ein. Dann wird folgendes SQL-Statement erzeugt:

```
select * from users where user='hansmeier' and password='' or 1=1
```

Die Folge ist, dass der Zugang erlaubt wird, obwohl nur der Benutzername existiert, nicht aber das Passwort.

In PHP ist es relativ einfach, solche SQL Injections zu vermeiden.

Unsichere Variante:

```
$sql = "select * from poducts where number = ".$_POST['pnumber'];
```

Sichere Variante:

```
$sql = "select * from poducts where number =
".mysql_real_escape_string($_POST['pnumber']);
```

Die Funktion `mysql_real_escape_string()`, die seit PHP-Version 4.3.0 verfügbar ist, maskiert spezielle Zeichen in einer an den SQL-Interpreter übergebenen Zeichenkette so, dass keine Seiten-Statements oder andere Kommandos mehr ausführbar sind. Angewendet werden sollte sie vor allem auf Inhalte aus Formularfeldern, also auf Werte, die in den superglobalen Arrays `$_GET` oder `$_POST` gespeichert sind und in SQL-Statements eingebaut werden sollen.

Funktion:	`mysql_real_escape_string()`
Aufgabe:	Maskiert spezielle Zeichen in einem SQL-Statement so, dass keine SQL Injection mehr möglich ist.
Parameter:	1. zu maskierende Zeichenkette. 2. (optional) Datenbank-Handler. Wenn nicht angegeben, wird der zuletzt von einem mysql_connect()-Aufruf erhaltene Handler verwendet.
Rückgabewert:	Gibt die maskierte Zeichenkette zurück.

Tabelle 13.25: Steckbrief Funktion »mysql_real_escape_string()«

Datenintegritätsprüfung

In unserem Praxisbeispiel sorgt die Funktion `check_data()` für die Überprüfung eingegebener Formulardaten. Dabei wird überprüft, ob alle Pflichtfelder einen Inhalt haben (z. B. `if(empty($_POST['pnumber'])) $error = ...`). Bei numerischen Feldern wie Produktnummer, Nettopreis oder Lagerstückzahl wird geprüft, ob der eingegebene Wert nicht numerisch ist (`if(is_nan($_POST['pnumber'])) $error = `). Bei den eingegebenen Euro- und Cent-Werten für den Nettopreis wird schließlich noch geprüft, ob die Euroangabe nicht mehr als vierstellig ist und ob die Cent-Angabe wie gefordert zweistellig ist.

Für unsere Zwecke genügen diese Prüfungen. Angenommen jedoch, ein von Benutzern eingegebener Wert wird von einem cron-Job-Script aus der Datenbank ausgelesen, um ihn als Parameter an ein Shellkommando zu übergeben. In diesem Fall sollte die Integritätsprüfung auf jeden Fall so weit gehen, dass Zeichen wie > oder | erkannt und verweigert werden. Nur so kann verhindert werden, dass der cron-Job heimlich zugeführte Pipes oder Umleitungen ausführt.

Auch wenn Anwender beispielsweise eine URL-Adresse in einem Formularfeld eingeben können, wie z. B. »Ihre Homepage-Adresse«, sollte eine erweiterte Integritätsprüfung stattfinden, um zu verhindern, dass etwa in einem Gästebuch am Ende Hyperlinks erscheinen, die zu Adressen führen, an denen Dialer aktiviert werden oder Ähnliches. Am sichersten ist es in diesem Fall, vor dem Akzeptieren der Daten im Script eine HTTP-Socket-Verbindung zur angegebenen Adresse durchzuführen, die Server-Antwort zu analysieren und gegebenenfalls zu reagieren. Keine triviale Aufgabe, doch andere Seitenbesucher werden dankbar sein, wenn sie nur »geprüfte« Links vorfinden.

HTML-Code in Eingabefeldern

In Feldern der Typen `<input type="text">` oder `<textarea></textarea>` kann jeder Anwender, wenn er denn will, beliebigen HTML-Code eingeben, z. B. auch einen Script-Bereich mit einem JavaScript, das auf eine andere Adresse umleitet oder einen ausführenden Browser zum Absturz bringt, oder ein eingebettetes Frame-Fenster mit einer Adresse von einem anderen Server. Die böswilligen Möglichkeiten in dieser Rubrik lassen sich um eine lange Liste erweitern. Zum Ärgernis werden solche böswilligen Eingaben dann, wenn der so gespeicherte Formularfeldinhalt zu einem anderen Zeitpunkt bei einem anderen Benutzer aus der Datenbank ausgelesen und ungeprüft als »Text« im Browser ausgegeben wird.

In PHP haben Sie mehrere Möglichkeiten, eingeschleusten HTML-Code zu entschärfen: Eine Möglichkeit besteht darin, auf den Formularfeldinhalt vor dem Speichern in der Datenbank, oder vor der Ausgabe eines aus der Datenbank geholten Wertes, die betroffene Zeichenkette an die PHP-Funktion `htmlspecialchars()` als Parameter zu übergeben. Damit wird der HTML-Code als Code im Browser ausgegeben, aber nicht mehr vom HTML-Parser des Browsers geparst. Noch radikaler ist die Funktion `strip_tags()`, die einfach alles HTML-Markup aus der betroffenen Zeichenkette entfernt. Elementinhalte, also z. B. der Code eines eingegebenen JavaScripts, bleibt dabei als normaler Text übrig. Die beiden PHP-Funktionen haben wir in ***Abschnitt 12.4.3*** bereits näher behandelt.

Eigenen Code benötigen Sie dann, wenn Sie nur bestimmte, harmlose HTML-Auszeichnungen erlauben möchten, andere dagegen nicht. Die nachfolgende Funktion ist ein Beispiel, wie sich so etwas realisieren lässt:

```
function deny_preg_html() {
   $arg_list = func_get_args();
   if(count($arg_list) == 0)
      return(false);
   $str = $arg_list[0];
   for($i = 1; $i < count($arg_list); $i++) {
      $pattern = "/".$arg_list[$i]."/";
      $str = preg_replace($pattern, "", $str);
   }
   return $str;
}
```

Die Funktion *deny_preg_html()* erwartet eine beliebig lange Liste von Parametern. Der erste Parameter muss die Zeichenkette sein, die behandelt werden soll. In den Parametern 2 bis n können reguläre Ausdrücke übergeben werden, die HTML-Code entfernen sollen. Eine Beispielzeichenkette und ein Beispielaufruf:

```
$test = "<h1>Test</h1>Ein Test mit <script type=\"text/
javascript\">location.href='http://www.example.org/'</script> und <b>fettem
<i>kursivfettem</i></b> Text";
$test = deny_preg_html($test, "<script.*?>.*?<\/script>", "<h1>", "<\/h1>");
echo $test;
```

Aus dem Inhalt von `$test` werden durch Aufruf von `deny_preg_html()` das komplette JavaScript samt JavaScript-Code sowie die h1-Überschriften-Tags entfernt. Die Formatierungen mit den b- und i-Elementen bleiben dagegen unversehrt.

Teil 6 – XML

XML (Extensible Markup Language, zu Deutsch: erweiterbare Auszeichnungssprache) ist ein Regelwerk zur semantisch sinnvollen Beschreibung beliebiger Daten. Mit Hilfe entsprechender Software lassen sich solche Daten verarbeiten. Im Bereich der Webentwicklung gewinnt XML an verschiedensten Stellen immer mehr Bedeutung.

14 XML für Webentwickler

Seit Jahren wird die Spezifizierung der XML-Sprachenwelt vom W3-Konsortium wie eine vorrangige Chefsache behandelt. Eine kaum noch überschaubare Flut an Dokumenten mit insgesamt etlichen tausend Seiten Umfang ist dabei entstanden. Die Anstrengungen haben das gemeinsame Ziel, die Basis für ein »semantisches Web« zu errichten.

Und während sich immer noch viele Praktiker im Web fragen, was unter dem »semantischen Web« eigentlich genau zu verstehen sei, tauchen bereits neue Schlagwörter im Netz auf: Von »Web 2.0« ist da die Rede und von »social software«. Geht man diesen Schlagwörtern nach, findet man heraus, dass die dabei zum Einsatz kommenden Technologien ebenfalls häufig mit XML zu tun haben.

Grund genug, sich als Webentwickler mit XML so weit vertraut zu machen, um damit:

- Daten eigener Webanwendungen XML-konform speichern zu können,
- XML-Daten in eigenen Webanwendungen verarbeiten zu können,
- neuere, das Web bereichernde Features wie RSS-Feeds oder Webservices in eigene Webanwendungen integrieren zu können.

Links zu weiteren Informationen:

W3C-Übersicht zu XML:
http://www.w3.org/XML/
XML in 10 points (Kurzeinführung des W3C):
http://www.w3.org/XML/1999/XML-in-10-points
XML 1.0 Empfehlung:
http://www.w3.org/TR/REC-xml/
XML 1.0 Empfehlung, deutsche Übersetzung:
http://www.edition-w3c.de/TR/REC-xml
XML 1.1 Empfehlung:
http://www.w3.org/TR/xml11/
Wikipedia-Artikel über XML:
http://de.wikipedia.org/wiki/XML

14.1 Einsatz- und Verarbeitungsmöglichkeiten

HTML kennen Sie bereits und Sie wissen, dass es dort bestimmte Elemente gibt, die manchmal durch ein Standalone-Tag repräsentiert werden, meistens jedoch durch ein Anfangs- und ein End-Tag, die einen Inhalt einschließen. Der Inhalt besteht manchmal nur aus Text, manchmal nur aus anderen HTML-Elementen und hin und wieder auch aus einer Mischung von beidem. Außerdem können in den Anfangs-Tags der Elemente bestimmte erlaubte Attribute notiert werden.

XML ist zunächst einmal nichts anderes als die Erweiterung dieser Grundregeln auf frei benennbare Elemente und Attribute. Stellen Sie sich einfach Quelltexte mit Abschnitten wie `<Tonart Typ="C-Dur"><Note Laenge="1/4" Ton="F" />` ... oder `<Person><ID>manuel_schmitt_01 </ID><Vorname>Manuel </Vorname><Zuname>Schmitt</Schmitt><Kind_von_ID> simon_schmitt_ 01</Kind_von_ID>` ... vor. Dann verstehen Sie schnell, dass sich mit diesen einfachen Auszeichnungsregeln beliebige Daten beliebig genau beschreiben lassen, egal ob es sich um Musikstücke, Stammbäume, multimediale Abläufe, Vektorgrafiken, Schaltpläne, Bücher, Nachrichten oder Workflow-Beschreibungen handelt.

Wichtig: XML ist selbst keine Auszeichnungssprache, sondern ein Regelwerk zur Definition von Auszeichnungssprachen – also nicht etwa ein »Konkurrenzformat für HTML«, sondern ein formaler Standard, an den sich alle XML-basierten Sprachen halten – XHTML ebenso wie andere.

XML oder DBMS

Die universelle Verwendbarkeit von XML führt zu der Frage, ob XML nicht auch Datenbank-Management-Systeme (DBMS) wie MySQL überflüssig macht. Das ist aus mehreren Gründen nicht der Fall:

XML ist keine Software, sondern nur ein Beschreibungsmodell für Daten. Ein DBMS ist dagegen immer auch ein Software-System, das zusätzliche Leistungen erbringt, wie beispielsweise das besonders schnelle Auffinden von Daten (Performanz) oder das besonders sichere Speichern von Daten im Mehrbenutzerbetrieb.

Aber auch bei der Datenmodellierung ist XML nicht zwangsläufig einem DBMS überlegen. Die heute vorherrschenden relationalen Datenbanksysteme sind vor allem bei serienartigen Daten effizient, bei denen ebenso serienartige Abhängigkeiten und Beziehungen bestehen, z. B. bei Kunden, Aufträgen und Rechnungen. Eine besondere Stärke von XML ist dagegen die Beschreibung baumartig strukturierbarer Daten.

Keine automatische Datenverarbeitung

Mit XML lassen sich Daten so beschreiben, wie es aus Menschensicht inhaltlich sinnvoll erscheint. Genau das ist gemeint, wenn von »semantisch« die Rede ist, wie beim Schlagwort »semantisches Web«.

Auf diese Weise lassen sich Daten jedoch nur beschreiben, nicht automatisch verarbeiten. Es gibt keine Software, die bei einem Quelltext wie `<Tonart Typ="C-Dur"><Note Laenge="1/`

4" Ton="F" /> ... automatisch Töne erzeugt. Software, die dazu in der Lage ist, muss erst geschrieben werden. Neben der Fähigkeit, Töne zu erzeugen, muss sie ein bestimmtes XML-Format erkennen und korrekt auslesen können.

Das gilt auch für Webbrowser. Diese zeigen, wenn Sie als Anwender eine beliebige XML-Datei als URI aufrufen, die darin enthaltenen Daten einfach als Quelltext an (vorausgesetzt, bei der Kommunikation zwischen Webserver und Browser wird überhaupt entschieden, die Daten als Text zu senden). Erst durch zusätzliche »Intelligenz« kann der Browser solche Daten im Sinne einer nützlichen Anwendung verarbeiten:

- Durch Zuordnung von Stylesheet-Definitionen ist es beispielsweise möglich, dem Browser mitzuteilen, wie er Elemente wie z. B. <chapter>...</chapter> oder <teaser>...</teaser>, mit denen er ansonsten nichts anfangen könnte, optisch darstellen soll.
- Durch Installation eines geeigneten Plug-in kann ein Browser befähigt werden, bestimmte XML-Daten zu interpretieren. So gibt es beispielsweise Browser-Plug-ins für SVG-Grafiken oder für RSS-Newsfeeds (beides basiert auf XML).

Individuelle und öffentlich standardisierte XML-Sprachen

Da alle XML-Daten grundsätzliche formale Regeln einhalten, kann die Verarbeitung von XML-Daten auf allgemeine Konzepte zurückgreifen. So stellen Scriptsprachen wie PHP von Haus aus Klassen oder Module zur Verfügung, mit deren Hilfe ein Programmierer XML-Daten verarbeiten kann. Das spart Programmieraufwand und reduziert Fehlerquellen.

Noch einfacher wird es, wenn keine individuellen XML-Sprachen eingesetzt werden, sondern öffentlich standardisierte. XHTML, SVG (Vektorgrafik), MathML (mathematische Ausdrücke) oder WML (für WAP/Mobiltelefonanwendungen) gehören zu den öffentlich standardisierten XML-Sprachen. Öffentlich standardisierte Sprachen haben aus Software-Sicht den Vorteil, dass es häufig schon fertige Klassen und Module gibt, mit denen die Datenverarbeitung der jeweiligen XML-Sprache möglich ist. Dadurch reduziert sich der Programmieraufwand nochmals. In ***Abschnitt 14.3*** werden wir wichtige, öffentlich standardisierte XML-Sprachen kennen lernen.

Eigene XML-Sprachen zu definieren, ist dann die beste Lösung, wenn keine öffentlich standardisierte XML-Sprache existiert, die das Gleiche leistet oder wenn ein begründbares Interesse daran besteht, ein ganz eigenes Namens- und Strukturschema zu verwenden. Eigene XML-Sprachen sind etwa auch eine geeignete Lösung, wenn für den angestrebten Verwendungszweck nur ein kleiner Teil des Sprachumfangs einer bereits existierenden XML-Sprache benötigt wird. In jedem Fall empfiehlt es sich jedoch, vor dem Entwickeln einer eigenen Sprache zu recherchieren, ob es bereits öffentlich standardisierte Sprachen für den angestrebten Verwendungszweck gibt und ob diese Sprachen für die eigenen Zwecke geeignet sind.

Datenkonvertierung (Transformation)

Eine der größten Stärken von XML besteht darin, dass Daten wegen der grundsätzlich einzuhaltenden Regeln hervorragend von einem XML-Format in ein anderes Format übertragen werden können. Für diesen Vorgang, in der XML-Welt als Transformation bezeichnet, gibt es sogar eine eigene XML-Sprache: XSLT.

In der Praxis werden XML-Daten vor allem serverseitig von ihrem Ausgangsformat in (X)HTML transformiert, um die Daten für die Ausgabe im Browser vorzubereiten. Wie das funktioniert, beschreiben wir in ***Abschnitt 14.7*** genauer.

14.1.1 XML-Editoren

Wenn Sie nur hin und wieder XML-Dateien auf Quelltextebene bearbeiten, genügt ein guter Universal-Editor, der idealerweise Syntax-Highlighting für XML anbietet und ansonsten mit den heute üblichen Komfortfunktionen wie Fenstersplitting, Textbausteine, Suchen und Ersetzen auch über mehrere Dateien usw. ausgestattet ist.

Unter Windows kommen hierzu führende Universal-Editoren wie EditPlus, UltraEdit oder Notepad++ in Frage.

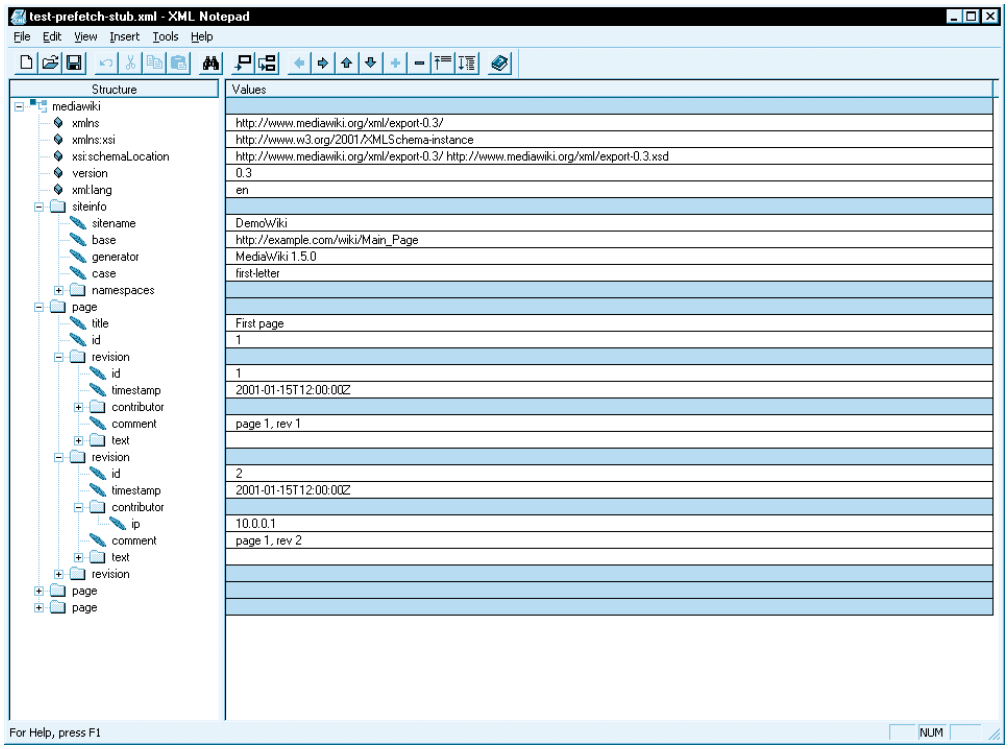


Abbildung 14.1: XML-Editor mit Baumstrukturbearbeitung (hier: XML Notepad)

Daneben gibt es eine ganze Reihe XML-spezifischer Bearbeitungs-Software auf dem Markt. Ein Konzeptschwerpunkt dieser Produkte ist, dass die Baumstruktur von XML-Dokumenten visualisiert wird. Elemente und Attribute erscheinen in einer explorer-artigen Ansicht und lassen sich gezielt ansteuern und bearbeiten. In Zusammenhang mit Stylesheet-Definitionen bieten einige Produkte auch Wysiwyg-Ansichten an. Einige unterstützen außerdem beim Erstellen von DTDs oder XML-Schemata durch grafische Visualisierung.

Für Universal-Editoren siehe auch der Link-Kasten in ***Abschnitt 2.1.2*** dieses Buchs.

Links zu XML-spezifischen Editoren für MS Windows

<oXygen/> (en, Shareware): *http://www.oxygenxml.com/*
Peter's XML-Editor (en, Freeware): *http://www.iol.ie/~pxe/*
UltraXML (en, Shareware): *http://www.webxsystems.com/UltraXML.htm*
XRay (en/en, Freeware): *http://www.architag.com/xray/*
XMetal (en/de, Shareware): *http://www.xmetal.com/products/*
XML Spy (en/de, Shareware): *http://www.altova.com/products_ide.html*
XML NotePad (en, Freeware): *http://www.webattack.com/get/xmlnotepad.html*

Links zu XML-spezifischen Editoren für Macintosh

<oXygen/> (en, Shareware): *http://www.oxygenxml.com/*
ElfData (en, Shareware): *http://www.elfdata.com/xmleditor/*

Links zu code-basierten Editoren für Linux

<oXygen/> (en, Shareware): *http://www.oxygenxml.com/*
Emacs XML-Erweiterung (en, Freeware): *http://xae.sunsite.dk/*

14.1.2 XML-Parser

HTML wird vor allem von Browsern und vergleichbaren Webclients interpretiert. Zu diesem Zweck verfügen die Browser über einen integrierten HTML-Parser, der HTML-Code zerlegt und daraus eine Bildschirmausgabe generiert. Wenn CSS im Spiel ist, werden entsprechende Style-Definitionen berücksichtigt. Wer Webseiten entwickelt, weiß, dass jeder Webanwender irgendeinen Browser benutzt. Die Frage nach dem Vorhandensein eines HTML-Parsers ist damit von vorneherein beantwortet.

Bei XML ist die Sache nicht so eindeutig. Manchmal sollen auch XML-Daten einfach im Browser angezeigt werden. Viele Browser, darunter der MS Internet Explorer 6.x, Mozilla Firefox 1.x und Opera 8.x, können XML-Dokumente, deren Elemente mit Hilfe von CSS formatiert werden, als attraktives Bildschirmergebnis anzeigen. Der Grund ist, dass diese Browser über einen integrierten XML-Parser verfügen.

Daneben gibt es aber auch viele Anwendungsfälle für XML, bei denen kein Endanwender im Spiel ist. So beispielsweise, wenn eine Webanwendung Nutz- oder Konfigurationsdaten in XML-Form einliest oder wenn Webanwendungen untereinander Daten in XML-Formaten austauschen. Für solche Zwecke gibt es separate XML-Parser. Diese arbeiten

software-technisch betrachtet wie ein Filter oder Treiber. Sie erhalten XML-Code als Input. Der Output besteht in einer neuen Datenstruktur, die sich durch das Zerlegen des XML-Codes ergibt. Diese Datenstruktur kann von einer weiterverarbeitenden Software genutzt werden, z. B. von einem PHP-Script. Was hier vereinfacht als »Datenstruktur« bezeichnet wird, ist in Wirklichkeit die Bedienung einer Programmierschnittstelle. Hierbei konkurrieren zwei bekannte Varianten: SAX (Simple API for XML) und das bereits im Zusammenhang mit JavaScript erläuterte DOM (Document Object Model). Wie Sie XML serverseitig einsetzen und mit PHP verarbeiten können, wird in ***Abschnitt 14.8*** näher beschrieben.

Unterschieden werden muss ferner zwischen Parsern, die validieren, und solchen, die das nicht tun. Validierende Parser überprüfen XML-Code, der sich auf bestimmte Dokumenttypdefinitionen bezieht, gegen diese Dokumenttypdefinitionen. Bei Fehlern wie nicht in der DTD definierte Elemente oder dort nicht erlaubte Elementverschachtelungen erzeugen sie eine Fehlermeldung. Nicht-validierende Parser überprüfen dagegen nur, ob die allgemeinen XML-Regeln eingehalten werden.

Links zu bekannten separaten XML-Parsern

Expat (en, OpenSource): *http://sourceforge.net/projects/expat/*
Schnell und nah am XML-Standard, unterstützt jedoch nur SAX, nicht DOM, nicht validierend

JAXP (en, OpenSource): *http://sourceforge.net/projects/expat/*
Java-basiert, unterstützt SAX und DOM 1.0, enthält auch einen XSLT-Compiler, validierend und nichtvalidierend

Xerxes (en, OpenSource): *http://xml.apache.org/index.html*
Java- oder C++-basiert, unterstützt SAX und DOM 1.0/2.0
validierend und nichtvalidierend

14.2 Aufbau von XML-Dokumenten

Mit dem grundsätzlichen Aufbau von XML-Dokumenten müssen Sie sich in jedem Fall vertraut machen – egal ob Sie eigene XML-Sprachen definieren oder einfach nur bereits vorhandene XML-Daten bearbeiten oder verarbeiten wollen. In diesem Abschnitt erhalten Sie das erforderliche Basiswissen.

14.2.1 Elemente, Tags und Attribute

Das formale Repertoire von XML besteht aus Elementen mit oder ohne eingeschlossenen Inhalt, repräsentiert durch Tags und optional durch Attribute mit Wertzuweisungen, die innerhalb von (Start-)Tags notiert werden. Die Notation ist dabei diejenige, die Sie in Zusammenhang mit XHTML bereits kennen gelernt haben.

Eine Reihe von Regeln gelten für alle XML-Dokumente:

- Standalone-Tags werden – wie bereits aus XHTML bekannt – in XML-Sprachen stets durch einen Schrägstrich / vor der schließenden spitzen Klammer markiert, wie in `<element />`.
- Wertzuweisungen an Attribute werden grundsätzlich in einfachen oder doppelten Anführungszeichen notiert. Typografische Anführungszeichen sind dabei nicht erlaubt.
- Groß- und Kleinschreibung wird bei Element- und Attributnamen unterschieden. Im gleichen Dokument können also beispielsweise `<Name>...</Name>` und `<name>...</name>` vorkommen – es handelt sich jedoch um unterschiedliche Elemente.
- Element- und Attributnamen dürfen aus Buchstaben, Ziffern und den Interpunktionszeichen _ (Unterstrich), – (Bindestrich), . (Punkt) und : (Doppelpunkt) bestehen. Die beiden letzteren Zeichen sind jedoch für XML-interne Zwecke reserviert und sollten bei eigenen Namen nicht verwendet werden.
  Bei den Buchstaben sind zwar auch solche erlaubt, die nicht im ASCII-Zeichensatz vorkommen (a–z und A–Z), doch in der Praxis beschränkt man sich normalerweise auf die ASCII-Buchstaben.
  Das erste Zeichen eines Namens darf keine Ziffer sein und Elementnamen dürfen nicht mit `xml-` beginnen.

Werfen wir einen Blick in den Quelltext eines einfachen SVG-Dokuments. SVG (Scalable Vector Graphics), eine XML-basierte Sprache, beschreibt vektorgrafische Daten wie geometrische Formen, Schriftzüge und deren Eigenschaften. Im folgenden Beispiel wird ein Rechteck definiert, dessen Füllmuster aus einem linearen Farbverlauf von Blau nach Weiß besteht:

Listing 14.1: Beispiel eines SVG-Dokuments

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg version="1.1"  width="150" height="150" xmlns="http://www.w3.org/2000/svg">
   <defs>
      <linearGradient id="verlauf" x1="0%" y1="0%" x2="100%" y2="100%">
         <stop offset="0%" stop-color="blue" />
         <stop offset="100%" stop-color="white" />
      </linearGradient>
   </defs>
   <rect x="55" y="20" width="200" height="100" fill="url(#verlauf)" />
</svg>
```

Es geht an dieser Stelle nicht darum, das SVG-Format zu verstehen. Entscheidend ist das Verständnis für den grundsätzlichen Aufbau eines XML-Dokuments.

In der ersten Zeile steht stets die so genannte ***XML-Deklaration*** (`<?xml ...?>`). Anschließend kann – wie im obigen Beispiel – eine ***Dokumenttypdeklaration*** folgen (`<!DOCTYPE ...>`). Auf beide Deklarationen gehen wir in ***Abschnitt 14.2.2*** noch näher ein.

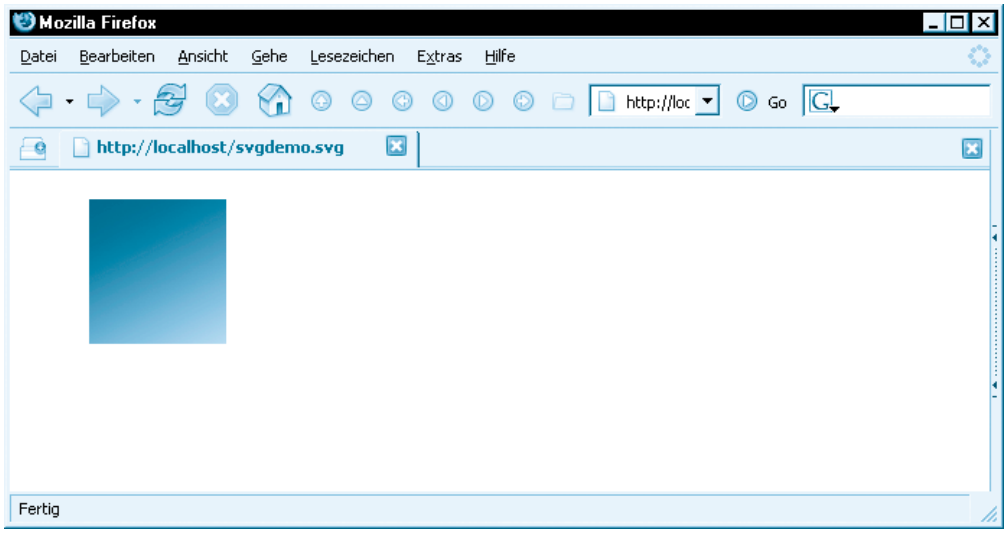


Abbildung 14.2: Beispiel-SVG-Dokument in SVG-befähigtem Webbrowser

Weiterhin besteht ein XML-Dokument aus einem so genannten ***Dokumentelement***. Dies ist das Element, zwischen dessen Start- und End-Tag der gesamte Nutzinhalt des Dokuments notiert wird. Im Fall von SVG hat das Dokumentelement den Namen `svg` und zwar deshalb, weil es in der Dokumenttypdeklaration so angegeben ist: `<!DOCTYPE svg ...>`.

Der gesamte Nutzinhalt des SVG-Dokuments ist daher also in die Tags `<svg>...</svg>` eingeschlossen.

Innerhalb der Dokumenttypdeklaration wird auch ein URI angegeben (*http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd*). Es handelt sich um den Ablageort einer so genannten ***DTD*** (***Document Type Definition***). Im Fall von SVG sind diese Definitionen öffentlich zugänglich. Sie können den genannten URI sogar im Browser aufrufen und erhalten die Dokumenttypdefinitionen als Textdokument. In der DTD wird mithilfe einer standardisierten Syntax unter anderem festgelegt, aus welchen Elementen und Attributen eine bestimmte XML-Sprache besteht, welche Elemente innerhalb von welchen anderen Elementen vorkommen dürfen, welche Elemente welche Attribute haben können und welche Regeln für Wertzuweisungen an diese Attribute gelten. Die DTD einer XML-Sprache ist somit der verbindliche Kanon dafür, was innerhalb der Sprache erlaubt ist und was nicht.

In der DTD zum SVG-Format werden unter anderem auch Elemente wie `defs`, `linearGradient` oder `rect` definiert, die im obigen SVG-Beispiel vorkommen. Gleiches gilt für Attribute wie `offset`, `stop-color`, `x`, `y` oder `width`. Die DTD legt auch fest, dass beispielsweise ein `linearGradient`-Element innerhalb eines `defs`-Elements vorkommen darf.

Auch Kommentare sind in XML erlaubt – sie werden genauso notiert, wie es aus HTML bekannt ist: `<!-- Kommentartext -->`.

14.2.2 XML-Deklarationen und Dokumenttypdeklaration

Die XML-Deklaration identifiziert nachfolgende Inhalte als XML-Daten. Die einfachste Ausprägung haben wir bereits kennen gelernt:

```
<?xml version="1.0"?>
```

Syntaktisch betrachtet gehört die XML-Deklaration zu den XML-Verarbeitungsanweisungen – Anweisungen an eine datenverarbeitende Software. Solche Anweisungen werden wie Element-Tags in spitze Klammern eingeschlossen. Das erste und letzte Zeichen innerhalb der spitzen Klammern ist jedoch ein Fragezeichen.

Die XML-Deklaration kennt drei mögliche Attribute. Diese müssen in einer bestimmten Reihenfolge notiert werden:

1. version = (Pflichtangabe: XML-Versionsnummer)
2. encoding = (optionale Angabe: verwendete Zeichenkodierung)
3. standalone = (optionale Angabe: DTD-Bezug)

XML-Versionen 1.0 und 1.1

Die XML-Deklaration muss immer das erste sein, was in einer XML-Datei steht. Sie enthält mindestens die Angabe zu der XML-Sprachversion, die nachfolgend verwendet wird. Damit ist die XML-Version gemeint, nicht etwa die Version der verwendeten XML-Sprache. Üblich ist hierbei immer noch die Angabe der XML-Version 1.0. Mittlerweile existiert jedoch auch eine als Empfehlung (Recommendation) vorliegende XML-Version 1.1. Letztere berücksichtigt unter anderem das mittlerweile weiterentwickelte Unicode-System, erlaubt im Gegensatz zur Version 1.0 das Verwenden von Steuerzeichen und bisher ausgesparten Zeichen (Dezimalwerte 1 bis 31 und 128 bis 159) in Inhalten und ist etwas freier bei der Vergabe von Namen für Elemente und Attribute.

Solange Sie keine guten Gründe haben, ist es jedoch vorerst sicherer, `version="1.0"` zu notieren, da diese Version von allen XML-Parsern akzeptiert wird. Wenn Sie Features der XML-Version 1.1 nutzen wollen, notieren Sie `version="1.1"`.

Zeichenkodierung

Neben dem `version`-Attribut enthält eine XML-Deklaration vielfach auch eine Angabe zur Zeichenkodierung. Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
```

XML ist ebenso wie HTML 4.0 oder XHTML 1.0 auf das Unicode-System ausgelegt. Für eine verarbeitende Software – und das gilt nicht nur für den XML-Parser, sondern beispielsweise auch für PHP-Scripts, die XML-Dateien direkt auslesen – ist es jedoch wichtig zu wissen, welche Zeichenkodierung beim Erstellen der Daten verwendet wurde. Nur so ist ein korrektes Auslesen möglich.

In den meisten Fällen benötigt man also Angaben, welche Zeichenkodierung beim Erstellen der XML-Daten verwendet wurde. Wenn z.B. eine XML-Datei mit einem Texteditor erstellt wird, muss beim `encoding`-Atrribut der XML-Deklaration diejenige Zeichenkodierung angegeben werden, mit der die XML-Datei abgespeichert wurde. Das kann z.B. ein herkömmlicher Zeichensatz sein, wie etwa ISO-8859-1, oder auch eine Kodierung wie UTF-8, bei der Zeichen je nach Zeichenwert durch ein oder zwei Byte repräsentiert werden. Bei guten Texteditoren können Sie auswählen, mit welcher Zeichenkodierung Sie Dateien abspeichern.

Werden XML-Daten durch ein Script erzeugt, muss sich das Script um eine bestimmte Zeichenkodierung kümmern. Scriptsprachen wie PHP bieten dazu entsprechende Kodier- und Dekodierfunktionen an – so etwa die Funktionen `utf8_encode($string)` und `utf8_decode($string)` zum Konvertieren einer Zeichenkette nach UTF-8 und zum Rück-Konvertieren einer UTF-8-konvertierten Zeichenkette in eine PHP-Zeichenkette.

XML-Dokumente ohne DTD-Bezug (wohlgeformte Dokumente)

In XML wird zwischen zwei syntaktischen »Qualitätsstufen« unterschieden: wohlgeformte und gültige Dokumente.

- Wohlgeformte Dokumente erfüllen die syntaktischen Regeln von XML, doch es fehlen Dokumenttypdefinitionen, an Hand derer das Dokument validierbar wäre.
- Gültige Dokumente sind wohlgeformte Dokumente, für die außerdem noch Dokumenttypdefinitionen existieren, an Hand derer das Dokument validierbar ist.

In der Praxis wird nicht immer mit gültigen Dokumenten gearbeitet. Webanwendunden etwa, die intern Daten in XML-Form abspeichern und diese wieder auslesen, benötigen nicht zwangsläufig den »Overhead« einer DTD.

Listing 14.2: Beispiel eines wohlgeformten XML-Dokuments

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<exercise>
   <metadata>
      <project>more</project>
      <book>1</book>
      <unit>1</unit>
      <subnumber>3</subnumber>
      <type_code>lac</type_code>
      <type>Vocabulary</type>
      <title>Vocabulary: Numbers</title>
   </metadata>
   <environment>
      <mediapath>media/</mediapath>
   </environment>
   <instructions>
      <english>Listen and click on the correct number.</english>
      <italian>Ascolta e fai clic sul numero giusto.</italian>
   </instructions>
   <items>
```

```
        <item nb="1">
           <text>one</text>
           <picfile>one.jpg</picfile>
           <sound>one.mp3</sound>
        </item>
        <item nb="2">
           <text>two</text>
           <picfile>two.jpg</picfile>
           <sound>two.mp3</sound>
        </item>
     </items>
  </exercise>
```

Das Beispieldokument ist **wohlgeformt**, weil es:

- eine korrekte XML-Deklaration enthält,
- ein Dokumentelement enthält, das alle Nutzdaten umschließt (exercise-Element),
- die syntaktischen Regeln für Element- und Attributnamen, Elementverschachtelung usw. einhält.

Es enthält jedoch keinen DTD-Bezug. Und damit ist auch nicht entscheidbar, ob verwendete Elemente wie `metadata`, `environment` oder `item` überhaupt im Rahmen des angenommenen XML-Formats erlaubt sind und ob sie beispielsweise korrekt verschachtelt sind.

XML-Dokumente mit DTD-Bezug (gültige Dokumente)

Enthält ein XML-Dokument zwischen der XML-Deklaration und dem Start-Tag des Dokumentelements noch eine Dokumenttypdeklaration, wird der Bezug zu einer DTD hergestellt. Anhand der angegebenen DTD kann das Dokument überprüft (validiert) werden. Bei der Validierung wird nicht nur geprüft, ob die allgemeinen XML-Regeln eingehalten werden, sondern auch, ob die Nutzdaten den in der DTD festgelegten Regeln entsprechen. Ein gültiges XML-Dokument ist also nicht nur im Sinne von XML syntaktisch korrekt, sondern auch im Sinne einer bestimmten XML-Sprache.

Ein DTD-Bezug ist immer dann sinnvoll, wenn unterschiedliche Personen und/oder Anwendungen Dokumente einer bestimmten XML-Sprache erstellen und bearbeiten. Bereits wenn mehrere Entwickler an unterschiedlichen Modulen einer Webanwendung arbeiten, die alle auf gemeinsame XML-Dokumente zugreifen, hilft eine DTD, Fehler zu vermeiden. Denn jede Sprache, die nur auf lockeren Konventionen beruht, neigt dazu, erweitert und verwässert zu werden. Bestes Beispiel ist HTML in seinen Frühzeiten, als Browser-Entwickler munter neue Features in HTML einführten, ohne sich um den Sprachstandard zu scheren.

In Listing 14.1 haben wir bereits eine für SVG typische Dokumenttypdeklaration kennen gelernt:

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
```

Die Deklaration besteht aus folgenden Teilen:

- `!DOCTYPE` signalisiert, dass der Inhalt zwischen diesen spitzen Klammern eine Dokumenttypdeklaration ist.
- svg nennt den Namen der Klasse von Dokumenten. Es handelt sich gleichzeitig um den Namen des Dokumentelements.
- `PUBLIC` gibt an, dass der Speicherort der DTD nicht in Form einer Pfadangabe angegeben wird, sondern in Form eines sogenannten öffentlichen Bezeichners (public identifier).
- `-//W3C//DTD SVG 1.1//EN` stellt den öffentlichen Bezeichner im Beispiel von SVG dar. Er besteht aus vier Teilen: 1. einem Plus- oder Minuszeichen, das angibt, ob es sich bei der DTD um einen ISO-Standard handelt oder nicht, 2. dem Herausgeber der DTD (hier: das W3C), 3. der DTD-Bezeichnung und ihre Versionsnummer und 4. das Sprachenkürzel für die natürliche Sprache, auf der verwendete Element- und Attributnamen dieser XML-Sprache basieren.
- `http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd` ist der URI der DTD.

Wenn die Validierungs-Software, also der XML-Parser, tatsächlich die DTD auslesen muss, um die Regeln zu ermitteln, nach denen er ein Dokument auf Gültigkeit prüfen soll, dann sollte die DTD tatsächlich unter dem angegebenen URI erreichbar sein. Parser, die ohnehin nur auf eine bestimmte Sprache ausgelegt sind, müssen die DTD dagegen meist nicht wirklich aufrufen, weil sie die Validierungsregeln bereits einprogrammiert bekommen haben. In diesem Fall genügt es, eine »Scheinadresse« anzugeben, einen sogenannten URN (Uniform Resource Name). Dazu kann man eine eigene Domain und einen sinnigen Pfadnamen erfinden. Wichtig ist nur, dass der gesamte URN weltweit eindeutig ist.

Nicht immer ist eine Internetverbindung erforderlich, um DTDs auszulesen. Innerhalb eines Projekts ist es durchaus wahrscheinlich, dass DTD und XML-Daten im gleichen lokalen Dateisystem abgelegt sind. Für solche Fälle genügt es, die DTD wie im folgenden Beispiel zu adressieren:

```
<!DOCTYPE exercise SYSTEM "/var/www/xml/exercise.dtd">
```

Dann wird SYSTEM anstelle von PUBLIC notiert. Die Angabe der viergeteilten Zeichenkette für den Herausgeber entfällt ebenfalls. Der URI zur DTD wird in Form einer lokalen Pfadangabe notiert.

Selbstverständlich sind auch Mischformen möglich, wie z. B.:

```
<!DOCTYPE exercise SYSTEM "http://www.meinserver.net/xml/exercise.dtd">
```

Listing 14.3: Beispiel eines gültigen XML-Dokuments

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE exercise SYSTEM "http://www.meinserver.net/xml/exercise.dtd">
<exercise>
   <metadata>
      <project>more</project>
```

```
      <book>1</book>
      <unit>1</unit>
      <subnumber>3</subnumber>
      <type_code>lac</type_code>
      <type>Vocabulary</type>
      <title>Vocabulary: Numbers</title>
   </metadata>
   <environment>
      <mediapath>media/</mediapath>
   </environment>
   <instructions>
      <english>Listen and click on the correct number.</english>
      <italian>Ascolta e fai clic sul numero giusto.</italian>
   </instructions>
   <items>
      <item nb="1">
         <text>one</text>
         <picfile>one.jpg</picfile>
         <sound>one.mp3</sound>
      </item>
      <item nb="2">
         <text>two</text>
         <picfile>two.jpg</picfile>
         <sound>two.mp3</sound>
     </item>
  </items>
</exercise>
```

Das Listing enthält das gleiche Beispiel wie Listing 14.2, jedoch mit Dokumenttypdeklaration.

Sonderfall: interne DTD

In manchen Fällen gibt es nur ein einziges XML-Dokument einer bestimmten XML-Sprache, das aber validierbar sein soll. Damit für dieses eine Dokument nicht extra eine externe DTD gespeichert werden muss, ist es auch möglich, die Inhalte der DTD direkt im XML-Dokument zu notieren.

Listing 14.4: Beispiel eines XML-Dokuments mit interner DTD

```
<?xml version="1.0"?>
<!DOCTYPE config [
   <!ELEMENT config (entry)+>
   <!ELEMENT entry (param, value)>
   <!ELEMENT param (#PCDATA)>
   <!ELEMENT value (#PCDATA)>
]>
<config>
   <entry>
      <param>max_save</param>
      <value>100000</value>
   </entry>
```

```
    <entry>
      <param>max_memory</param>
      <value>16M</value>
    </entry>
</config>
```

In diesem Fall erstreckt sich die Dokumenttypdeklaration von `<!DOCTYPE` bis zu `]>`. Anstelle von Identifizierungsangaben wie `PUBLIC` oder `SYSTEM` und statt einer Angabe zum Speicherort der DTD werden eckige Klammern notiert. Dazwischen wird der Inhalt der DTD notiert.

Auf die Syntax von DTDs gehen wir in ***Abschnitt 14.6*** noch näher ein. An dieser Stelle genügt es, den Aufbau des XML-Dokuments zu verstehen. Das Dokument besteht aus der XML-Deklaration, gefolgt von einer Dokumenttypdeklaration für eine Dokumentklasse namens `config`. Die Dokumenttypdeklaration enthält eine interne DTD. Unterhalb der Dokumenttypdeklaration beginnen die Nutzdaten, eingeschlossen in das bei der Dokumentklasse genannte Dokumentelement `config`. Verwendete Elemente und ihre Anordnung entsprechen den Regeln der DTD.

14.2.3 Zeichenvorrat, PCDATA und CDTATA

XML setzt auf dem Zeichenvorrat des Standards ISO/IEC 10646 auf, der wiederum mit dem Unicode-System gleichzusetzen ist.

Solange Sie in einem XML-Dokument nur Zeichen aus dem ASCII-Zeichenvorrat verwenden, ist keine besondere Angabe zur Zeichenkodierung nötig. Wenn höherwertige Zeichen vorkommen, müssen Sie entscheiden, wie Sie diese notieren. Ähnlich wie in HTML gibt es in XML mehrere Möglichkeiten:

- Wenn das Dokument nur einzelne höherwertige Zeichen enthält, können Sie diese wie aus HTML bekannt numerisch umschreiben. Ein »ü« beispielsweise können Sie numerisch dezimal als `ü` umschreiben oder hexadezimal als `ü`
- Wenn das Dokument vorwiegend Zeichen aus einem bestimmten Zeichensatz verwendet, z. B. deutschsprachige Texte, dann können Sie in der XML-Deklaration einen entsprechenden Zeichensatz wie `ISO-8859-1` angeben, Zeichen wie z. B. Umlaute so wie gewohnt eingeben und die Datei im entsprechenden Zeichensatz abspeichern.
- Wenn das Dokument Zeichen aus verschiedenen Zeichensätzen enthält oder nicht vorhersehbar ist, welche Zeichen enthalten sind, empfiehlt sich die Angabe `UTF-8` in der XML-Deklaration. Sie können dann auch Zeichen z. B. aus kyrillischen, arabischen oder fernöstlichen Sprachen direkt eingeben. Wichtig ist jedoch, dass das Dokument auch tatsächlich in UTF-8-Kodierung abgespeichert wird.

Einige Zeichenbereiche von ISO/IEC 10646 bzw. Unicode sind in XML 1.0 allerdings ausgespart: Das betrifft die Steuerzeichen mit den Hexadezimalwerten 1 bis 1F und unter den höherwertigen Zeichen die Bereiche mit den Hexadezimalwerten D800 bis DFFF, FFFE bis FFFF sowie oberhalb von 110000.

Besonders die Begrenzung des nach oben hin offenen Zeichenbereichs ist angesichts der emsigen Weiterentwicklung des Unicode-Standards fragwürdig. In XML 1.1 hat man sich daher zu einer Lockerung entschlossen. (Tenor: »Was nicht explizit verboten ist, ist erlaubt.«)

Notationen wie `&#x110001;` sind also in XML1.1 erlaubt. Weiterhin erlaubt ist es, Steuerzeichen numerisch zu kodieren, also etwa `&#x8;` (Beep-Zeichen). Allerdings dürfen in XML 1.1 (im Gegensatz zu XML 1.0) Zeichen aus dem Bereich zwischen hexadezimal 7F und 9F nur numerisch umschrieben notiert werden, also z.B. `&#x7F;` – in XML 1.0 gab es diese Vorschrift noch nicht.

XML-eigene Zeichen

Folgende Zeichen müssen Sie im Text eines XML-Dokuments analog zu HTML durch die angegebenen Notationen umschreiben:

Zeichen	Notation	Bedeutung
<	<	`lt` steht für »less than«, also »kleiner als«.
>	>	`gt` steht für »greater than«, also »größer als«.
»	"	`quot` steht für »quotation mark«, also »Anführungszeichen«.
’	'	`apos` steht für »apostroph«.
&	&	`amp` steht für »ampersand«, also »kaufmännisches Und«.

Tabelle 14.1: Notation XML-eigener Zeichen

Interpretierte Inhalte (PCDATA)

Wenn der Inhalt des XML-Elements, also das, was zwischen Start- und End-Tag steht, Zeichendaten enthalten darf, dann werden diese Zeichen vom XML-Parser interpretiert. Gemeint ist damit vor allem, dass auch Entity-Umschreibungen wie `ü` oder `&#orgName;` erkannt und durch ihre korrekten Inhalte ersetzt werden. Der XML-Parser muss die Zeichendaten also interpretieren, um sie korrekt aufzulösen. PCDATA steht für ***parsed character data*** und bedeutet: vom Parser interpretierte Zeichendaten.

Nicht interpretierte Inhalte (CDATA)

Zeichendaten innerhalb von Elementen sind also per Voreinstellung interpretierte Zeichen (PCDATA). Mitunter ist es jedoch wünschenswert, das Interpretieren von Zeichendaten durch den Parser zu verhindern. Bestes Beispiel dafür sind Script-Bereiche, etwa für JavaScript. Innerhalb von JavaScript haben Zeichen wie < und > eine ganz eigene Bedeutung. Der JavaScript-Interpreter würde sie jedoch nicht erkennen, wenn sie in der für XML verpflichtenden PCDATA-Form `<` und `>` notiert wären. JavaScript-Fehlermeldungen wären die Folge.

Um einzelne Zeichenbereiche vom Interpretieren durch den XML-Parser auszuschließen, steht folgende Syntax zur Verfügung:

Listing 14.5: JavaScript im CDATA-Abschnitt

```
<script type="text/javascript">
<![CDATA[
   function is_weekend() {
      var now = new Date();
      var dow = now.getDay();
      if(dow < 1 || dow > 5)
         return true;
      else
         return false;
   }
]]>
</script>
```

Eingeleitet wird ein CDATA-Abschnitt durch `<![CDATA[`, beendet wird er durch die Zeichenfolge `]]>`. Alles, was dazwischen steht, wird vom XML-Parser ignoriert.

Das Verschachteln von CDATA-Abschnitten ist allerdings nicht möglich und die Zeichenfolge `]]>` darf innerhalb von CDATA-Abschnitten selbst nicht vorkommen.

14.2.4 Namensräume in XML

XML bietet ein weiteres interessantes Feature an, nämlich das Mischen von zwei oder mehreren XML-Sprachen in ein und demselben Dokument. So ist es syntaktisch beispielsweise kein Problem, mitten in einer XHTML-Datei den Code einer SVG-Grafik einzubetten. Das einzige Problem dabei ist ein praktisches, nämlich dass die meisten heutigen Browser noch nicht in der Lage sind, so etwas umzusetzen. (Amaya, der experimentelle Browser des W3-Konsortiums, beherrscht dieses Feature.)

Da XML jedoch nicht nur für die direkte Verarbeitung im Browser gedacht ist, gehört es zum XML-Basiswissen, das Konzept der Namensräume (*name spaces*) zu kennen und anwenden zu können.

1. Variante: Dateninseln

Betrachten wir ein XHTML-Dokument, in dem eine SVG-Grafik eingebettet ist.

Listing 14.6: XHTML mit SVG als Dateninsel

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1" />
<title>XHTML mit SVG</title>
</head>
<body>
<h1>XHTML mit SVG</h1>
```

```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg">
<defs>
<linearGradient id="verlauf" x1="0%" y1="0%" x2="100%" y2="100%">
<stop offset="0%" stop-color="blue" />
<stop offset="100%" stop-color="white" />
</linearGradient>
</defs>
<rect x="55" y="20" width="200" height="100" fill="url(#verlauf)" />
</svg>
</body>
</html>
```

Es gibt ein »äußeres« XML-Dokument, nämlich das XHTML-Dokument. Unterhalb der h1-Überschrift ist ein »inneres« Dokument, nämlich das SVG-Dokument, eingebettet. Damit sich die beiden Dokumenttypen auseinander halten lassen, stellt XML das vordefinierte Attribut `xmlns=` (die Abkürzung steht für XML name space) zur Verfügung.

Das `xmlns`-Attribut wird jeweils im **Dokumentelement** notiert – im obigen Beispiel also im einleitenden <html>-Tag und im einleitenden <svg>-Tag. Als Wert wird ihm ein URI zugewiesen – genauer: ein URN. Also keine tatsächlich existierende Datenressource, sondern nur ein weltweit eindeutiger Name, gebildet nach dem Schema eines URI. In der Regel wird eine HTTP-Adresse verwendet, wobei der Server des »Sprachenherausgebers« zusammen mit einer frei erfindbaren Pfadangabe angegeben wird.

Die Gültigkeit eines Namensraums erstreckt sich über alle inneren Elemente eines Dokumentelements, mit Ausnahme der Elemente, die durch einen anderen Namensraum eingeschlossen werden.

2. Variante: direktes Mischen

In Fällen von »Einbettung« eines kompletten Dokuments in ein anderes ist die Variante der Dateninseln vollkommen ausreichend. Es gibt jedoch auch Fälle, in denen Elemente von verschiedenen Dokumenten stärker gemischt vorkommen. Für solche Fälle ist die zweite Variante gedacht. Hierbei wird jedem Element ein namentliches ***Namensraum-Präfix*** vorangestellt, das den Bezug zu einem bestimmten Dokumenttyp herstellt.

Angenommen, es gibt ein XML-Dokument mit Produkten und eines mit Kundenadressen. Ein typischer Ausschnitt des XML-Dokuments mit den Produkten lautet:

```
<product id="p4832" />
  <name>SkyChaser</name>
  <date>2006-03-24</date>
  <quantities>12</quantities>
</product>
```

Ein typischer Ausschnitt des XML-Dokuments mit den Kundendaten lautet:

```
<customer id="c5683" />
  <name>Johann Friedhof</name>
  <date>2006-03-24</date>
  <delivery-address>Kirchhof 6, 99912 Totenstein</delivery-address >
</customer>
```

Bestellvorgänge sollen in einem eigenen XML-Dokument festgehalten und die beiden XML-Sprachen für Produkte und Kunden gemischt werden. Wenn nun der Beispielkunde c5683 das Produkt p4832 bestellt, so könnte das XML-Protokoll der Bestellung wie folgt aussehen:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<order xmlns:product="http://meinserver.net/urns/product" xmlns:customer="http:/
/meinserver.net/urns/customer">
<product:product product:id="p4832" />
  <product:name>SkyChaser</product:name>
  <product:date>2006-03-24</product:date>
  <product:quantities>12</product:quantities>
</product:product>
<customer:customer customer:id="c5683" />
  <customer:name>Johann Friedhof</customer:name>
  <customer:date>2006-03-24</customer:date>
  <customer:delivery-address>Kirchhof 6, 99912 Totenstein</customer:delivery-
address >
</customer:customer>
</order>
```

In diesem Beispiel ist `order` das Dokumentelement eines übergeordneten XML-Dokuments. Das Dokumentelement `order` erhält zwei `xmlns`-Attribute. In beiden Fällen folgt jedoch hinter `xmlns` ein Doppelpunkt und dahinter die Angabe eines ***Namensraumnamens***. Im Beispiel werden die Namen `product` und `customer` vergeben. Dieser Name ist frei wählbar. Erst dann wird dem Attribut wie schon zuvor beschrieben ein URN als Wert zugewiesen.

Elemente können nun nach dem Schema `Namenraumname:Elementname` notiert werden. Im Beispiel bedeutet `<product:product ... />`, dass das Element `product`, markiert durch den Namensteil hinter dem Doppelpunkt, dem Namensraum `product`, markiert durch den Namensteil vor dem Doppelpunkt, zugeordnet wird. Der Namensraumname `product` wurde im einleitenden `order`-Tag mit `xmlns:product` deklariert.

Wenn Elemente keine Namensraumzuordnung erhalten, gehören sie zum Default-Namensraum. Im obigen Beispiel ist dies der Namensraum des übergeordneten Dokuments, dessen Dokumentelement `order` ist.

Wenn ein Element mit Namensraumnamen notiert wird, so muss dies bei Elementen, die aus Start- und End-Tag bestehen, sowohl beim Start-Tag als auch beim zugehörigen End-Tag geschehen, also z. B. `<product:name>SkyChaser</product:name>`.

Die Syntax des vorangestellten und mit Doppelpunkt vom eigentlichen Namen getrennten Namensraumnamens ist nicht nur auf Elementnamen anwendbar, sondern auch auf Attributnamen. Im obigen Beispiel ist dies bei den Attributen `product:id` und `customer:id` zu erkennen.

Namensräume und gültiges XML

Vielleicht ist Ihnen aufgefallen, dass unsere Beispiele zu Namensräumen zwar wohlgeformte, aber keine gültigen XML-Dokumente sind, da kein DTD-Bezug hergestellt wird.

Dahinter steckt das Problem, dass Gültigkeit und Namensräume nicht direkt kompatibel sind. Voraussetzung für gültige XML-Dokumente ist die Angabe einer Dokumenttypdeklaration und eine DTD, gegen die der Dokumentinhalt geprüft werden kann.

Nun ist es zwar möglich, durch etwas Trickserei bereits in der DTD mit Namensraumnamen zu arbeiten. Doch innerhalb der DTD-Syntax ist das Konzept der Namensraumnamen nicht vorgesehen. Wird dort also beispielsweise so etwas wie `<!ELEMENT product:name (#PCDATA)>` notiert, so wird dies nicht in Namensraum und Elementname aufgelöst, sondern komplett als Elementname verstanden.

Abhilfe bei diesem Problem schafft nur das Alternativkonzept für DTDs, XML Schema. In ***Abschnitt 14.6.6*** gehen wir näher darauf ein.

14.3 Wichtige öffentlich standardisierte XML-Formate

Nach dem notwendigen Basiswissen können wir uns nun der XML-Praxis zuwenden. Dazu ist es am besten, sich einen Überblick zu verschaffen, wie und wo XML heute bereits erfolgreich eingesetzt wird.

Es gibt eine Reihe von öffentlich standardisierten XML-Sprachen, die in der Praxis immer häufiger Anwendung finden. Zunächst eine kleine Auswahl von Beispielen, die außerhalb des Web anzusiedeln sind:

- OpenDocumentFormat (ODF):
  XML-basierte Komplettlösung für Office-Dokumente (Text, Tabellen, Flowcharts und Präsentationen). Ist mittlerweile das Standardspeicherformat der OpenSource-Suite OpenOffice, die bereits einen geschätzten Marktanteil von 5–15% im Office-Bereich hat.
- Microsoft Office Open XML:
  Derzeit noch nicht in aller Munde – doch dieses Format soll ab MS Office 2007 das Standardspeicherformat für Word, Excel und PowerPoint werden. Das Speicherformat ist offen dokumentiert und kann lizenzfrei verwendet werden.
- DocBook:
  Dokumentformat für technische Dokumentation. Es erlaubt es, aus einer Datenquelle z.B. sowohl HTML-Ausgaben als auch PDF-Dokumente zu erzeugen. Wird vielfach zur Dokumentation größerer OpenSource-Projekte eingesetzt.
- eXtensible Business Reporting Language (XBRL):
  International etabliertes Standardaustauschformat für Geschäftsberichte wie z.B. Jahresabschlüsse.
- Geography Markup Language (GML):
  Beschreibungssprache für raumbezogene Objekte. Wird unter anderem von der Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der Bundesrepublik Deutschland (AdV) eingesetzt.

Innerhalb des Web kommen vor allem folgende XML-Sprachen immer häufiger zum Einsatz:

- **XHTML (W3C-Standard):**
  Die Redefinition von HTML in XML. Immer mehr engagierte Webdesigner setzen XHTML ein, um ihre Webinhalte von vorneherein innerhalb der XML-Welt zu platzieren.
- **MathML (W3C-Standard):**
  Auszeichnungssprache für mathematisch-technische Ausdrücke. Nachdem moderne Browser die MathML-Ausgaben genuin oder mit Hilfe verfügbarer Plug-ins anzeigen können, zeichnet sich eine Verbreitung zumindest in Fachkreisen ab.
- **Resource Description Framework (RDF) (W3C-Standard):**
  Beschreibungsmodell für Metadaten für Webinhalte, unter anderem als XML-Sprache verfügbar. Basis der Bemühungen des W3-Konsortiums, ein »semantisches Web« zu etablieren. Speichert ähnliche Daten wie Meta-Tags in HTML, ist jedoch universell anwendbar, auch auf andere Ressourcen als HTML-Seiten.
- **Scalable Vector Graphics (SVG) (W3C-Standard):**
  Auszeichnungssprache für Vektorgrafik. Der bisherige Erfolg dieser Sprache, die vieles übernehmen könnte, wofür heute zu Flash gegriffen wird, ist bescheiden. Einerseits, weil die Browser SVG nur zögerlich integrieren, andererseits aber auch, weil das W3-Konsortium SVG zu »wissenschaftlich« vermarktet und die Flasher-Fraktion nicht erreicht. SVG hat jedoch das Potenzial für eine erfolgreiche XML-Sprache.

Einige der erfolgreichsten XML-Sprachen der Webpraxis gehen jedoch nicht vom W3-Konsortium aus. Allen voran ist hierbei das RSS-Format zu nennen, das derzeit einen wahren Hype erlebt. Aber auch Webservices gewinnen immer mehr an Bedeutung. In diesem Bereich konkurrieren Modelle wie WDDX und SOAP.

In den nachfolgenden Buchabschnitten gehen wir zunächst auf RSS ein, mit dessen Hilfe die allerorten im Web sprießenden Newsfeeds und Podcasts realisierbar sind. Anschließend werden wir SOAP als Nachrichtenformat für Webservices behandeln.

14.4 RSS als Basis für Newsfeeds und Podcasts

RSS steht (mittlerweile) für *Really Simple Syndication* (»wirklich einfache Verbreitung«). Dass die Technik selbst Verbreitung fand, ist vor allem der Blogger-Szene zu verdanken. Bei einem *Weblog*, grob übersetzt als »Web-Tagebuch«, kommt es nicht so sehr auf das Seitenlayout an, sondern auf die aktuellen Inhalte. Leser von Weblogs neigen außerdem dazu, nicht nur ein einziges Blog, sondern mehrere oder zahlreiche zu lesen. Dies hat dazu geführt, dass man über RSS die Inhalte der gewünschten Blogs unter einer persönlich eingerichteten, einheitlichen Benutzeroberfläche abonnieren konnte.

Das RSS-Prinzip hat sich jedoch auch in anderen Zusammenhängen als interessant erwiesen. Zahlreiche größere Websites bieten ihren Besuchern aktuelle Nachrichten an, beispielsweise größere Verbände, Parteien oder Institutionen, und natürlich auch die Webpräsenzen großer Medienunternehmen. Die News werden manchmal auf der Start-

seite angeboten, manchmal auch in speziellen News-Bereichen. Außerdem wird häufig die Möglichkeit angeboten, die News in Form eines per Mail verschickten Newsletters zu abonnieren. In diesem Bereich setzt sich RSS zunehmend als weitere Art, die News anzubieten, durch. Man spricht von *Newsfeeds*. Besonders Zielgruppen wie Journalisten wissen die Newsfeeds zunehmend zu schätzen.

Da RSS im Grunde ein allgemeines Abonniersystem für Inhalte ist, eignet es sich aber auch für ganz andere Zwecke. So hat es sich binnen kürzester Zeit zu einem neuen Quasi-Standard für die Verbreitung von Audio- und Videodateien übers Internet entwickelt. Das ganze System nennt sich in diesem Zusammenhang **Podcasting**. Das Wort ist ein Kunstwort, gebildet aus »Pod« (von Apples iPod, einem tragbaren MP3-Player mit zugehöriger Internetsoftware namens iTunes) und »broadcasting«, also dem Ausstrahlen von Sendungen.

Um in den Genuss der Vorteile von RSS-basierten Newsfeeds oder Podcasts zu kommen, benötigen Sie als Endbenutzer eine entsprechende Client-Software. Da sowohl Newsfeeds als auch Podcasts boomende Bereiche sind, gibt es bereits zahlreiche Produkte dieser Art und immer neue kommen hinzu. Viele sind OpenSource oder Freeware, manche aber auch Shareware. Einige sind einfach Plug-ins für andere Clients, etwa für Browser oder E-Mail-Programme. Andere sind eigenständige Programme.

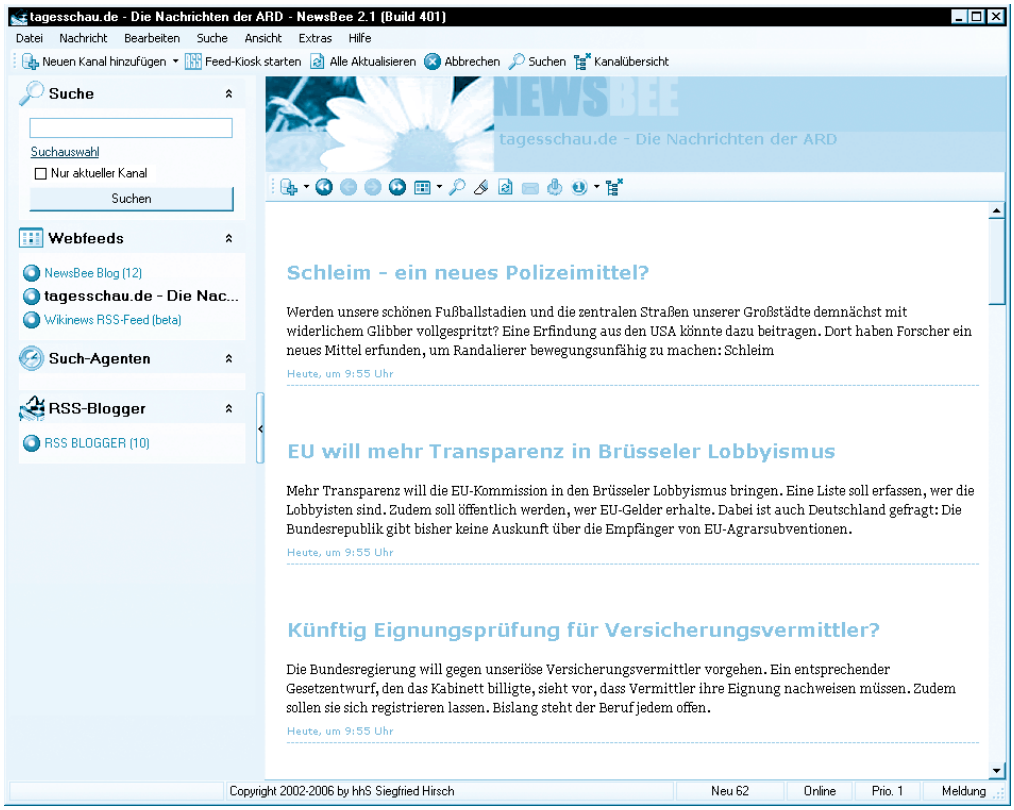


Abbildung 14.3: Typischer separater RSS-Client (hier: NewsBee)

Welcher Client am besten geeignet ist, hängt viel von persönlichen Vorlieben, Gewohnheiten und benötigten Features ab. Wenn Sie noch ein Neuling auf diesem Gebiet sind, sollten Sie auf jeden Fall mehrere Clients ausprobieren. Dadurch finden Sie am ehesten heraus, welcher Ihnen selbst am meisten zusagt. Außerdem bekommen Sie einen guten Eindruck davon, worin die Vorteile der XML-Kodierung liegen.

Links zu RSS-Newsfeed-Clients für Windows

Awasu: *http://www.awasu.com/*
(en, separater Client, Freeware- und Shareware-Versionen)
FeedBuster: *http://www.alnera.com/*
(de, separater Client, Shareware)
FeedReader: *http://www.feedreader.com/*
(en, separater Client, OpenSource)
Sage: *http://sage.mozdev.org/*
(en/de u.a., Plug-in für Mozilla und Firefox, Freeware)
NewsBee: *http://www.newsbee.de/branded/*
(de, separater Client, Freeware)

Links zu RSS-Newsfeed-Clients für Macintosh

NetNewsWire: *http://ranchero.com/netnewswire/*
(en, separater Client, Shareware)
NewsFan: *http://www.makienterprise.com/newsfan/*
(en, separater Client, Shareware)
Shrook: *http://www.utsire.com/shrook/*
(en, separater Client, Shareware)
Sage: *http://sage.mozdev.org/*
(en/de u.a., Plug-in für Mozilla und Firefox, Freeware)

Links zu RSS-Newsfeed-Clients für Linux

Liferea: *http://liferea.sourceforge.net/*
(en, separater Client, Gnome-Oberfläche, OpenSoure)
Snownews: *http://kiza.kcore.de/software/snownews/*
(de, separater Client, Textmodus/Shellebene, OpenSource)
Straw: *http://freshmeat.net/projects/straw/*
(en, separater Client, Gnome2-Oberfläche, OpenSource)
Syndigator: *http://syndigator.sourceforge.net/*
(en, separater Client, GTK-Oberfläche, OpenSource)
Sage: *http://sage.mozdev.org/*
(en/de u.a., Plug-in für Mozilla und Firefox, Freeware)

RSS-Newsfeed-Clients werden auch als ***Aggregatoren*** bezeichnet. Damit wird eine Software-Gattung bezeichnet, die Inhalte nach definierbaren Vorgaben filtert und aufbereitet. Vor allem unter Zielgruppen wie Journalisten sind solche Programme beliebt. Es spart einfach Zeit, nicht Dutzende von verschiedenen Websites aufrufen zu müssen, von denen

jede ihr eigenes Layout und ihre eigene Navigation hat. Stattdessen holt sich das Aggregator-Programm die News von all diesen Sites ähnlich wie ein Mailclient die Mails vom POP3-Server in einstellbar regelmäßigen Abständen und präsentiert sie unter einer einheitlichen Oberfläche.

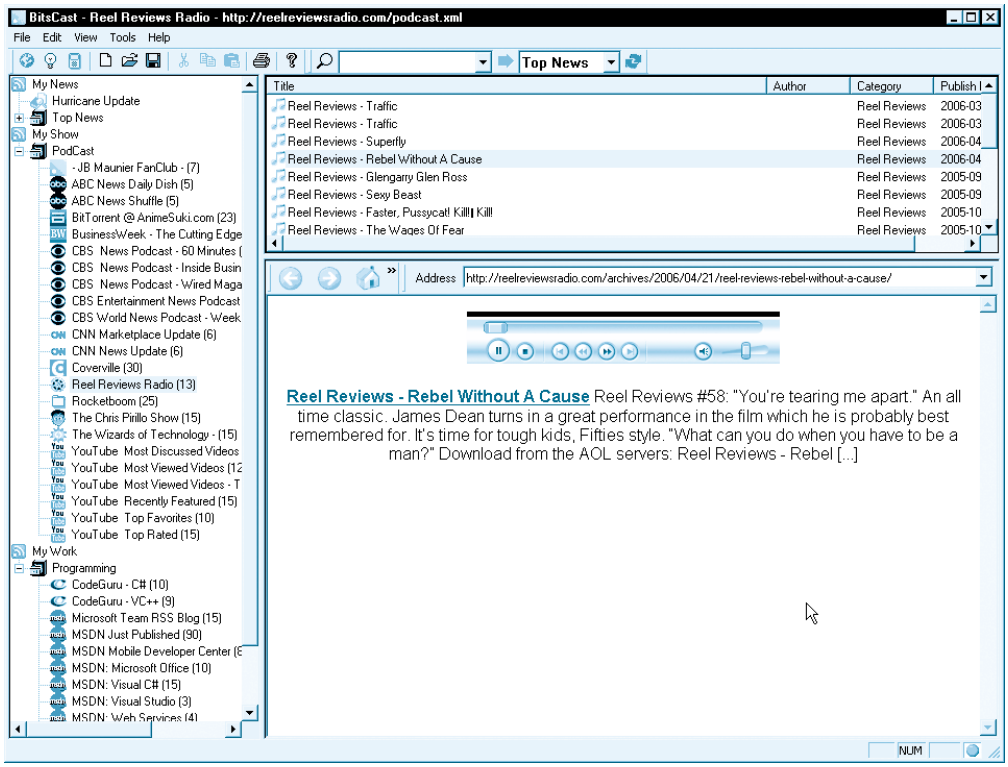


Abbildung 14.4: Typischer separater Podcast-Client (hier: Bitscast)

Podcasts lassen sich zwar auch am PC mit entsprechenden Clients anhören. Der eigentliche und häufigste Verwendungszweck ist allerdings eher der Direktdownload auf einen tragbaren MP3-Player.

Links zu Podcasting-Clients für Windows

Apple iTunes: *http://www.apple.com/itunes/*
(en, separater Client, Freeware)
BitsCast: *http://www.bitscast.com/software/*
(en, separater Client für Newsfeeds und Podcasts, Freeware)
HappyFish: *http://www.thirstycrow.net/happyfish/*
(en, separater Client für Newsfeeds und Podcasts, Freeware)
PwopCatcher: *http://www.pwopcatcher.com/*
(en, separater Client, Freeware)

RSSRadio: *http://www.dorada.co.uk/*
(en, separater Client, Shareware)
WinAmp: *http://www.winamp.com/*
(en, beliebter Mediaplayer mit integrierter Podcast-Unterstützung, Freeware)

Links zu Podcasting-Clients für Macintosh

Apple iTunes: *http://www.apple.com/itunes/*
(en, separater Client, Freeware)
NetNewsWire: *http://ranchero.com/netnewswire/*
(en, separater Client für Newsfeeds und Podcasts, Shareware)
Yamipod: *http://www.yamipod.com/*
(en/de u.a., separater Client, Freeware)

Links zu Podcasting-Clients für Linux

Amarok: *http://amarok.kde.org/*
(en, separater Client, verschiedene Oberflächen, Freeware)
Yamipod: *http://www.yamipod.com/*
(en/de u.a., separater Client, GTK2.0-Oberfläche, Freeware)

Content Syndication

Beliebte RSS-Feeds werden aber nicht nur von Endbenutzern mit entsprechenden Endbenutzer-Clients abonniert. Es gibt auch Webanwendungen, die RSS-Feeds abonnieren. Websites, die so etwas anbieten, betreiben Content Syndication, also das Wiederveröffentlichen gesammelter Feed-Inhalte. An sich betrachtet ist das eine durchaus sinnvolle Anwendung. Eine Website kann auf diese Weise News aus ganz verschiedenen Quellen, die ein Endbenutzer sonst gar nicht so leicht findet, sammeln und zur Verfügung stellen. Besonders sinnvoll ist dies, wenn etwa News für eine bestimmte Interessenszielgruppe wie Gewerkschaftler, Mieter oder Fußballfans gesammelt und unter einem Portal angeboten werden.

Leider haben jedoch viele Websites, die Content Syndication betreiben, keine wirklich informationsfilternden Interessen, sondern sind lediglich werbefinanzierte Portale, die inhaltlich planlos erfolgreiche Feeds sammeln und bei unbedarften Benutzern auf den ersten Blick den Eindruck erwecken, als seien die Inhalte redaktionell selbst erstellt.

Interessenskonflikte zwischen Anbietern und Profitierenden

Das Wesen von RSS-Feeds besteht also darin, bestimmte Inhalte aus dem Zusammenhang einer Website herauszulösen, damit sie in speziellen Clients oder Content-Syndication-Sammlungen wiederveröffentlicht werden. Die Vorteile für den Endbenutzer wurden bereits angesprochen und sind für den Erfolg der Feeds verantwortlich. Der Endbenutzer muss sich nicht mehr mit vielen unterschiedlichen Sites, deren Layout, Navigationsstruktur und Platzierung von News herumschlagen, sondern bekommt alle News unter einer einheitlichen Oberfläche aus verschiedenen Quellen automatisch präsentiert.

Für Anbieter entsprechender Inhalte ist diese Endbenutzerfreiheit allerdings gewöhnungsbedürftig. Da das corporate-identity-stiftende Layout der Originalsite fehlt, verliert der Endbenutzer leichter das Gefühl dafür, von woher oder von wem eine Information letztlich kommt. Vor allem kommerziell orientierte News-Anbieter wie Zeitungen oder Sendeanstalten haben damit ein Problem. War es für viele Anbieter klassischer Print- oder Broadcasting-Medien bereits ein schwieriger Schritt, überhaupt ein frei verfügbares Webangebot mit echten Inhalten zu verantworten, so wird ihnen mit den Feeds ein weiteres Tribut abverlangt: Die Inhalte sollen aus der identitätsstiftenden Site herausgelöst werden.

Andererseits kann sich praktisch kein namhafter Anbieter von News mehr leisten, keinen Feed anzubieten. Der Kompromiss besteht in der Regel darin, dass in den Feeds nur die Titel und die »Appetizer-Texte« der News stehen, während die Langtexte nach wie vor nur über die Original-Website abrufbar sind. Immerhin führen von den Feed-News direkte Links (Deeplinks) zu den entsprechenden Originalartikeln auf der Anbieter-Website.

Manche Feed-Hardliner betrachten diese Beschneidung von Inhalten wiederum als Zumutung, Bevormundung und Verballhornung der Feed-Idee. Für sie gehören Newsfeeds zu jener neuen Medienrealität im Web, bei der redaktionelle Inhalte ähnlich wie OpenSource-Software Allgemeingut sind. Es gibt zwar noch verpflichtende Hinweise auf die Herkunft eines Inhalts, doch der Anbieter des Inhalts verzichtet praktisch auf alle einschränkenden Rechte, die mit Wiederveröffentlichung oder Weiterverwertung zu tun haben. Der Anbieter tritt hinter den guten Zweck zurück, nämlich die Verbreitung von Information und Aufgeklärtheit. Es sind also nicht unbedingt die geistigen Urheberrechte, die durch den »OpenContent«-Gedanken zur Disposition gestellt werden, wohl aber die typischen Interessensrechte von Verlagen, Sendern und ähnlichen klassischen Verteilerunternehmen.

14.4.1 Aufbau eines RSS-Dokuments

Nachfolgendes Beispiel zeigt den Aufbau der RSS-Datei (XML-Format).

Listing 14.7: Beispiel einer RSS-Newsfeed-Datei

```
<?xml version="1.0" encoding="iso-8859-1"?>
<rss version="2.0">
<channel>
   <title>Meine News</title>
   <link>http://www.example.org/rss/news.rss</link>
   <description>
       Newsfeed von mir,
       ein Service der Ich AG
   </description>
   <language>de</language>
   <pubDate>Wed, 06 Apr 2005 17:00:00 GMT</pubDate>
   <lastBuildDate>Wed, 06 Apr 2005 17:00:00 GMT</lastBuildDate>
   <generator>Weblog Editor 2.0</generator>
   <managingEditor>editor@example.org</managingEditor>
```

```
    <webMaster>webmaster@example.org</webMaster>
    <item>
       <title>Schon gewußt?</title>
       <description>
           Blindtexte können philosophische Abgründe bergen.
       </description>
       <pubDate>Wed, 06 Apr 2005 17:00:00 GMT</pubDate>
       <link>
          http://www.example.org/rss/artikel-050406170000-01.htm
       </link>
    </item>
    <item>
       <title>Blindtext selbst gemacht</title>
       <description>
           Was Sie beim Verfassen eigener Blindtexte
           beachten müssen.
       </description>
       <pubDate>Wed, 06 Apr 2005 18:07:24 GMT</pubDate>
       <link>
          http://www.example.org/rss/artikel-050406180724-01.htm
       </link>
    </item>
  </channel>
  </rss>
```

In dieser Form handelt es sich um ein wohlgeformtes XML-Dokument, das allerdings nicht gültig ist, da keine DTD zur Validierung angegeben ist. In der Praxis wird auch meistens auf eine Dokumenttypdeklaration verzichtet. Wenn Sie ein gültiges Dokument wünschen, können Sie hinter der XML-Deklaration z. B. folgende Dokumenttypdeklaration einfügen:

```
<!DOCTYPE rss PUBLIC "-// Silmaril Consultants//DTD RSS 2.0//EN"

" http://www.silmaril.ie/software/rss2.dtd">
```

Damit wird eine öffentlich verfügbare DTD zur RSS-Version 2.0 eingebunden, gegen die ein XML-Parser validieren kann.

Wenden wir uns dem Aufbau der Datei zu. Der Nutzinhalt wird in das Dokumentelement rss eingeschlossen, markiert durch die Tags `<rss>` bzw. `</rss>`. Im Start-Tag wird durch ein version-Attribut die RSS-Versionsnummer mit angegeben – das obige Beispiel verwendet die zum Redaktionszeitpunkt dieses Buchs aktuelle Version 2.0.

Ein RSS-Feed wird auch als ***Kanal*** (channel) bezeichnet. Dieser Begriff taucht nicht nur an der Benutzeroberfläche vieler RSS-Clients auf, sondern ebenso im XML-Code, markiert durch das `channel`-Element. Ein Kanal besteht aus einigen allgemeinen Daten sowie aus den einzelnen verfügbaren News.

Allgemeine Kanaldaten

Zu den allgemeinen Daten eines RSS-Kanals gehören folgende Elemente:

- `<title>...</title>`:
  (Pflichtelement) Der Titel des Kanals.
- `<link>...</link>`:
  (Pflichtelement) URI einer Webseite, die zum Kanal gehört.
- `<description>...</description>`:
  (Pflichtelement) Kurzbeschreibung des Kanals.
- `<language>...</language>`:
  (optionales Element) Sprache der Nachrichten im Kanal. Sprachenkürzel nach RFC 1766 (z.B. `de` für Deutsch).
- `<pubDate>...</pubDate>`:
  (optionales Element) Publikationszeitpunkt des aktuellen Kanalinhalts. Angabe nach RFC 822, z.B. `Thu, 06 Apr 2006 08:42:01 GMT`.
- `<lastBuildDate>...</lastBuildDate>`:
  (optionales Element) Letzter Änderungszeitpunkt des aktuellen Kanalinhalts. Angabe nach RFC 822, z.B. `Thu, 06 Apr 2006 08:42:01 GMT`.
- `<docs>...</docs>`:
  (optionales Element) URI mit Angabe zu einer Dokumentation des verwendeten RSS-Formats. Üblicherweise `http://blogs.law.harvard.edu/tech/rss`
- `<generator>...</generator>`:
  (optionales Element) Bezeichnung oder Name des Programms oder Scripts, das den Inhalt der RSS-Datei generiert hat.
- `<managingEditor>...</managingEditor>`:
  (optionales Element) E-Mail-Adresse der redaktionell verantwortlichen Person.
- `<webMaster>...</webMaster>`:
  (optionales Element) E-Mail-Adresse der technisch verantwortlichen Person.
- `<copyright>...</copyright>`:
  (optionales Element) Rechtlicher Hinweis für die Inhalte, z.B. `Alle Rechte vorbehalten, Beispiel AG, 2006`.
- `<category>...</category>`:
  (optionales Element) Kategorienangabe für die Art der News des Kanals. Durch Schrägstrich getrennte Angaben wie `Nachrichten / Politik / Ausland` geben eine Kategorienhierarchie an, die ein RSS-Reader, falls er dies unterstützt, erkennen kann.
- `<cloud>...</cloud>`:
  (optionales Element) Möglichkeit für externe Prozesse, den Kanal zu registrieren. Beispiel:

```
<cloud domain="rpc.sys.com" port="80" path="/RPC2" registerProcedure="pingMe"
```

protocol="soap"/>
Der Zugriff ist über Webservices möglich, und zwar über die Protokolle HTTP-POST, XML-RPC oder SOAP 1.1.

- `<ttl>...</ttl>:`
  (optionales Element) Abkürzung für time to live. Anzahl der Minuten, die der Kanal im Cache-Speicher z.B. eines Proxy-Servers oder in einem File-Sharing-System zwischengespeichert werden darf, bevor er dort aktualisiert werden sollte. Sinnvoll vor allem, wenn die RSS-Datei in regelmäßigen Abständen neue Inhalte erhält.

- `<image>...</image>:`
  (optionales Element) Eine Grafik mit Logocharakter, die den Kanal identifiziert. Ein RSS-Client kann diese Grafik anzeigen, falls er die entsprechende Funktionalität unterstützt, Die Grafik kann in den üblichen Webgrafikformaten GIF, JPEG oder PNG vorliegen, muss unter einem URI erreichbar sein und sollte in ihren Ausmaßen die Breite von 144 Pixel und die Höhe von 400 Pixel nicht überschreiten. Empfohlene Default-Werte sind 88 Pixel Breite und 31 Pixel Höhe. Das image-Element enthält weitere Elemente:
  `<url>...</url>` für die Grafikadresse,
  `<title>...</title>` für den Titel/Alternativtext der Grafik,
  `<link>...</link>` für den Link zu einer angegebenen Webadresse (die Grafik ist in diesem Fall anklickbar).

- `<rating>...</rating>:`
  (optionales Element) Eine PICS-Angabe zum Kanal. Für Details zu PICS, einem vom W3-Konsortium definierten Regularium zur Auszeichnung nichtjugendfreier Inhalte, siehe ***http://www.w3.org/PICS/***.

- `<skipHours>...</skipHours>:`
  (optionales Element) Möglichkeit, dem RSS-Client mitzuteilen, in welchen Stunden des Tages er den Kanal nicht neu einlesen muss, da zu diesen Zeiten keine neuen News publiziert werden. Beispiel:
  ```
  <skipHours><hour>0</hour><hour>1</hour><hour>2</hour><hour>3</hour><hour>4</
  hour><hour>5</hour><hour>6</hour></skipHours>
  ```
  Damit wird einem RSS-Client mitgeteilt, dass er den Kanal zwischen 0 und 6 Uhr nicht zu aktualisieren braucht.

- `<skipDays>...</skipDays>:`
  (optionales Element) Möglichkeit, dem RSS-Client mitzuteilen, an welchen Wochentagen er den Kanal nicht neu einlesen muss, da zu diesen Zeiten keine neuen News publiziert werden. Beispiel:
  ```
  <skipDays><day>Saturday</day><day>Sunday</day></skipDays>
  ```
  Damit wird einem RSS-Client mitgeteilt, dass er den Kanal samstags und sonntags nicht zu aktualisieren braucht.

Einzelne Nachrichten innerhalb eines Kanals

Jede Nachricht eines RSS-Kanals wird durch ein item-Element (`<item>...</item>`) markiert. Dieses Element enthält selbst wiederum einige Pflichtelemente sowie diverse optionale Elemente, durch die Einzelheiten der jeweiligen News definiert werden:

- `<title>...</title>:`
  (Pflichtelement) Titel der jeweiligen News.
- `<link>...</link>:`
  (Pflichtelement) URI einer Webseite, auf welcher die News als normale Webseite verfügbar ist.
- `<description>...</description>:`
  (Pflichtelement) Anlesertext oder Synopsis der News.
- `<author>...</author>:`
  (optionales Element) E-Mail-Adresse und Name des Autors der jeweiligen News. Beispiel:
  `<author>s6m4@yahoo.de (Stefan Münz)</author>`
- `<category>...</ category>:`
  (optionales Element) Kategorie, zu der die jeweilige News gehört. Es gelten die gleichen Möglichkeiten wie beim category-Element innerhalb des channel-Elements.
- `<comments>...</ comments>:`
  (optionales Element) URI einer Seite mit Leserkommentaren zu einer News. Die Angabe ist dann sinnvoll, wenn es sich um ein script-basiertes News-System oder eine Weblog-Anwendung handelt, wo die Möglichkeit für Leser besteht, News-Einträge für alle Leser sichtbar zu kommentieren.
- `<enclosure />:`
  (optionales Element) URI einer Multimedia-Ressource, die den jeweiligen News-Beitrag ergänzt, z.B. ein Video. Beispiel:
  `<enclosure url="http://www.example.org/media/item934.mpg" length="869489" type="video/mpeg" />`
  Das `length`-Attribut gibt die Anzahl der Bytes der Multimedia-Ressource an und das `type`-Attribut den Mime-Type.
- `<guid>...</guid>:`
  (optionales Element) Abkürzung für ***globally unique identifier***. Der Inhalt sollte ein URN sein, also ein URI, hinter dem sich keine echte Datenressource verbirgt und der einfach als eindeutiger Bezeichner für die jeweilige News dient, z.B. `http://www.meinserver.net/news/item934`.
- `<pubDate>...</pubDate>:`
  (optionales Element) Publikationszeitpunkt der jeweiligen News. Angabe nach RFC 822, z.B. `Thu, 06 Apr 2006 08:42:01 GMT`.

- `<source>...</source>`:
  (optionales Element) Angabe eines anderen RSS-Feeds, von dem die jeweilige News bezogen wurde. Beispiel:
  ```
  <source url="http://www.example.org/other.xml">Titel des anderen Newsfeeds</
  source>
  ```

14.4.2 RSS-Integration ins eigene Webangebot

Aus Anbietersicht ist ein RSS-Feed, egal ob es sich um einen Newsfeed oder einen Podcast handelt, nichts anderes als eine XML-Datei, die den weiter oben beschriebenen Aufbau haben muss. Die XML-Datei muss als URI zugänglich sein. In der Regel wird sie unterhalb der document root des Webservers abgelegt und ist dadurch als HTTP-Adresse verfügbar.

Im RSS-Client muss dann nur der entsprechende URI eingetragen werden, um den RSS-Feed darin zu abonnieren. Damit Webseitenbesucher den URI erfahren, bietet sich ein Link dorthin an. Der Link sollte jedoch in jedem Fall kenntlich machen, dass er nicht auf eine web-übliche HTML-Ressource zeigt, sondern auf eine XML-Ressource, genauer, auf einen RSS-Feed. Für entsprechende Links haben sich bereits einige Standardsymbole etabliert:



Abbildung 14.5: Typische Symbole für RSS-Feeds

Ein eleganter Weg ist auch eine Verlinkung über das link-Element von HTML. Allerdings ist diese Art der Verlinkung derzeit nur als zusätzliche Möglichkeit zu empfehlen, da nicht alle verbreiteten Browser das link-Element hinreichend unterstützend. Beispiel:

```
<link rel="alternate" type="application/rss+xml" title="Peters Blog als RSS
Feed" href="http://www.example.org/feed/peterblog.xml">
```

Wie Sie als Anbieter den XML-Code der RSS-Newsfeed-Datei speichern, bleibt Ihnen überlassen. Er kann z. B. in einer Textdatei mit einer geeigneten Endung wie Endung ***.xml*** oder ***.rss*** gespeichert werden oder auch von einem PHP-Script mit dem Content-Typ `application/xml` als HTTP-Header an den aufrufenden Client ausgeliefert werden.

Da ein RSS-Feed eine **zusätzliche** Art darstellt, Inhalte verfügbar zu machen, existiert in der Regel bereits ein Konzept, wie laufende neue Inhalte, z. B. News, in das Webangebot integriert werden. Viele Websites, die solche Services anbieten, werden über ein Content Management System (CMS) gepflegt. Viele CMS-Produkte haben wiederum eigene Module speziell für News und bieten dabei auch die Möglichkeit an, einen RSS-Feed zu generieren. Das Gleiche gilt für Weblogs (Blogs). Letztere werden in der Regel mit webbasierten Weblog-Publishing-Systemen gepflegt, die wiederum in vielen Fällen optional das Generieren von Blog-Einträgen in RSS-Form erlauben.

Links zu Weblog-Publishing-Systemen mit RSS-Unterstützung

WordPress: *http://wordpress.org/*
(en, PHP- und MySQL-basiert, OpenSource)
Serendipity: *http://www.s9y.org/*
(en, PHP-basiert, OpenSource/BSD-Lizenz)

Links zu Content-Management-Systemen mit RSS-Unterstützung

Typo3: *http://typo3.com/*
(en, PHP- und MySQL-basiert, OpenSource)
Drupal: *http://drupal.org/*
(en, PHP- und MySQL- oder PostgreSQL-basiert, OpenSource)

Falls Sie nicht mit einem CMS oder einem Blog-System arbeiten können oder wollen und Ihre Webseiten z. B. als statische Seiten ohne jede Script-Unterstützung pflegen, bleibt Ihnen in jedem Fall die Möglichkeit, eine statische RSS-Datei von Hand über einen Editor zu bearbeiten, ebenso wie Sie die HTML-Dateien der Website pflegen. Da Sie die Inhalte jedoch gleichzeitig auf einer normalen Webseite anbieten sollten, bleibt in diesem Fall nur, die Daten an zwei Stellen zu pflegen. Für einfache Anwendungsfälle genügt das. Sobald jedoch mehrere Personen redaktionell an einem News-Service arbeiten, ist eine solche Lösung unbefriedigend.

In ***Abschnitt 14.8.6*** stellen wir eine separate, PHP-basierte Lösung vor, die einen RSS-Feed dynamisch generiert.

14.5 Webservices – Anwendungskommunikation

Angenommen, Sie möchten auf einer eigenen Website einen aktuellen Wetterbericht anbieten. Eine andere Site, die sich auf Wetterdaten spezialisiert hat, liefert die nötigen aktuellen Daten. Diese Daten können Sie an der gewünschten Stelle Ihrer Website einbauen. Oder Sie überwachen mit Hilfe eigener Webanwendungen eBay-Auktionen, ermit-

teln aktuelle Börsen- oder Währungskurse oder integrieren Suchergebnisse von Google in eigene Ausgabedaten. Es handelt sich dabei wohlgemerkt nicht um ein mehr oder weniger unerlaubtes »Grabbing« von Quelltextteilen aus fremden Webseiten, sondern um einen ganz offiziellen Datenaustausch, bei dem die eigentlichen Daten sauber strukturiert in XML übertragen werden.

Die Webanwendung auf Ihrer Website ist in all diesen Fällen der *Service-Konsument* oder der *Service-Client* und der Datenlieferant ist der *Service-Anbieter* oder *Service-Server*. Es findet also ein Datentransfer statt, bei dem der Service-Client vom Service-Server Daten anfordert. Beides sind aus Sicht eines Endbenutzers am Browser jedoch serverseitige Prozesse. Der Service-Client ist ein Prozess, der auf dem Webserver abläuft, von dem der Browser des Endbenutzers gerade eine Webseite anfordert. Der Service-Server läuft auf einem ganz anderen, vom Anbieter des Webservice betriebenen Rechner. Die Kommunikation zwischen Service-Client und Service-Server findet statt, während der Endbenutzer-Browser auf die Daten der angeforderten Webseite wartet (in diesem Fall hat der Endbenutzer die gesamte Kommunikation angestoßen, da die von ihm aufgerufene Webseite einen Webservice nutzt, um Inhalte »herbeizuholen«). Ebenso gut ist es jedoch denkbar, dass etwa ein cron-Job auf einem Webserver regelmäßig eine Webservice-Anfrage startet und die Ergebnisse speichert (in diesem Fall könnten die Ergebnisdaten z. B. sogar in statische HTML-Dateien geschrieben werden, die ein Browser vom Webserver anfordern kann).

Wie bei allen Client-Server-Kommunikationen gibt es bei Webservices ein Datenübertragungsprotokoll, über das einerseits ein Service-Konsument beim Service-Anbieter gezielt Daten anfordern kann und über das andererseits der Service-Anbieter die gewünschten Daten an den Service-Konsumenten ausliefert.

Da die übertragenen Nutzdaten eine Kommunikation zwischen zwei Software-Prozessen bedeuten, muss auch das Nutzdatenformat eindeutig sein. Dazu kommen XML-Sprachen zum Einsatz.

14.5.1 SOAP, SOAP-Messages und SOAP-Envelopes

Obwohl der Einsatz von Webservices in der heutigen Webpraxis noch recht selten ist, wird eifrig an zum Teil konkurrierenden Spezifikationen gearbeitet. Der Grund ist, dass Experten im Laufe der Zeit eine deutliche Verbreitung von Webservices erwarten. Denn das offiziell ermöglichte und gezielte Integrieren von Fremddaten in eigene Webinhalte ist, wie am Beispiel RSS-Content-Syndication sichtbar, ein Trend, der sich absehbar verstärkt. Geberfreundliche Groß-Sites wie Google folgen diesem Trend bereits und bieten entsprechende Webservices an.

So ist es auch nicht verwunderlich, dass das W3-Konsortium den zurzeit bekanntesten Standard für Webservices erarbeitet: *SOAP*. Das Akronym steht für *Simple Object Access Protocol*. Es handelt sich bei SOAP also nicht vorrangig um eine XML-basierte Beschreibungssprache, sondern um die Spezifikation des gesamten Regulariums für eine Webservice-Kommunikation.

Eine SOAP-Kommunikation besteht aus *Messages* (Nachrichten) zwischen Webservice-Client und Webservice-Server. Die Daten einer Message werden in einem XML-Dokument verpackt. SOAP selbst definiert jedoch nur XML-seitig zunächst einmal nur den so genannten *Envelope* (Briefumschlag) einer Message. Dieser besteht einfach aus einem optionalen Element namens `Header` und einem Element namens `Body`. Was dazwischen steht, sind weitere XML-Daten, die jedoch anwendungsspezifisch sind. Der allgemeine Aufbau einer SOAP-Message sieht wie folgt aus:

```
<?xml version="1.0" ?>
 <SOAP-ENV:Envelope
     xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope">
   < SOAP-ENV:Header>
     <!-- Header-Inhalte (andere XML-Daten, anderer XML-Namensraum) -->
   </SOAP-ENV:Header>
   <SOAP-ENV:Body >
     <!-- Body-Inhalte (andere XML-Daten, anderer XML-Namensraum) -->
   </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```

Den Rahmen der SOAP-Message bildet also das Dokumentelement `Envelope`, im Beispiel mit dem Namensraumpräfix `SOAP-ENV` versehen. Das Namensraumpräfix ist deshalb wichtig, da die Inhalte der beiden `envelope`-Elemente `Header` und `Body` aus anderen XML-Sprachen kommen. Der Header-Bereich ist für allgemeine Daten bzw. Metadaten gedacht, z.B. für Absenderidentifikationen, Transaktionsdaten usw. Der Body-Bereich ist für die eigentlichen Nutzdaten vorgesehen.

Versand von SOAP-Messages

Für die Kommunikation zwischen Webservice-Client und Webservice-Server reicht natürlich ein XML-Dokument alleine nicht aus. Es muss auch übertragen werden. Dazu wird meistens das HTTP-Protokoll verwendet. Der Service-Client stellt also einen HTTP-Request (vergleiche Kapitel 9) an den Service-Server und dieser liefert (in der Regel) einen HTTP-Response aus. Zwar können auch andere Protokolle als HTTP für die Datenübertragung verwendet werden, doch da es sich um einen Austausch zwischen Webanwendungen handelt, die ohnehin Webserver benötigen, bietet sich das HTTP-Protokoll an.

RPC und EDI

In den meisten Fällen löst die Anfrage des Webservice-Client beim Webservice-Server den Aufruf eines Prozesses oder einer Programmfunktion aus. Dies wird als *Remote Procedure Call (RPC)* bezeichnet. So führt etwa eine Wetterberichtsanfrage dazu, dass auf dem Webservice-Server ein Prozess angestoßen wird, der aus den aktuell gespeicherten Wetterdaten die Daten für den in der Anfrage gewünschten Ort und Vorhersagezeitraum auswählt. Die XML-Daten der SOAP-Message müssen in diesem Fall angeben, welcher Prozess oder welche Programmfunktion aufgerufen werden soll und welche Ergebnisdaten gewünscht werden.

Ein anderer professioneller Einsatzbereich von SOAP ist *EDI (Electronic Document Interchange)*. EDI ist ein internationaler Standard für geschäftliche Transaktionen. Eine Waren-

bestellung, eine Stornierung, eine Auftragsbestätigung – all das sind Dokumentarten, die sich mit Hilfe von EDI beschreiben lassen. Die XML-Daten der SOAP-Message müssen in solchen Fällen den EDI-Standard abbilden.

14.5.2 Header- und Body-Daten einer SOAP-Message

Ein Teil der SOAP-Spezifikation beschreibt das zuvor behandelte Envelope-Verfahren und den allgemeinen Ablauf eines SOAP-Nachrichtenaustauschs. Ein anderer Teil beschreibt dagegen Vorgaben für die Kodierungsart der XML-Inhalte von Header und Body. Dabei werden keine konkreten Elemente oder Attribute definiert, wie etwa bei Sprachen wie HTML. Vielmehr beschreiben die Vorgaben, wie die XML-Inhalte von SOAP-Messages strukturiert sein müssen, damit sie den »Bedürfnissen« geläufiger Programmiersprachen entgegenkommen. Denn SOAP-Messages müssen letztlich von Prozessen erzeugt und verarbeitet werden, die in typischen Hochsprachen wie PHP, C(++), Java usw. geschrieben sind. Die übermittelten Daten müssen außerdem häufig in Datenbank-Management-Systemen gespeichert werden, in denen jedes Datenfeld einen definierten Typ hat, wie z.B. Zeichenkette, Integerwert, Floatzahl oder Datum. Auch für programmiersprachentypische Datentypen wie Arrays oder Structs wird beschrieben, wie deren XML-Repräsentation allgemein aussehen sollte.

Bei den Datentypen wird dabei auf den Standard von *XML-Schema* zurückgegriffen (mehr dazu später im Buch in ***Abschnitt 14.6.6***). Wie das in einem SOAP-Message-Austausch konkret aussieht, soll das folgende Beispiel demonstrieren. Der XML-Code der Anfrage-Message vom Service-Client an den Service-Server lautet:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header>
   <W:auth xmlns:W="http://www.example.org/urns/w">
   <W:user>client0837</W:user>
   <W:pwd>fzudfhrnfd5567832hh145bnfnjrug5</W:pwd>
   </W:auth>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
   <WD:getWeather xmlns:WD="http://www.example.org/urns/wd">
      <WD:forCountryCode xsi:type="xsd:string">
      GER
      </WD:forCountryCode>
      <WD:fromDate xsi:type="xsd:date">2006-05-17</WD:fromDate>
      <WD:toDate xsi:type="xsd:date">2006-05-20</WD:toDate>
   </WD:getWeather>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Die Message besteht darin, von einem fiktiven Wetterdienst den Wetterbericht für Deutschland vom 17. bis zum 20. Mai 2006 anzufordern.

Im `Header`-Bereich des SOAP-Envelope werden dazu Anmeldedaten übertragen, die den sendenden Prozess zur Abfrage des Beispiel-Webservice berechtigen. Im `Body`-Bereich sind die Daten enthalten, mit deren Hilfe der genaue Anfragewunsch definiert wird. Das XML-Dokument enthält also eine Menge komplexen XML-Overhead für relativ wenige Nutzdaten – was durchaus ein Kritikpunkt an SOAP ist. Bei einfachen Anwendungen ist das Verhältnis von Formalitätsdaten und Nutzdaten in der Tat ineffizient. Erst bei komplexen Daten und größeren Datenmengen macht sich der XML-Aufwand wirklich bezahlt.

Erkennbar ist, dass ins Dokumentelement `Envelope` die Namensraumdefinitionen für XML-Schema und XML-Schema-Instance mit aufgenommen wurden. Damit wird es möglich, Datentypen auszuzeichnen.

Für die Nutzdaten von `Header` und `Body` wird jeweils ein Namensraum definiert, mit den Namensraumnamen `W` im Header und `WD` im Body. Die Namensräume sind wie bereits erwähnt wichtig, damit beim Parsen des XML-Dokuments klar ist, welche Daten zu welchem Dokumenttyp gehören.

Im Header ist ein Element namens `auth` notiert, dessen Inhalt aus einem `user`- und einem `pwd`-Element besteht. Es handelt sich also offensichtlich um Anmeldedaten für den Webservice. Viele Webservices erfordern zur Nutzung, dass Interessenten einen Zugang anlegen. Über die dabei erhaltenen Zugangsdaten kann sich der Webservice-Client dann beim Webservice-Server als registrierter Benutzer anmelden.

Die Daten im Body sind ähnlich strukturiert. Hier schließt ein Element namens `getWeather` die Elemente `forCountryCode`, `fromDate` und `toDate` ein. Interessant ist dabei die Angabe der Datentypen. Mit `xsi:type="xsd:string"` wird der Elementinhalt als Zeichenkette typisiert und mit `xsi:type="xsd:date"` als Datumsangabe. Möglich ist dies durch die eingebundenen Namensräume von XML-Schema und XML-Schema-Instance. Neben `string` und `date` wären auch die Angaben `integer` für Ganzzahlen, `decimal` für Fließkommawerte, `boolean` für Ja/Nein-Werte und `time` für Zeitangaben erlaubt.

Betrachten wir nun noch die fiktive Antwort, die der Wetterdienst-Service an unseren Webservice-Client zurückliefert:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
<ns1:WeatherResponse xmlns:namesp1="urn:xmethodsWeatherResponse">
   <ns1:Request
      ns1:forCountryCode="GER"
      ns1:fromDate="2006-05-17
      ns1:toDate="2006-05-17" />
   <ns1:Return>
      <ns1:TempMin xsi:type="xsd:integer">8</ns1:TempMin>
      <ns1:TempMax xsi:type="xsd:integer">21</ns1:TempMax>
      <ns1:Text xsi:type="xsd:string">cloudy, some rain showers</ns1:Text>
   </ns1:Return>
```

```
</ns1:WeatherResponse
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

In diesem Fall fehlen die Header-Daten, was durchaus erlaubt ist, wenn keine dafür geeigneten Daten übertragen werden müssen. Das `Body`-Element enthält ein Element namens `WeatherResponse`, für das auch wieder ein Namensraum definiert wird. Innerhalb des `WeatherResponse`-Elements sind ein `Request`-Element und ein `Return`-Element notiert. Das `Request`-Element bestätigt die Daten der Anfrage und das `Return`-Element enthält in einigen Unterelementen die eigentlichen Antwortdaten.

Ihr auswertendes Server-Programm (der Webservice-Client) muss die Daten dieser Antwort verstehen und verarbeiten können.

In ***Abschnitt 14.8.7*** werden wir an Hand eines Praxisbeispiels zeigen, wie Sie einen derartigen Webservice mit Hilfe von PHP nutzen können.

14.5.3 Die SOAP-Attribute mustUnderstand, role und relay

Wenn ein Service-Server ein Element oder Attribut in den XML-Daten einer SOAP-Anfrage nicht versteht, kann er dies ignorieren oder mit einer Fehlermeldung reagieren (auf Letzteres kommen wir im nächsten Abschnitt noch zu sprechen). Das Ignorieren von nicht verstandenen Daten kann jedoch fatale Folgen haben. So etwa, wenn sensible Daten ausgegeben werden, weil eine fehlerhafte Authentifizierung wegen Nichterkennung nicht zurückgewiesen wurde. Um dies zu verhindern, stellt SOAP das Attribut `mustUnderstand` (»muss verstanden werden«) zur Verfügung. In unserem Beispiel aus dem letzten Abschnitt wäre es sinnvoll, dieses Attribut auf die Authentifizierung im Header anzuwenden:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header>
   <W:auth xmlns:W="http://www.example.org/urns/w"
           SOAP-ENV:mustUnderstand="true">
   <W:user>client0837</W:user>
   <W:pwd>fzudfhrnfd5567832hh145bnfnjrug5</W:pwd>
   </W:auth>
</SOAP-ENV:Header>
<!-- usw. -->
</SOAP-ENV:Envelope>
```

Hier wird das `auth`-Element mit dem `mustUnderstand`-Attribut ausgezeichnet. Es hat den Namensraum des Envelope. Als Werte sind `true` oder `false` erlaubt. In der Regel wird `true` zugewiesen, da das Attribut überhaupt nur dann verwendet wird, wenn vom verarbeitenden Prozess erzwungen werden soll, dass er die Daten »versteht«.

Im obigen Beispiel wird also von dem Prozess, der auf dem Service-Server die SOAP-Anfrage bearbeitet, verlangt, dass er die Daten, die in das `auth`-Element eingeschlossen sind, versteht. Andernfalls soll er in jedem Fall mit einer Fehlerrückmeldung reagieren.

In manchen Fällen kann es vorkommen, dass eine SOAP-Message mehrere Stationen bzw. verarbeitende Prozesse durchläuft. Für diesen Fall kann angegeben werden, welcher Prozess welche Daten verarbeiten soll. In älteren SOAP-Versionen geschah dies mit Hilfe des `actor`-Attributs. Seit der zum Redaktionszeitpunkt aktuellen Version 1.2 des SOAP-Standards ist das Attribut `role` dafür vorgesehen. Ein Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header>
   <W:auth xmlns:W="http://www.example.org/urns/w"
           SOAP-ENV:role="http://www.w3.org/2003/05/soap-envelope/role/next">
   <W:user>client0837</W:user>
   <W:pwd>fzudfhrnfd5567832hh145bnfnjrug5</W:pwd>
   </W:auth>
</SOAP-ENV:Header>
<!-- usw. -->
</SOAP-ENV:Envelope>
```

Im Beispiel erhält das `auth`-Element ein `role`-Attribut. Wie das `mustUnderstand`-Attribut gehört das `role`-Attribut zum Namensraum des Envelope-Elements, also des Dokumentelements.

Als Wert wird dem `role`-Attribut ein URI zugewiesen. Das W3-Konsortium nennt in seiner SOAP-Spezifikation folgende URIs mit folgender Bedeutung:

- **`http://www.w3.org/2003/05/soap-envelope/role/next:`**
  Der Prozess muss die zum Element gehörigen Daten (inklusive Daten aus Kind- und Kindeskindelementen) verarbeiten.
- **`http://www.w3.org/2003/05/soap-envelope/role/none:`**
  Der Prozess muss dieses Element nicht verarbeiten.
- **`http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver:`**
  Der Prozess soll die zum Element gehörigen Daten (inklusive Daten aus Kind- und Kindeskindelementen) nur dann verarbeiten, wenn er in der Verarbeitungskette der letzte Prozess ist.

Damit die Verwendung des `role`-Attributs einen Sinn ergibt, müssen mehrere Prozesse, die an der Verarbeitung einer solchen Message beteiligt sind, die selbst bearbeiteten Elementstrukturen aus der Message löschen, bevor sie diese an den nächsten Verarbeitungsprozess weiterleiten. Auf diese Weise findet jeder beteiligte Prozess den aktuell noch zu verarbeitenden Stand vor.

Da das `role`-Attribut optional ist, können Sie es für gewöhnliche Webservices, an denen auf der Service-Server-Seite nur ein Prozess beteiligt ist, ignorieren.

Sollte ein verarbeitender Prozess in einer Kette mehrerer verarbeitender Prozesse Daten, die mit `http://www.w3.org/2003/05/soap-envelope/role/next` an ihn gerichtet sind, nicht

verarbeiten können, so muss er dem Element, das mit dem entsprechenden `role`-Attribut ausgestattet ist, ein `relay`-Attribut hinzufügen. Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header>
   <W:auth xmlns:W="http://www.example.org/urns/w"
           SOAP-ENV:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           SOAP:ENV:relay="true">
   <W:user>client0837</W:user>
   <W:pwd>fzudfhrnfd5567832hh145bnfnjrug5</W:pwd>
   </W:auth>
</SOAP-ENV:Header>
<!-- usw. -->
</SOAP-ENV:Envelope>
```

Dem `auth`-Element, das auch das `role`-Attribut enthält, fügt ein Prozess, der die Authentifizierungsdaten nicht verarbeiten kann, ein `relay`-Attribut mit dem Wert `true` hinzu. So weiß der nächste verarbeitende Prozess, dass sein Vorgänger die Daten hätte bearbeiten sollen, aber dies nicht konnte.

Auch das `relay`-Attribut gehört zum gleichen Namensraum wie das `Envelope`-Element.

Ebenso wie das `role`-Attribut ist das `relay`-Attribut nur in Zusammenhang mit mehreren in Kette geschalteten Verarbeitungsprozessen von Bedeutung und kann in allen anderen Fällen weggelassen werden.

14.5.4 SOAP-Fehlerrückmeldungen (Faults)

Wenn eine Anfrage an einen Webservice-Server unverständliche oder formal fehlerhafte Daten enthält, sollte der verarbeitende Prozess auf dem Webservice-Server dem aufrufenden Client-Prozess melden, dass ein Fehler aufgetreten ist. Solche Fehlerrückmeldungen werden in SOAP als ***Faults*** bezeichnet.

Angenommen, der aufgerufene Service-Server-Prozess erwartet bei Authentifizierungsdaten eine Passwortangabe, die aus einer 32 Zeichen langen Zeichenkette besteht. Diese Angabe interpretiert er als Hash-verschlüsselten Wert, die er durch Anwendung des gleichen Hash-Algorithmus überprüfen kann. Enthält nun eine Anfrage jedoch statt der erwarteten 32 Zeichen beispielsweise nur 20 Zeichen, so sollte der verarbeitende Prozess mit einer Fehlerrückmeldung reagieren.

Die Antwort-Message an den aufrufenden Webservice-Client könnte dann so aussehen:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
```

```
    <SOAP-ENV:Fault>
      <SOAP-ENV:Code>
        <SOAP-ENV:Value>Sender</SOAP-ENV:Value>
      </SOAP-ENV:Code>
      <SOAP-ENV:Reason xml:lang="de">
        Passwortangabe fehlerhaft
      </SOAP-ENV:Reason>
      <SOAP-ENV:Detail>
        20 Zeichen wurden empfangen.
        32 Zeichen wurden erwartet.
      </SOAP-ENV:Detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

Das Beispiel zeigt eine Fehlerrückmeldung gemäß SOAP-Standard 1.2. In SOAP 1.1 waren Fault-Meldungen noch etwas anders definiert.

Im Body-Bereich ist das Element `Fault` notiert. Es gehört zum Namensraum des Envelope. Am Vorhandensein eines `Fault`-Elements in einer Antwort-Message vom Webservice-Server kann ein Webservice-Client erkennen, dass ein Fehler aufgetreten ist.

Das `Fault`-Element muss die beiden folgenden Kindelemente enthalten:

- **`Code:`**
  Dieses Element enthält ein oder mehrere weitere Kindelemente, in denen der Fehler so spezifiziert wird, dass er für den Webservice-Client verarbeitbar ist.
- **`Reason:`**
  Dieses Element enthält Angaben, die für Menschen zum besseren Verständnis des aufgetretenen Fehlers dienen. Der Webservice-Client könnte die hier enthaltenen Textdaten beispielsweise an den aufrufenden Browser senden.

Alle diese Elemente gehören ebenfalls zum gleichen Namensraum wie das `Envelope`-Element.

Das `Code`-Element muss in jedem Fall ein `Value`-Element enthalten. Für dessen Inhalt sind folgende möglichen Werte vorgeschrieben:

- **`VersionMismatch:`**
  Wenn der Service-Client bei der Anfrage eine andere SOAP-Version angibt als diejenige, die auf der Service-Server-Seite verarbeitet wird.
- **`MustUnderstand:`**
  Wenn für Daten einer Anfrage das Attribut `mustUnderstand="true"` notiert ist und diese Daten von der Service-Server-Seite nicht verstanden wurden.
- **`Sender:`**
  Wenn der Service-Client die Nachricht formal nicht korrekt erstellt hat oder Daten der Anfrage nicht das erwartete Format haben.

- `Receiver:`
  Wenn bei der Verarbeitung der Anfrage auf Seiten des Service-Servers ein Fehler aufgetreten ist.
- `DataEncodingUnknown:`
  Wenn die Anfrage eine Zeichenkodierung angibt, die vom verarbeitenden Prozess des Service-Servers nicht unterstützt wird.

Neben dem `Value`-Element kann das `Code`-Element ein oder mehrere optionale Elemente namens `Subcode` enthalten (im obigen Beispiel nicht verwendet). Der Inhalt eines `Subcode`-Elements besteht wiederum in jedem Fall aus einem `Value`-Element, das nähere Angaben zum aufgetretenen Fehler enthält, jedoch nur für die Datenverarbeitung. Denkbar sind hier z. B. Fehlermeldungsnummern oder Fehlercodes. `Subcode`-Elemente können auch verschachtelt werden, d.h., ein `Subcode`-Element kann neben einem `Value`-Element auch selbst wieder `Subcode`-Elemente enthalten, die wieder den gleichen Aufbau haben. Das ist z. B. sinnvoll für die Darstellung von Fehlerpfaden oder Debug-Informationen.

Der Inhalt des `Reason`-Elements ist im Gegensatz zu dem des `Code`-Elements frei gestaltbar. Er kann – wie in unserem Beispiel – einfach nur aus Text bestehen, aber auch gemischten Inhalt aus Text und anderen XML-Elementen haben.

14.5.5 Die Rolle von WSDL

Die Daten, die in die SOAP-Envelope-Elemente `Header` und `Body` eingeschlossen werden, werden letztlich vom Anbieter eines Webservices vorgegeben. Art und Umfang der Daten ergeben sich daraus, welche Anfragen der Webservice verarbeitet und welche Daten er ausliefert. Im Falle von Wetterdaten könnte eine Anforderungsnachricht beispielsweise den gewünschten Ort und den gewünschten Vorhersagezeitraum angeben und die Antwortnachricht könnte genau diese Daten übermitteln.

Wenn Sie in Ihrer Webanwendung einen Webservice nutzen wollen, dann müssen Sie wissen, welche Daten der verwendete Webservice erwartet und welche er zurückliefert. Der Webservice muss also dokumentiert sein, um benutzt werden zu können. Doch natürlichsprachige Dokumentation hat den Nachteil, dass sie nur von Menschen verstanden werden kann, welche die Sprache der Dokumentation beherrschen, nicht aber von Software.

An diesem Punkt greift die ***Web Services Description Language (WSDL)*** ein. Die XML-basierte Sprache wird wie SOAP vom W3-Konsortium spezifiziert. Sie hat die Aufgabe, den Leistungsumfang und die für Anwendungen wichtigen Details eines Webservices nach einem bestimmten Schema zu dokumentieren. WSDL-Dateien werden über den Server des Webservice-Anbieters zugänglich gemacht. Ein Webservice-Client kann das WSDL-Dokument auslesen, um daraus Details für das Konstruieren einer Webservice-Anfrage zu entnehmen.

Für das allgemeine Verständnis von WSDL genügt an dieser Stelle ein Blick auf den allgemeinen Aufbau einer WSDL-Datei.

Listing 14.8: Beispiel einer WSDL-Datei

```
<?xml version="1.0" ?>
<definitions name="WeatherReport"
targetNamespace="http://www.example.org/weather.wsdl"
xmlns:tns="http://www.example.org/weather.wsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
<types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.org/xsd">
  </schema>
</types>
<message name="getWeatherRequest">
  <part name="forCountryCode" type="string" />
  <part name="fromDate" type="date" />
  <part name="toDate" type="date" />
</message>
<message name="getWeatherResponse">
  <part name="forCountryCode" type="string" />
  <part name="fromDate" type="date" />
  <part name="toDate" type="date" />
</message>
<portType name="PortForGetWeather">
  <operation name="getWeather">
     <input message="tns:getWeatherRequest"/>
     <output message="tns:getWeatherResponse"/>
  </operation>
</portType>
<binding name="getWeatherSOAPBinding"
type="tns:PortForGetWeather">
  <soap:binding style="document"
  transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="getWeather">
     <soap:operation
     soapAction="http://www.example.org/soap/Weather"/>
     <input>
       <soap:body use="literal"/>
     </input>
     <output>
       <soap:body use="literal"/>
     </output>
  </operation>
</binding>
<service name="Weather Report">
  <port name="PortForGetWeather"
  binding="tns:getWeatherSOAPBinding">
    <soap:address
    location="http://www.example.org/Weather"/>
  </port>
</service>
</definitions>
```

Wir werden nicht auf die zahlreichen Details von WSDL eingehen, da diese Sprache eine Menge versteckter Komplexität enthält. Als Webservice-Konsument kommen Sie ohnehin höchstens insofern mit WSDL in Berührung, als ein vom Webservice-Anbieter bereitgestelltes WSDL-Dokument möglicherweise die einzige Quelle ist, um Detailinformationen zu den Aufruf- und Nutzungsmöglichkeiten seines Webservices zu erhalten und beim Erstellen von SOAP-Messages zu berücksichtigen.

Das Dokumentelement von WSDL lautet `definitions`. Es enthält folgende Kindelemente, welche die Sektionen einer WSDL-Datei darstellen:

- `types:`
  Definition von vorkommenden Datentypen. In vielen Fällen genügt hierbei so wie im Beispiel die Angabe des Namensraums von XML-Schema, da XML-Schema alle üblichen Datentypen definiert.
- `message:`
  Beschreibung der Datenfelder aller Anfrage- und Antwort-Messages, die zum Webservice gehören.
- `portType:`
  Abstrakte Beschreibung aller Operationen, die im Rahmen des Webservice vorkommen.
- `binding:`
  Details zu den einzelnen Operationen.
- `service:`
  Verknüpfungsangaben zu definierten Porttypen und Bindings.

14.5.6 UDDI und WSIL

UDDI ist eine Art Telefonbuch für Webservices. *UDDI* steht für *Universal Description, Discovery and Integration*. Ähnlich wie bei Telefonbüchern werden die Daten, die über UDDI abrufbar sind, in verschiedene Arten unterteilt:

- *White pages* (»Weiße Seiten«) enthalten Angaben zu Webservice-Anbietern und deren Kontaktdaten.
- *Yellow pages* (»Gelbe Seiten«) beschreiben Webservices in Form eines Branchenverzeichnisses und enthalten allgemeine Angaben zu den angebotenen Webservices.
- *Green pages* (»Grüne Seiten«) beschreiben technische und geschäftliche Details zu den einzelnen Webservices.

Um einen UDDI-Service wie auf *http://www.uddi.org/* zu nutzen, muss man sich registrieren. Die Registrierung ist kostenlos, da ein Interesse besteht, dass Anbieter und Nutzungsinteressenten von Webservices über den UDDI-Verzeichnis-Service zusammenfinden.

Um das Konzept von UDDI besser zu verstehen, muss man sich vergegenwärtigen, dass das Potenzial von Webservices derzeit vor allem in der kommerziellen Welt und speziell

für die Business-to-Business-Kommunikation gesehen wird. Dazu gehören auch serviceabhängige, teilweise recht komplexe und mehrstufige Lizenzierungsmodelle für die Nutzung von Webservices. All solche Daten sollen über UDDI ermittelbar sein.

Häufig kritisierter Nachteil von UDDI ist die Tatsache, dass zentrale UDDI-Verzeichnisanbieter eine unkontrollierbare »Verbreitungshoheit« über Webservices haben. Gegen dieses Konzept tritt WSIL an. *WSIL* steht für *Web Service Inspection Language* und ist eine vergleichsweise einfache, XML-basierte Beschreibungssprache für Webservices. Anbieter von Webservices sollten WSIL-Dateien unter einem konventionell bekannten Dateinamen (vorgeschlagen wird *inspection.wsil*) an einer leicht auffindbaren Stelle ihres Webspace speichern (z. B. in der document root des Webservers bzw. der Domain). Such-Robots, z. B. auch solche bekannter Suchmaschinen, können die Dateien dann leicht aufspüren und die ausgelesenen WSIL-Daten in ihre Datenbank übernehmen.

UDDI und WSIL müssen allerdings nicht zwangsläufig als Kontrahenten auftreten. Ein UDDI-Verzeichnis kann schließlich ebenfalls Robots aussenden, die WSIL-Dateien auslesen, um so die Verzeichnisdatenbank zu erweitern oder aktuell zu halten. Vorteile eines UDDI-Verzeichnisses sind nämlich die gezielte Beschränkung auf Webservices und die zahlreichen Beschreibungsdetails zu den verzeichneten Services.

14.5.7 Die SOAP-Alternative WDDX

An unseren Beschreibungen zu SOAP haben Sie vermutlich schon gemerkt, dass das SOAP-Modell auf sehr komplexe Webservices ausgerichtet ist, was in der Business-to-Business-Kommunikation ja durchaus vorstellbar ist. Für Webservices jedoch, die vergleichsweise einfache und gleichartige Anfragen ohne mehrstufige Verarbeitung abwickeln, ist SOAP wie ein Großraumflugzeug, das für eine Wegstrecke von drei Kilometern benutzt werden soll. So ist es nicht verwunderlich, dass auch alternative Konzepte für Webservices angeboten werden.

Die wohl bekannteste Alternative zum SOAP-Konzept ist WDDX, auch wenn manche Fachleute vermeiden, beide Konzepte als Wettbewerber zu betrachten. Der Grund ist, dass SOAP objektorientiert ist, während WDDX nur reinen Datenaustausch zwischen Webanwendungen regelt. SOAP beschreibt, vor allem in Verbindung mit WSDL, ganze Anwendungsmodule in XML-Form. Das wesentlich früher entwickelte WDDX ist dagegen nur dazu gedacht, Daten aus der Programmiersprachenumgebung einer Webanwendung in XML zu übersetzen und aus XML wieder in die Programmiersprachenumgebung einer anderen Webanwendung zurückzuübersetzen.

WDDX steht für *Web Distributed Data Exchange*. WDDX ist ein von der Firma Allaire (bekannt durch das Software-Produkt Homesite und die Programmiersprache ColdFusion) entwickeltes, XML-basiertes Datenformat für den direkten Datenaustausch zwischen Webanwendungen. Die Webanwendungen können in den dafür üblichen Programmiersprachen wie PHP, Perl, Java/JSP, ColdFusion, C/C++ usw. geschrieben sein. Anwendungen, die für einen Webservice miteinander kommunizieren, können programmiertechnisch völlig unterschiedlich sein. Über WDDX können die Anwendungen miteinander kommunizieren. So kann etwa eine Webanwendung, welche die Anzeige von Webseiten einer

Website steuert, über WDDX Daten von einem Wetterservice abrufen und dynamisch in die Einstiegsseite integrieren. Voraussetzung ist natürlich, dass die Webanwendungen von Service-Konsument und Service-Anbieter die auszutauschenden WDDX-Daten erzeugen bzw. verstehen.

Das WDDX-Modell ist insgesamt pragmatischer als das SOAP-Modell und hat einige unbestreitbare Vorteile: So ist die WDDX-Unterstützung im Gegensatz zur SOAP-Unterstützung in der am weitesten verbreiteten Webprogrammiersprache PHP bereits von Haus aus enthalten (sofern PHP mit WDDX-Funktionalität kompiliert wurde). Für SOAP muss dagegen erst ein zusätzliches PEAR-Modul installiert werden, was für viele Interessenten bereits eine Barriere darstellt. Zwar soll PHP künftig auch eine Standard-SOAP-Unterstützung erhalten, doch bis dahin bleibt die unbefriedigende Situation.

Derzeit sind Webservices generell noch nicht verbreitet genug, um zuverlässig vorhersagen zu können, ob das Rundumschlagkonzept von SOAP oder das Straightforward-Konzept von WDDX am Ende erfolgreicher sein wird. Beide Schnittstellen werden in der Praxis bereits verwendet – so bietet Google über SOAP die Möglichkeit an, Suchergebnisse in eigene Webseiten zu integrieren. Das Bundeszentralamt für Steuern bietet dagegen an, die Abfrage von Umsatzsteuer-Identifikationsnummern über WDDX in eigene Webseiten zu integrieren.

Ein WDDX-Beispiel

Bleiben wir bei dem Beispiel der Wettervorhersage. In den Erläuterungen zu SOAP haben wir in den Beispielen dem Webservice mitgeteilt, für welches Land und für welchen Vorhersagezeitraum wir die Daten wünschen. Die Anfrage vom Webservice-Client an den Webservice-Server könnte in WDDX so formuliert werden:

```
<?xml version='1.0'?>
<wddxPacket version='1.0'>
<header/>
<data>
<struct>
<var name='CountryCode'><string>GER</CountryCode></var>
<var name='fromDate'><date>2006-05-17</date></var>
<var name='toDate'><date>2006-05-20</date></var>
</struct>
</data>
</wddxPacket>
```

Damit wird dem Service-Server mitgeteilt, dass die Wetteraussichten für Deutschland vom 17. bis zum 20. Mai 2006 gewünscht werden. Natürlich muss der Service-Server diese Anfragedaten verstehen. Seine Antwort könnte, in WDDX formuliert, lauten:

```
<?xml version='1.0'?>
<wddxPacket version='1.0'>
<header/>
<data>
<struct>
<var name='CountryCode'><string>GER</CountryCode></var>
```

```
<var name='fromDate'><date>2006-05-17</date></var>
<var name='toDate'><date>2006-05-20</date></var>
<var name='TempMin'><number>8</number></var>
<var name='TempMax'><number>21</number></var>
<var name='Text'><string>cloudy, some rain showers</string></Text>
</struct>
</data>
</wddxPacket>
```

Die Antwort enthält im Beispiel nochmals die Anfragedaten zur Bestätigung und außerdem die gewünschten Nutzdaten, hier für Tiefst- und Höchsttemperatur sowie eine Kurzbeschreibung der Wetterlage.

Auf den leicht nachvollziehbaren Aufbau der WDDX-Dokumente gehen wir nicht näher ein. Zur Erzeugung des XML-Codes eines *WDDX-Packet* (so werden die WDDX-Dokumente bezeichnet, die eine Datenanfrage oder Datenantwort enthalten) stellen die meisten Programmiersprachen fertige Funktionen bereit. In PHP gibt es spezielle WDDX-Funktionen dafür und in Perl gibt es ein CPAN-Modul für WDDX. Es genügt zu erkennen, dass der WDDX-Code Datenstrukturen und Datentypen einer anfragenden oder antwortenden Anwendung in einer programmiersprachenunabhängigen XML-Form abbildet.

In ***Abschnitt 14.8.8*** stellen wir ein Praxisbeispiel für WDDX mit PHP vor.

14.6 Dokumenttypdefinitionen (DTDs)

Wenn Daten in XML gespeichert werden sollen, aber kein öffentlich bekanntes XML-Format den eigenen Anforderungen genügt, dann bietet es sich an, ein eigenes XML-Format zu entwickeln. Soll dieses XML-Format aber an verschiedenen Stellen eingesetzt und von unterschiedlichen Programmen oder Personen bearbeitet werden, so stellt sich in jedem Fall das Problem, dass das Format genau dokumentiert werden muss. Genau das leistet eine Document Type Definition (DTD). Darin wird in Form einer bestimmten Syntax exakt festgehalten, welche Elemente es in dem gewünschten XML-Format geben soll, welche Attribute, welche Elemente innerhalb welcher anderer Elemente vorkommen dürfen, welche Attribute optional sind und welche obligatorisch usw.

DTDs dienen jedoch nicht nur der Dokumentation, sondern sind – ebenso wie XML-Sprachen selbst – außerdem maschinenlesbar. Daher eignen sie sich auch zur Überprüfung (Validierung) von XML-Dokumenten gegen die in der DTD definierten Regeln. Die entsprechende Software-Gattung dafür ist, wie bereits beschrieben, die der Parser.

Der Bezug zur DTD wird, wie wir bereits gesehen haben, innerhalb der XML-Dokumente in der Dokumenttypdeklaration (`<!DOCTYPE...>`) hergestellt.

14.6.1 Aufbau von DTDs

Um zu verstehen, wie eine DTD aufgebaut ist, betrachten wir ein XML-Dokument und seine zugehörige DTD.

Listing 14.9: Beispiel eines XML-Dokuments

```
<?xml version="1.0"?>
<!DOCTYPE members SYSTEM "members.dtd">
<members>
   <member nr="19341" position="none">
      <given_name>Anna</given_name>
      <family_name>Lyse</family_name>
      <member_since>
         <yyyy>2003</yyyy><mm>10</mm><dd>18</dd>
      </member_since>
   </member>
   <member nr="11294" position="committee">
      <given_name>Reiner</given_name>
      <family_name>Wahnsinn</family_name>
      <member_since>
         <yyyy>1998</yyyy><mm>02</mm><dd>23</dd>
      </member_since>
   </member>
   <member nr="20845" position="none">
      <given_name>Mira</given_name>
      <family_name>Mare</family_name>
      <member_since>
         <yyyy>2005</yyyy><mm>06</mm><dd>01</dd>
      </member_since>
   </member>
</members>
```

Das Beispiel zeigt eine einfache Mitgliederliste, z. B. eines Vereins. Das Dokumentelement, `members` (Mitglieder), kann mehrere Elemente namens `member` (Mitglied) enthalten. Ein `member`-Element hat zwei Attribute namens `nr` und `position`. Innerhalb eines `member`-Elements kommen folgende Elemente vor: `given_name` (Vorname des Mitglieds), `family_name` (Zuname des Mitglieds) und `member_since` (Angabe, seit wann die Mitgliedschaft besteht). Die Angabe zu `member_since` teilt sich wiederum in drei Elemente namens `yyyy` (Jahr), `mm` (Monat) und `dd` (Tag) auf.

Betrachten wir nun, wie die zugehörige DTD *members.dtd* aussehen könnte:

Listing 14.10: DTD *members.dtd*

```
<!ELEMENT members (member)+>
<!ELEMENT member (given_name, family_name, member_since)>
<!ELEMENT given_name (#PCDATA)>
<!ELEMENT family_name (#PCDATA)>
<!ELEMENT member_since (yyyy, mm, dd)>
<!ELEMENT yyyy (#PCDATA)>
<!ELEMENT mm (#PCDATA)>
<!ELEMENT dd (#PCDATA)>
<!ATTLIST member
   nr ID #REQUIRED
   position CDATA #REQUIRED
>
```

Eine DTD ist wie die XML-Dokumente selbst eine Textdatei, die mit jedem Texteditor bearbeitet werden kann. Sie enthält spezielle Definitionsanweisungen. Im Beispiel ist erkennbar, dass die Elemente, also `members`, `member`, `given_name`, `family_name`, `member_since`, `yyyy`, `mm` und `dd`, mit Hilfe von `<!ELEMENT...>` definiert werden. Außerdem werden mit `<!ATTLIST...>` die beiden Attribute `nr` und `position` für das `member`-Element definiert.

Bevor wir auf die syntaktischen Details der Definitionen eingehen, noch ein paar allgemeine Hinweise zu DTDs.

Entwurf von DTDs

Bei komplexeren Datenstrukturen ist es vor dem Kodieren einer DTD wichtig, die gewünschte Datenstruktur genau zu durchdenken. Empfehlenswert ist auch, die Überlegungen niederzuschreiben oder grafisch zu visualisieren.

Folgende Überlegungen gehören dazu:

- Welche anderen Elemente sollen innerhalb des Elements vorkommen **müssen** und welche sollen darin vorkommen **dürfen**?
- Welche Daten sollen in **Elementen** stehen und welche in **Attributen**? Im obigen XML-Beispiel könnte folgender Code:

  ```
  <member_since>
     <yyyy>2005</yyyy><mm>06</mm><dd>01</dd>
  </member_since>
  ```

  ebenso gut auch so realisiert werden:

  ```
  <member_since yyyy="2005" mm="06" dd="01" />
  ```

  In der DTD müsste dies entsprechend anders definiert werden.
- Welche Attribute in welchen Elementen sollen notiert werden **müssen** und welche notiert werden **dürfen**?
- Wie sollen Elemente und Attribute genau benannt werden?
  Hier ist zu entscheiden, ob z. B. englischsprachige Wörter zur Namensgebung dienen sollen oder deutschsprachige. Englischsprachige Bezeichnernamen sind beispielsweise vorzuziehen, wenn die Daten international bearbeitet werden. Bei Lösungen für den Eigenbedarf spricht jedoch nichts dagegen, deutsche Bezeichnernamen zu wählen. Wichtig ist in jedem Fall ein einheitliches Namenskonzept.
  Dabei ist etwa zu entscheiden, ob Abkürzungen wie z. B. `fn` oder Langnamen wie `family_name` verwendet werden sollen. Kurze Elementnamen führen bei häufigem Vorkommen zur Reduzierung der XML-Dateigrößen, Klartextnamen sind dagegen leserlicher.
  Es ist zu entscheiden, wie Teile zusammengesetzter Bezeichnernamen getrennt werden sollen, also z. B. `family_name` oder `family-name` oder `familyName`.

DTDs bearbeiten

Ähnlich wie beim Bearbeiten von HTML oder XML sind Sie beim Editieren von DTDs weitgehend frei, was die Formatierung im Quelltext betrifft. Den nachfolgenden Inhalt einer DTD:

```
<!ELEMENT members (member)+>
<!ELEMENT member (given_name, family_name, member_since)>
<!ELEMENT given_name (#PCDATA)>
können Sie ebensogut wie folgt notieren:
<!ELEMENT
        members (member)+
>
<!ELEMENT
        member (given_name, family_name, member_since)
>
<!ELEMENT
        given_name (#PCDATA)
>
```

An den Stellen, an denen **Leerraum** erforderlich ist, kann beliebig viel Leerraum stehen, und zwar mit allen Whitespace-Zeichen wie Zeilenumbruch, Tabulator oder Leerzeichen.

Da DTDs nicht selten auch von anderen Entwicklern verstanden werden müssen, sind **Kommentare** zu nichttrivialen Definitionen empfehlenswert. Kommentare können Sie innerhalb von DTDs so setzen wie aus HTML gewohnt – also durch `<!-- Kommentartext -->`.

Element- und Attributnamen dürfen aus Buchstaben, Ziffern und den Interpunktionszeichen _ (Unterstrich), – (Bindestrich), . (Punkt) und : (Doppelpunkt) bestehen. Die beiden letzteren Zeichen sind jedoch für XML-interne Zwecke reserviert und sollten bei eigenen Namen nicht verwendet werden.
Bei den Buchstaben sind zwar auch solche erlaubt, die nicht im ASCII-Zeichensatz vorkommen (a–z und A–Z), doch in der Praxis beschränkt man sich normalerweise auf die ASCII-Buchstaben.
Das erste Zeichen eines Namens darf keine Ziffer sein und Elementnamen dürfen nicht mit `xml-` beginnen.

14.6.2 Elemente und Elementabhängigkeiten definieren

Ein Element kann:

- durch Start- und End-Tag (`<element>...</element>`) markiert werden, also Inhalt haben, oder durch ein Standalone-Tag (`<element />`) markiert werden und demnach keinen Inhalt haben,
- als Inhalt nur Text, aber keine weiteren Elemente erlauben,
- als Inhalt nur ein oder mehrere andere Elemente erlauben, aber keinen Text,
- als Inhalt sowohl Text als auch ein oder mehrere andere Elemente erlauben.

Außerdem kann ein Element Attribute enthalten. Doch zunächst konzentrieren wir uns auf die Definition von Elementen und ihren möglichen Inhalten, also auch auf die mögliche Verschachtelung von Elementen.

Beim Entwurf einer DTD ist es am besten, »top-down« vorzugehen, d.h. mit dem Dokumentelement zu beginnen. Auch das Dokumentelement muss in einer DTD explizit definiert werden.

Bleiben wir beim Beispiel eines Mitgliederverzeichnisses. Das Dokumentelement heißt dort `members`. Im Beispiel wird es wie folgt definiert:

```
<!ELEMENT members (member)+>
```

Jede Elementdefinition in einer DTD steht in spitzen Klammern und beginnt mit `<!ELEMENT...` (großgeschrieben). Dahinter folgt nach einem Leerraum der gewünschte Name des Elements. Er muss den Namenskonventionen entsprechen. Beachten Sie, dass Groß- und Kleinschreibung unterschieden werden. Wenn also `<!ELEMENT members ...` definiert wird, darf in einem XML-Dokument dieser Dokumentklasse nicht `<Members>...</Members>` notiert werden!

Nach einem weiteren Leerraum folgt die Angabe zum Inhalt des Elements. Diese Angabe kann je nach Element sehr unterschiedlich sein und recht komplex werden.

In unserem Beispiel lautet die Angabe zum Inhalt:

```
(member)+
```

Das bedeutet so viel wie:

Das Element `members` kann ein oder mehrere Elemente namens `member` als Kindelemente enthalten, und sonst nichts. Also auch keinen Text.

Betrachten wir nun die Definition zum Element `member`:

```
<!ELEMENT member (given_name, family_name, member_since)>
```

In diesem Fall lautet die Angabe zum Inhalt:

```
(given_name, family_name, member_since)
```

Was so viel bedeutet wie:

Das Element `member` kann und muss folgende Kindelemente enthalten, und zwar in genau dieser Reihenfolge: `given_name`, `family_name` und `member_since`. Wiederholungen dieser Elemente, andere Elemente oder Text sind als unmittelbarer Inhalt von `member` nicht erlaubt.

Und schließlich noch die Definition zum Element `given_name`:

```
<!ELEMENT given_name (#PCDATA)>
```

Die Angabe zum erlaubten Elementinhalt lautet in diesem Fall:

```
(#PCDATA)
```

Das bedeutet, dass dieses Element als Inhalt nur Text (genauer: interpretierte Zeichendaten) enthalten darf, aber keine anderen Elemente.

Nachfolgende Tabelle erläutert die verschiedenen Syntaxmöglichkeiten bei der Definition von Elementinhalten:

Ziel	Beispiel	Erläuterung
Element soll keinen Inhalt haben.	<!ELEMENT newline EMPTY>	Durch das Schlüsselwort `EMPTY` wird ein Element als leer definiert (Standalone-Element). Im XML-Dokument müsste das Beispielelement als `<newline />` notiert werden.
Element soll nur Text als Inhalt haben.	<!ELEMENT given_name (#PCDATA)>	Durch die Angabe `(#PCDATA)` wird definiert, dass das Element nur interpretierte Zeichendaten enthalten darf, aber keine anderen Elemente. Im XML-Dokument wäre z. B. `<given_name>Stefan</given_name>` erlaubt.
Element soll nur ein bestimmtes anderes Element enthalten.	<!ELEMENT product_id product_number>	Als Inhalt wird der Name des gewünschten Elements notiert. Im XML-Dokument wäre in diesem Fall erlaubt: `<product_id><product_number>123</product_number></product_id>` Innerhalb von `product_id` dürfen weder andere Elemente als `product_number` noch Text notiert werden.
Element soll nur mehrere andere Elemente in bestimmter Reihenfolge enthalten.	<!ELEMENT member (given_name, family_name, member_since)>	Als Inhalt werden die gewünschten Kindelemente notiert, und zwar in runden Klammern und untereinander durch Kommata getrennt. Die Reihenfolge der Nennung ist die erlaubte Reihenfolge im XML-Dokument. Dort ist also erlaubt: `<member><given_name>...</given_name><family_name>...</family_name><member_since>...</member_since></member>` Innerhalb von `member` dürfen jedoch weder andere Elemente noch Text notiert werden.
Element soll nur mehrere andere Elemente enthalten, jedoch in freier Reihenfolge.	<!ELEMENT member (given_name | family_name | member_since)>	Als Inhalt werden die gewünschten Kindelemente notiert, und zwar in runden Klammern und untereinander durch Senkrechtstriche | getrennt. Im XML-Dokument ist damit z. B. auch erlaubt: `<member><member_since>...</member_since><family_name>...</family_name<given_name>...</given_name></member>`

Tabelle 14.2: Möglichkeiten bei Elementdefinitionen in einer DTD

Ziel	Beispiel	Erläuterung
Ein oder mehrere Kindelemente sollen 0, 1 oder mehrere Male vorkommen dürfen.	<!ELEMENT members (member)+> <!ELEMENT members (member)*> <!ELEMENT members (member)?>	Das Element oder die Elementfolge, trennbar durch , oder |, das bzw. die in runden Klammern notiert ist, darf durch ein + dahinter 1- bis n-mal als Inhalt vorkommen, durch ein * dahinter 0-, 1- oder n-mal und durch ein ? dahinter 0- oder 1-mal.
Das Element soll gemischten Inhalt haben können, bestehend aus Text und bestimmten anderen Elementen.	<!ELEMENT text (#PCDATA | bold | italic | link | name)*>	Das Element `text` kann sowohl interpretierte Zeichendaten als auch die Kindelemente `bold`, `italic`, `link` und `name` enthalten. Das heißt, die erlaubten Kindelemente können auch mitten im Text in beliebiger Häufigkeit vorkommen. Beispiel: `<text>Der <bold>Stern</bold> bewirkt, dass Elemente <italic>beliebig oft</italic> vorkommen können, aber nicht vorkommen müssen,</text>`
Das Element soll beliebige Inhalte haben können.	<!ELEMENT text ANY>	Durch das Schlüsselwort `ANY` wird ein Element zum Behälter für beliebige andere Elemente und Text.

Tabelle 14.2: Möglichkeiten bei Elementdefinitionen in einer DTD (Fortsetzung)

Das Arsenal der Möglichkeiten ist zwar klein, doch es ist nicht immer trivial, gewünschte Angaben zum erlaubten Inhalt eines Elements exakt so zu definieren wie gewünscht. Vor allem, wenn sehr viele Elemente im Spiel sind, kann es schnell unübersichtlich werden in einer Elementdefinition. Daher besteht die Möglichkeit, Elemente zu gruppieren. Wir betrachten dazu einen Ausschnitt aus der XHTML-DTD des W3-Konsortiums:

```
<!ENTITY % heading "h1|h2|h3|h4|h5|h6">
<!ENTITY % lists "ul | ol | dl">
<!ENTITY % misc "noscript | %misc.inline;">
<!ENTITY % blocktext "pre | hr | blockquote | address">
<!ENTITY % block
     "p | %heading; | div | %lists; | %blocktext; | fieldset | table">
<!ENTITY % Block "(%block; | form | %misc;)*">
```

In diesem Ausschnitt werden Elementnamen mit Hilfe von `<!ENTITY`-Definitionen gruppiert. Auf die genaue Syntax solcher Definitionen gehen wir in ***Abschnitt 14.6.4*** noch näher ein. Im ersten Fall werden die Elemente `h1`, `h2`, `h3`, `h4`, `h5` und `h6` zu einer Kategorie namens `heading` gruppiert. Dann werden die Elemente `ul`, `ol` und `dl` zu einer Kategorie `lists` gruppiert usw. Am Ende wird eine Kategorie namens `Block` definiert, die aus allen Elementen der Kategorien `block` und `misc` besteht sowie aus dem Element `form`.

Solche Kategorien können Sie in Elementdefinitionen beim Definieren von Elementinhalten anstelle von Elementnamen verwenden. Nochmals ein Beispiel aus der XHTML-DTD:

```
<!ELEMENT body %Block;>
```

Diese schlichte Definition betrifft das body-Element von XHTML, also den Container aller im Browser-Fenster sichtbaren Inhalte. Als Elementinhalt wird einfach das Entity `Block` zugewiesen. Dahinter verbirgt sich, wie zuvor deutlich wurde, eine recht komplexe Definition.

Elementdefinitionen im Zusammenhang

Wir werden nun noch einmal eine etwas erweiterte und komplexere Version unserer DTD für ein Mitgliederverzeichnis vorstellen:

Listing 14.11: Erweiterte DTD ***members.dtd***

```
<!ELEMENT members ((member+) | org>
<!ELEMENT org (org_name | org_type | org_city)>
<!ELEMENT org_name (#PCDATA)>
<!ELEMENT org_type (#PCDATA)>
<!ELEMENT org_city (#PCDATA)>
<!ELEMENT member (member_contact? | member_name | member_info)>
<!ELEMENT member_contact (member_address | member_phone? | member_mail?)>
<!ELEMENT member_address (member_street? | member_zipcode? | member_city)>
<!ELEMENT member_phone (#PCDATA)>
<!ELEMENT member_mail (#PCDATA)>
<!ELEMENT member_street (#PCDATA)>
<!ELEMENT member_zipcode (#PCDATA)>
<!ELEMENT member_city (#PCDATA)>
<!ELEMENT member_name (member_nick_name? | member_given_name |
mamber_family_name)>
<!ELEMENT member_nick_name (#PCDATA)>
<!ELEMENT member_given_name (#PCDATA)>
<!ELEMENT member_family_name (#PCDATA)>
<!ELEMENT member_info (member_since | member_self_info?)>
<!ELEMENT member_since (yyyy, mm, dd)>
<!ELEMENT yyyy (#PCDATA)>
<!ELEMENT mm (#PCDATA)>
<!ELEMENT dd (#PCDATA)>
<!ELEMENT member_self_info (p | h)*>
<!ELEMENT p (#PCDATA | b | i | url)*>
<!ELEMENT h (#PCDATA)>
<!ELEMENT b (#PCDATA | i)*>
<!ELEMENT i (#PCDATA)>
```

Damit werden folgende Zusammenhänge definiert:

Das Dokumentelement `members` muss ein Element namens `org` sowie eines oder mehrere Elemente namens `member` enthalten (markiert durch das `+`). Die Reihenfolge der Kindelemente von `members` ist frei (markiert durch das `|`). Ein XML-Dokument gemäß dieser DTD könnte damit schon einmal folgende Struktur haben (Beispiel):

```
<?xml version="1.0"?>
<!DOCTYPE members SYSTEM "members.dtd">
<members>
   <org>
   </org>
```

```
    <member>
    </member>
    <member>
    </member>
</members>
```

Das Element `org` soll allgemeine Daten zu der Organisation speichern, deren Mitgliederverzeichnis in XML abgebildet werden soll. Die Organisationsdaten sollen in den Kindelementen `org_name`, `org_type` und `org_city` gespeichert werden. Alle drei Elemente sind verpflichtend zu notieren, die Reihenfolge ist jedoch egal. Die Definitionen für die drei Kindelemente legen fest, dass deren Inhalt nur aus Zeichendaten (`#PCDATA`) bestehen darf. Der entsprechende Ausschnitt eines korrekten XML-Dokuments gemäß dieser Definition würde lauten (Beispiel):

```
<org>
   <org_name>Single-Club Waginger See</org_name>
   <org_city>Petting</org_city>
   <org_type>GbR</org_type>
</org>
```

Neben dem `org`-Element kann das Dokumentelement `members` eine unbegrenzte Anzahl von `member`-Elementen enthalten, jedoch mindestens eins. Der Aufbau eines `member`-Elements ist um einiges komplexer als derjenige des `org`-Elements. Jedes `member`-Element kann die Kindelemente `member_contact`, `member_name` und `member_info` in freier Reihenfolge enthalten, wobei das Element `member_contact` nicht zwingend nötig ist (markiert durch das ?). Eine korrekte Umsetzung in einem XML-Dokument wäre also beispielsweise:

```
<member>
   <member_name>
   </member_name>
   <member_info>
   </member_info>
</member>
```

Das Element `member_contact` kann, wenn vorhanden, aus den Kindelementen `member_address`, `member_phone` und `member_mail` bestehen, wobei lediglich `member_address` obligatorisch ist. Das Element `member_name` kann aus `member_nick_name`, `member_given_name` und `member_family_name` bestehen, wobei `member_nick_name` kein Pflichtelement ist. Das Element `member_info` kann als Inhalt die Elemente `member_since` und `member_self_info` haben, wobei `member_self_info` nicht zwingend erforderlich ist. Die Kindelemente von `member_since` – `yyyy`, `mm` und `dd` – sind dagegen alle drei obligatorisch, und zwar in der angegebenen Reihenfolge. `member_self_info` ist für Freitext gedacht und kann aus den Kindelementen `h` (für Überschriften) oder `p` (für Textabsätze) bestehen. Das Element für Überschriften kann als Inhalt nur Zeichendaten enthalten, während das Element für Textabsätze neben Zeichendaten auch beliebig viele Elemente namens `b`, `i` und `url` enthalten kann. Das Element `b` kann außerdem noch das Element `i` enthalten, nicht aber umgekehrt. Eine mögliche Ausprägung eines `member`-Elements könnte also so aussehen:

```
<member>
   <member_name>
      <member_given_name>Sieglinde</member_given_name>
```

```
      <member_family_name>Bleibtreu</member_family_name>
    </member_name>
    <member_info>
      <member_since>
        <yyyy>2002</yyyy><mm>07</mm><dd>31</dd>
      </member_since>
      <member_self_info>
        <p>Ich bleib dir <b>immer</b> treu!</p>
      </member_self_info>
    </member_info>
  </member>
```

Wie aus diesen Definitionen ersichtlich ist, lassen sich in einer DTD mit wenig syntaktischem Aufwand sehr komplexe Elementabhängigkeiten und Inhaltsregeln erstellen.

14.6.3 Attribute und Attributzuweisungen definieren

In vielen Fällen ist zu überlegen, ob es nicht sinnvoller ist, Daten in Attributen unterzubringen statt in Kindelementen. Ein Attribut kann:

- ebenso wie ein Element verpflichtend oder optional sein,
- als Wertzuweisung eine beliebige Zeichenkette erhalten,
- als Wertzuweisung nur eine Reihe festgelegter alternativer Werte zulassen (was es bei Elementinhalten nicht gibt),
- gar keine Wertzuweisung zulassen, sondern ein Standalone-Attribut sein, das allein durch sein Vorhandensein etwas Bestimmtes signalisiert.

Darüber hinaus ermöglicht XML noch weitere definierte Formen von Wertzuweisungen an Attribute.

Im Beispiel zu ***Abschnitt 14.6.1*** haben wir bereits Attributdefinitionen kennen gelernt. Innerhalb einer DTD werden diese mit `<!ATTLIST...>` definiert. Im erweiterten Beispiel eines Mitgliederverzeichnisses haben wir in ***Abschnitt 14.6.2*** unter anderem folgende Elementunterstruktur definiert (Ausschnitt eines XML-Dokuments, das die DTD-Vorgaben erfüllt):

```
<org>
  <org_name>Single-Club Waginger See</org_name>
  <org_city>Petting</org_city>
  <org_type>GbR</org_type>
</org>
```

Der entsprechende Ausschnitt aus der DTD lautet:

```
<!ELEMENT org (org_name | org_type | org_city)>
<!ELEMENT org_name (#PCDATA)>
<!ELEMENT org_type (#PCDATA)>
<!ELEMENT org_city (#PCDATA)>
```

Da alle drei Unterelemente nur kurze Zeichenketten speichern, könnte die konzeptionelle Entscheidung beim DTD-Entwurf auch so aussehen, dass `org` als Standalone-Element definiert wird und die drei Unterelemente als Attribute dieses Elements. Der DTD-Ausschnitt könnte dann wie folgt aussehen:

```
<!ELEMENT org EMPTY>
<!ATTLIST org
   org_name CDATA #REQUIRED
   org_type CDATA #REQUIRED
   org_city CDATA #REQUIRED
>
```

Ein Ausschnitt aus einem XML-Dokument, das diese Definitionen korrekt umsetzt, könnte beispielsweise so aussehen:

```
   <org org_name=" Single-Club Waginger See " org_type="GbR" org_city="Petting"
/>
```

Attributdefinitionen zu einem Element werden also durch `<!ATTLIST...` (großgeschrieben!) eingeleitet. Dahinter folgt nach Leerraum der Elementname. Das Element muss zu diesem Zeitpunkt bereits bekannt sein, also weiter oberhalb in der DTD mit `<!ELEMENT...` definiert worden sein. Achten Sie auf exakte Übereinstimmung bei der Groß-/Kleinschreibung.

Das Schlüsselwort `ATTLIST` zeigt bereits an, dass innerhalb einer solchen Definition mehrere Attribute definiert werden können. Dabei ist es zweckmäßig, alle Attribute je in einer eigenen Zeile zu notieren. Im obigen Beispiel werden auf diese Weise die drei Attribute `org_name`, `org_type` und `org_city` für das Element `org` definiert.

Beim Attributnamen gelten wieder die bereits beschriebenen Namenskonventionen von XML. Groß- und Kleinschreibung werden auch hier unterschieden. Es wäre also durchaus möglich, zwei unterschiedliche Attribute `Org_name` und `org_name` für das gleiche Element zu definieren.

Hinter dem Attributnamen folgt, durch Leerraum getrennt, die Angabe zum Inhalt der Wertzuweisung. Abschließend folgt noch eine der beiden Angaben `#REQUIRED`, `#IMPLIED` oder `#FIXED`. Mit `#REQUIRED` definierte Attribute sind verpflichtend, müssen also im XML-Dokument notiert werden. Mit `#IMPLIED` definierte Attribute sind dagegen optional, können also im XML-Dokument wahlweise angegeben werden oder auch nicht. Die Angabe `#FIXED` ist dann sinnvoll, wenn ein Attribut wirklich nur einen festen möglichen Wert enthalten darf.

Nachfolgende Tabelle erläutert die Möglichkeiten bei Attributdefinitionen.

Ziel	Beispiel	Erläuterung
Attribut mit beliebiger Zeichenkette als Wertzuweisung	<!ATTLIST name given_name CDATA #REQUIRED family_name CDATA #REQUIRED >	Durch das Schlüsselwort `CDATA` wird die Wertzuweisung an ein Attribut allgemein als »Zeicheninhalt« definiert. Zeichen in solchen Wertzuweisungen können durchaus wie in XML generell erlaubt numerisch umschrieben werden, also etwa `€`.
Attribut mit festen alternativen Werten in der Wertzuweisung	<!ATTLIST name given_name CDATA #REQUIRED family_name CDATA #REQUIRED gender (male|female) #REQUIRED >	Durch die Angabe `(male|female)` wird definiert, dass als Wertzuweisung an das Attribut `gender` nur die Werte `male` oder `female` erlaubt sind, im XML-Dokument also z. B. `<name ... gender=»male«>`.
Attribut ohne Wertzuweisung (Standalone-Attribut)	<!ATTLIST member active CDATA #FIXED »active« >	Attribute ohne Wertzuweisung gibt es in XML nicht. Ein Standalone-Attribut erhält daher bei der Definition üblicherweise seinen eigenen Attributnamen als einzigen möglichen Wert. Die Definition folgt am besten dem Beispiel. Als Inhaltstyp wird `CDATA` festgelegt, gefolgt von der Angabe `#FIXED` und dahinter vom einzig möglichen Wertinhalt, in Anführungszeichen notiert. Im XML-Dokument kann dies dann in der Form `<member active=»active«>` notiert werden.
Attribut mit ID-Wert als Wertzuweisung, der dokumentweit einmalig sein muss	<!ATTLIST member member_id ID #REQUIRED >	Durch das Schlüsselwort `ID` wird für den Inhalt der Wertzuweisung festgelegt, dass dieser bei mehreren Elementen gleichen Namens dokumentweit nur einmal verwendet werden darf. Ferner muss der zugewiesene Wert den XML-Namenskonventionen entsprechen, darf also z. B. nicht mit einer Ziffer beginnen. Falsch wäre also z. B. `<member member_id=»123«>`, richtig dagegen `<member member_id=»m123«>`.
Attribut mit ID-Referenzwert als Wertzuweisung, um Bezüge zu anderen Elementen herzustellen	<!ATTLIST entry id ID #REQUIRED parent_id IDREF #REQUIRED >	Durch das Schlüsselwort `IDREF` wird festgelegt, dass der Inhalt der Wertzuweisung dem Wert eines anderen, mit `ID` definierten Attributs entsprechen muss. Auf diese Weise können Elementabhängigkeiten definiert werden. Im Beispiel muss ein `entry`-Element die beiden Attribute `id` und `parent_id` enthalten, wobei `id` einen dokumentweit eindeutigen Namen enthält und `parent_id` den id-Wert eines gleichnamigen anderen Elements. Auf diese Weise lässt sich eine Baumstruktur definieren.

Ziel	Beispiel	Erläuterung
Attribut mit nmtoken-Wert als Wertzuweisung (für Namen wie bei Ankern in HTML)	<!ATTLIST member db_key NMTOKEN #REQUIRED >	Durch das Schlüsselwort `NMTOKEN` wird für den Inhalt der Wertzuweisung festgelegt, dass dieser **nicht** dokumentweit eindeutig sein muss. Er muss den XML-Namenskonventionen entsprechen, darf jedoch auch mit einer Ziffer beginnen. In der Praxis wird dieser Typ gerne für Attribute verwendet, deren Inhalt eine Laufnummer, ein Datum oder Ähnliches ist.
Attribut mit Entity-Wert (z. B. für externe Datenreferenzen)	<!ENTITY % URI "CDATA"> <!ATTLIST link target_uri %URI #REQUIRED >	Eine als Entity definierte Zeichenkette kann bei der Definition der Wertzuweisung eines Attributs anstelle einer direkten Angabe verwendet werden. Im Beispiel wird eine Entity namens URI definiert, die die Zeichenkette CDATA speichert. In der Attributdefinition wird mit %URI die Entity URI referenziert. Das Attribut `target_uri` erhält in Wirklichkeit also CDTATA als Definition für den Attributwert. Entity-Definitionen dieser Art werden z. B. in der XHTML-DTD intensiv verwendet, um semantisch sinnvolle Unterscheidungen von Inhalten zu treffen.

14.6.4 Entities definieren und referenzieren

Im Prinzip genügt es zum Erstellen von DTDs, die zuvor beschriebenen Syntaxmöglichkeiten von Element- und Attributdefinitionen zu kennen. In der Praxis können Datenstrukturen jedoch schnell sehr komplex werden. Damit DTDs in solchen Fällen änderungsfreundlicher bleiben, besteht die Möglichkeit, mit Entities (»benannten Einheiten«) zu arbeiten.

Das beste Beispiel für einen sinnvollen Einsatz liefert die XHTML-DTD des W3-Konsortiums. Nachfolgend ein Ausschnitt aus dieser DTD:

```
<!ENTITY % special "br | span | bdo | object | img | map">
<!ENTITY % fontstyle "tt | i | b | big | small">
<!ENTITY % phrase "em | strong | dfn | code | q | sub | sup |
                   samp | kbd | var | cite | abbr | acronym">
<!ENTITY % inline.forms "input | select | textarea | label | button">
<!ENTITY % misc "ins | del | script | noscript">
<!ENTITY % inline "a | %special; | %fontstyle; | %phrase; | %inline.forms;">
<!ENTITY % Inline "(#PCDATA | %inline; | %misc;)*">
<!ENTITY % heading "h1|h2|h3|h4|h5|h6">
<!ENTITY % lists "ul | ol | dl">
<!ENTITY % blocktext "pre | hr | blockquote | address">
<!ENTITY % block
     "p | %heading; | div | %lists; | %blocktext; | fieldset | table">
<!ENTITY % Block "(%block; | form | %misc;)*">
```

Entity-Definitionen werden durch `<!ENTITY` eingeleitet. Dahinter folgt, getrennt durch Leerraum, ein Prozentzeichen (`%`) und dahinter, wiederum durch Leerraum getrennt, der gewünschte Name der Entity. In der ersten Definition des obigen Beispiels wird also eine Entity namens `special` definiert, in der zweiten Definition eine Entity namens `fontstyle` usw. In Anführungszeichen wird abschließend eine Zeichenkette notiert, die der Entity

zugewiesen wird. Eine Entity ist damit nichts anderes als eine Platzhaltervariable. Einmal definiert, kann die Entity im weiteren Verlauf der DTD einfach an einer gewünschten Stelle notiert werden. Intern wird dann der Entity-Name durch die zugehörige Zeichenkette ersetzt.

Betrachten wir zur Verdeutlichung, wie in der XHTML-DTD die Elemente für Überschriften definiert werden:

```
<!ELEMENT h1 %Inline;>
<!ELEMENT h2 %Inline;>
<!ELEMENT h3 %Inline;>
<!ELEMENT h4 %Inline;>
<!ELEMENT h5 %Inline;>
<!ELEMENT h6 %Inline;>
```

An der Stelle, an der normalerweise der erlaubte Inhalt für diese Elemente notiert werden müsste, wird einfach die Entity `%Inline` referenziert. Entities werden innerhalb einer DTD stets mit dem Entity-Namen, unmittelbar voranstehendem Prozentzeichen und unmittelbar anschließendem Semikolon referenziert, also `%Entityname;`.

Verfolgen wir zurück, was sich hinter `%INLINE` verbirgt, so stoßen wir zunächst auf:

```
<!ENTITY % Inline "(#PCDATA | %inline; | %misc;)*">
```

Die Elementdefinition für `h1`-Überschriften würde damit nach Ersetzung lauten:

```
<!ELEMENT h1 (#PCDATA | %inline; | %misc;)*>
```

Damit ist bereits erkennbar, dass h1-Überschriften gemischten Inhalt haben können, bestehend aus interpretierten Zeichendaten #PCDATA sowie aus weiteren Elementen. Welche das sind, wird wiederum durch Entities ausgedrückt. Löst man alle Entity-Referenzen auf, so würde die vollständige Elementdefinition am Ende so aussehen:

```
<!ELEMENT h1 (#PCDATA | a | br | span | bdo | object | img | map | tt | i | b |
big | small | em | strong | dfn | code | q | sub | sup | samp | kbd | var | cite
| abbr | acronym | input | select | textarea | label | button | ins | del |
script | noscript)*>
```

Entities für Textkonstanten

In einer DTD definierte Entities können nicht nur im weiteren Verlauf einer DTD referenziert werden, sondern auch in XML-Dokumenten, die sich auf diese DTD beziehen. Innerhalb von XML-Dokumenten werden Entities nicht wie innerhalb der DTD durch `%Entityname;` referenziert, sondern durch `&Entityname;`. Anstelle des Prozentzeichens wird also einfach ein Ampersand-Zeichen (kaufmännisches »Und«) notiert.

Aus HTML/XHTML kennen Sie solche Entities in Form »benannter Zeichen« wie `ü` oder `α`. Diese Umschreibungen existieren nur deshalb, weil sie in der DTD als Entities definiert wurden. Dazu wieder ein kleiner Ausschnitt aus der XHTML-DTD:

```
<!ENTITY nbsp   " "> <!-- no-break space = non-breaking space,
                                U+00A0 ISOnum -->
<!ENTITY iexcl  "&#161;"> <!-- inverted exclamation mark, U+00A1 ISOnum -->
```

```
<!ENTITY cent   "&#162;"> <!-- cent sign, U+00A2 ISOnum -->
<!ENTITY pound  "&#163;"> <!-- pound sign, U+00A3 ISOnum -->
<!ENTITY curren "&#164;"> <!-- currency sign, U+00A4 ISOnum -->
<!ENTITY yen    "&#165;"> <!-- yen sign = yuan sign, U+00A5 ISOnum -->
```

Hier werden Entity-Definitionen also dazu genutzt, um »griffigere« Umschreibungen anstelle numerischer Notation von Zeichen verfügbar zu machen. Wird im XHTML-Dokument also beispielsweise ` ` notiert, so wird dies von einem XML-Parser, der die DTD ausliest oder kennt, in ` ` aufgelöst, was die numerische Notation des Zeichens mit dem Unicode-Zeichenwert 160 ist, also das Zeichen für ein erzwungenes Leerzeichen.

Entities können jedoch noch ganz andere Aufgaben wahrnehmen. So können sie beispielsweise fehlerträchtige Tipparbeit sparen und zur Datenreduktion beitragen. Angenommen, in einer DTD wird folgende Entity definiert:

```
<!ENTITY % schlu "Schludenhäuser und Söhne Maschinenbau GmbH, Postfach 1234,
98765 Treustadt an der Laich">
```

Dann genügt es, in einem XML-Dokument, das sich auf diese DTD bezieht, zu notieren (Beispiel):

```
<producer>&schlu;</producer>
```

Beim Parsen des XML-Dokuments wird die Zeichenfolge `&schlu;` durch die Zeichenkette ersetzt, die bei der Entity-Definition angegeben ist.

14.6.5 Modulare DTDs

Im vorherigen Abschnitt haben wir Entities als Platzhaltervariablen für Zeichenketten kennen gelernt. Das Potenzial von Entities reicht aber noch weiter. Eine spezielle Syntax ermöglicht es, einem XML-Parser mitzuteilen, an der Stelle einer Entity-Referenz Inhalte aus einer anderen Datenquelle, also z. B. aus einer separaten Datei einzubinden. Durch dieses Feature wird es beispielsweise möglich, eine DTD auf mehrere Dateien zu verteilen. Sinnvoll ist dies, wenn eine DTD sehr umfangreich ist oder wenn sie aus deutlich unterscheidbaren Teilen besteht.

Das Konzept erlaubt es sogar, DTDs modular zu gestalten. Angenommen, innerhalb eines Unternehmens sollen interne Formulare wie Urlaubsantrag, Verbesserungsvorschlag oder Beauftragung einer anderen Abteilung in XML abgebildet und gespeichert werden. Die Formulare enthalten einerseits sehr unterschiedliche Daten, andererseits auch gemeinsame Daten, wie z. B. Identifikationsdaten zur Person, die das Formular ausfüllt, oder das aktuelle Datum. In diesem Fall ist es sinnvoll, die immer wieder benötigten Daten in ein »DTD-Modul« auszulagern und diese DTD in den Einzel-DTDs für die Formulare einzubinden. Das folgende Beispiel zeigt, wie ein solches »DTD-Modul« mit Hilfe einer Entity eingebunden werden kann:

```
<!ENTITY % general_form_data SYSTEM "general_form_data.dtd">
%general_form_data;
```

Zunächst wird eine Entity namens `general_form_data` definiert. Durch eines der beiden anschließenden Schlüsselwörter `SYSTEM` oder `PUBLIC` wird angezeigt, dass die mit der

Entity verknüpfte Zeichenkette aus einer externen Datenquelle geholt werden soll. In Anführungszeichen wird dahinter ein URI erwartet. Wenn Sie eine bestimmte, existierende Datenquelle angeben, z. B. eine existierende Datei, egal ob sich diese im eigenen Dateisystem oder auf einem anderen Server befindet, dann verwenden Sie `SYSTEM`. Nur wenn Sie die DTD eines öffentlichen XML-Formats über einen weltweit eindeutigen Bezeichnernamen (URN) referenzieren, dann verwenden Sie `PUBLIC` und geben als URI den URN an. Wenn Sie `SYSTEM` verwenden, können Sie beim Ablageort der DTD die aus HTML bekannten Referenzierungsmöglichkeiten nutzen. Liegt die referenzierte DTD im gleichen Verzeichnis wie die aktuelle DTD, kann sie wie im obigen Beispiel einfach über den Dateinamen referenziert werden. Ebenso sind aber auch relative oder absolute Pfadangaben sowie vollständige URIs wie etwa HTTP-Adressen möglich, um die einzubindende Datei zu referenzieren.

Um die referenzierte Datei tatsächlich einzubinden, muss die so definierte Entity allerdings noch angewendet werden. Das geschieht in der zweiten Zeile des obigen Beispiels. Die Entity wird einfach mit `%Entityname;` notiert, was bedeutet, dass genau an dieser Stelle der Inhalt der referenzierten Datei eingefügt wird.

Die referenzierte Datei kann natürlich anstelle von DTD-Inhalten auch etwas anderes enthalten, z. B. eine längere Zeichenkette. Die Anwendungsart »modulare DTDs« ist nur eine mögliche.

14.6.6 XML-Schema (XSD) als modernere DTD-Alternative

Bei der Syntax von DTDs fällt auf, dass diese selber zwar »irgendwie ähnlich« zu der von XML-Sprachen ist, aber doch anders. Die etwas eigenwillige Syntax der spitzen Klammern mit den durch Ausrufezeichen eingeleiteten Deklarationen verrät die historische Ableitung von XML aus SGML. Auch dort werden »Dokumentvorlagen« auf diese Weise definiert.

Die wichtigsten Schwächen des DTD-Konzepts sind aus heutiger Sicht:

- DTDs folgen selbst nicht den Regeln von XML-Sprachen, können daher von einem XML-Parser nicht als XML-Dokument verarbeitet werden und sind z. B. auch nicht selbst validierbar.
- DTDs unterstützen das Konzept der XML-Namensräume nicht (vergleiche auch ***Abschnitt 14.2.4***).
- DTDs kennen keine Feinunterscheidung von Datentypen, d.h., es gibt nur die Datentypen CDATA und PCDATA, aber keine Möglichkeit, anzugeben, dass eine solche Zeichenkette z. B. als Integerzahl, Fließkommazahl oder Datums- und Zeitangabe interpretierbar sein soll.

Nun gehören zum »XML-Komplex« bereits eine Reihe von XML-Sprachen mit XML-internen Aufgaben, wie etwa XSL als Formatiersprache. Es lag also nahe, auch zum Beschreiben von Dokumenttypen eine Sprache zu entwickeln, die selbst eine XML-Sprache ist. Das

Ergebnis ist die Sprache **XML Schema**. Die Sprache wird vom W3-Konsortium spezifiziert. Sie zielt vor allem darauf ab, die oben aufgelisteten Schwächen des DTD-Konzepts zu beseitigen.

In der Praxis wird allerdings immer noch vorzugsweise mit DTDs gearbeitet. Das liegt teilweise daran, dass die Dokumenttypdefinitionen bereits aus einer Zeit stammen, als XML Schema noch nicht ausgearbeitet war. Auch die Dokumenttypdefinitionen bekannter öffentlicher XML-Formate wie XHTML, SVG usw. liegen offiziell als DTD vor. Und noch ein Grund ist, dass zwar alle validierenden XML-Parser gegen eine DTD, aber nicht alle auch gegen XML-Schema-Definitionen validieren können.

XML Schema teilt daher ein wenig das Schicksal von SVG: eine sinnvolle, ausgereifte Sache, die sich jedoch nicht durchsetzt, weil andere, übermächtige De-facto-Standards existieren und daher auch die Software-Unterstützung nur zögerlich anläuft. Wegen der Dominanz des DTD-Konzepts in der Praxis werden wir XML Schema nur kurz an einem kleinen Beispiel vorstellen.

Listing 14.12: Beispiel eines XML-Dokuments mit Schema-Bezug

```
<?xml version="1.0"?>
<members xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="members.xsd">
   <member nr="19341" position="none">
      <given_name>Anna</given_name>
      <family_name>Lyse</family_name>
      <member_since>
         <yyyy>2003</yyyy><mm>10</mm><dd>18</dd>
      </member_since>
   </member>
   <member nr="11294" position="committee">
      <given_name>Reiner</given_name>
      <family_name>Wahnsinn</family_name>
      <member_since>
         <yyyy>1998</yyyy><mm>02</mm><dd>23</dd>
      </member_since>
   </member>
   <member nr="20845" position="none">
      <given_name>Mira</given_name>
      <family_name>Mare</family_name>
      <member_since>
         <yyyy>2005</yyyy><mm>06</mm><dd>01</dd>
      </member_since>
   </member>
</members>
```

Das Beispiel zeigt wieder das bereits bekannte XML-Dokument eines Mitgliederverzeichnisses. Es fehlt jedoch eine Dokumenttypdeklaration mit Bezug zu einer DTD. Stattdessen enthält das Dokumentelement (members) zwei Attribute. Der Grund ist, dass ein XML Schema nicht wie eine DTD über eine Dokumenttypdeklaration eingebunden wird, sondern über die Technik der Namensräume (vergleiche ***Abschnitt 14.2.4***).

Das erste Attribut im Dokumentelement ist das `xmlns`-Attribut, über das der Namensraum von XML Schema deklariert wird, genauer, der Namensraum einer XML-Schema-Instanz. Als Namensraumname wird `xsi` gewählt (`xsi` steht für XML Schema Instance). Die gesamte Angabe

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

ist in dieser Form üblich und sollte nicht verändert werden.

Das zweite Attribut bezieht sich bereits auf den zuvor deklarierten Namensraum und spricht mit `xsi:noNamespaceSchemaLocation` das XML-Schema-Attribut `noNamespaceSchemaLocation` an. Als Wert wird diesem Attribut der URI einer Datei mit den XML-Schema-Definitionen zugewiesen. Im obigen Beispiel liegt diese Datei im gleichen Verzeichnis, kann also ohne weitere Pfad- oder Protokoll- und Server-Angabe referenziert werden. Anstelle von `xsi:noNamespaceSchemaLocation` könnte auch `xsi:SchemaLocation` notiert werden. Das ist dann erforderlich, wenn dem Dokumentelement von vorneherein ein bestimmter Namensraum zugeordnet werden soll. Das einleitende `<members>`-Tag könnte in diesem Fall so aussehen:

```
<members xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
SchemaLocation="http://meinserver.net/members members.xsd">
```

Die Zuweisung an `xsi:SchemaLocation` besteht 1. aus einem URN, also einer frei wählbaren, eindeutigen Namensadresse, mit der keine tatsächliche Datenressource verknüpft sein muss, und 2., getrennt durch Leerzeichen, der URI der Datei mit den XML-Schema-Definitionen.

Dateien mit XML-Schema-Definitionen erhalten die typische Dateiendung .xsd, was für XML Schema Definition steht. Nachfolgendes Listing zeigt, wie der Inhalt der im Beispiel angegebenen Datei *members.xsd* aussehen könnte.

Listing 14.13: Datei ***members.xsd*** mit XML-Schema-Definitionen

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="members">
  <xs:complexType>
     <xs:element name="member" minOccurs="1"
      maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="given_name" type="xs:string" />
            <xs:element name="family_name" type="xs:string" />
            <xs:element name="member_since">
              <xs:complexType>
                 <xs:sequence>
                    <xs:element name="yyyy" type="xs:integer" />
                    <xs:element name="mm" type="xs:integer" />
                    <xs:element name="dd" type="xs:integer" />
                 </xs:sequence>
              </xs:complexType>
            </xs:element>
```

```
            </xs:sequence>
            <xs:attribute name="nr" use="required" type="xs:integer"/>
          <xs:attribute name="position" use="required" default="none">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="none" />
              <xs:enumeration value="committee" />
              <xs:enumeration value="sponsor" />
            </xs:restriction>
          </xs:simpleType>
          </xs:attribute>
          </xs:complexType>
        </xs:element>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

Sofort erkennbar ist, dass die XSD-Datei selbst ein XML-Dokument ist. Das Dokumentelement lautet `schema`. Der gesamte Inhalt einer XSD-Datei wird also zwischen Start- und End-Tag dieses Elements notiert. Allerdings wird bei XML Schema grundsätzlich mit »qualifizierten Namen« gearbeitet, also mit einem definierten Namensraum. Die Namensraumdefinition geschieht wie üblich durch das `xmlns`-Attribut im Start-Tag des Dokumentelements und lautet bei XML Schema üblicherweise:

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

Damit wird `xs` als Namensraumname deklariert. Dieser Namensraumname wird allen Elementnamen innerhalb des XSD-Dokuments vorangestellt.

Wie bei DTDs werden auch in XML Schema vor allem Elemente und Attribute definiert. Zum Definieren von Elementen dient das `element`-Element, markiert je nach Bedarf durch `<xs:element>...</xs:element>` oder `<xs:element ... />`. Attribute werden durch `<xs:attribute />...</xs:attribute>` oder `<xs:attribute ... />` definiert.

Die Standalone-Varianten kommen dann zum Einsatz, wenn ein Element oder Attribut nichts weiter als einfache Zeichendaten bzw. einen einfachen Wert enthalten soll. So wird im Beispiel etwa das Element `given_name` mit `<xs:element name="given_name" type="xs:string" />` definiert. Da dieses Element keine weiteren Elemente oder gemischten Inhalte enthalten soll, sondern einfach eine Zeichenkette, genügt die Angabe des Elementnamens, markiert durch das `name`-Attribut und die Angabe des Datentyps, markiert durch das `type`-Attribut.

Sobald ein Element Attribute oder als Inhalt weitere Elemente enthalten kann, ist es aus Sicht von XML Schema ein »komplexes Element«. Im Beispiel ist etwa `member_since` ein solches Element. Seine Definition wird durch Start- und End-Tag, also durch `<xs:element>...</xs:element>`, markiert. Dazwischen wird durch `<xs:complexType>...</xs:complexType>` angezeigt, dass das Element Kindelemente enthalten kann. Innerhalb davon wiederum wird durch `<xs:sequence>...</xs:sequence>` eine feststehende Reihenfolge von Kindelementen definiert. Für die Definition der drei Kindelemente selbst reichen wieder einfache Elementdefinitionen. Beim Datentyp ist erkennbar, dass XML Schema auch das Definieren von Datentypen wie Integer ermöglicht. Daneben erlauben es diverse Attri-

bute, die Definition für ein Element feinzusteuern. So tauchen in der Definition für das Element `member` im Beispiel die Attribute `minOccurs` und `maxOccurs` auf. Es wird bestimmt, dass das `member`-Element mindestens einmal vorkommen muss und beliebig oft vorkommen darf.

Attribute für Elemente werden innerhalb deren Definition (`<xs:element>...</xs:element>`) definiert. In unserem Beispiel werden für das `member`-Element die Attribute `nr` und `position` definiert. Während für die Definition von `nr` ein Standalone-Tag genügt, weil das Attribut einfach nur eine Integer-Zahl als Wert erhalten soll, ist für das `position`-Attribut eine komplexere Definition erforderlich. Durch `<xs:restriction>...</xs:restriction>` und davon eingeschlossene `xs:enumeration`-Elemente werden mögliche Werte definiert, die diesem Attribut zugewiesen werden dürfen.

Es würde den Rahmen sprengen, an dieser Stelle auf alle Möglichkeiten von XML Schema einzugehen. Das Beispiel soll nur einen allgemeinen Eindruck vermitteln, wie XML Schema aussieht und funktioniert. Erkennbar ist jedenfalls, dass bei Definitionen mit XML Schema deutlich mehr Möglichkeiten existieren als bei Definitionen in DTDs. Es bleibt zu hoffen, dass XML Schema mittelfristig das Konzept der DTDs als Basis für gültige XML-Dokumente ablösen wird.

14.7 Übersetzung von XML-Daten in HTML/XHTML

Eines der beliebtesten Schlagwörter in Zusammenhang mit XML ist ***Single Source Publishing***. Gemeint ist damit, dass Daten wie Produktbeschreibungen, Geschäftsberichte oder Handbücher nicht mehr an mehreren verschiedenen Stellen in unterschiedlichen Dateiformaten gepflegt werden sollten, weil das immer wieder zu unterschiedlichen Ständen und Mehrarbeit führt. Stattdessen sollten die Daten nur an einem einzigen Ort in abstrakter Form gespeichert werden. Aus dieser abstrakten Datenquelle sollen dann Zielformate wie HTML für Webseiten, PDF für Druckdokumente oder RTF für weiterbearbeitbare Zieldokumente auf Knopfdruck generiert werden können.

XML bietet sich für eine »abstrakte«, also zielformatunabhängige Datenspeicherung geradezu an. Denn XML erlaubt es, Daten, egal welcher Art, beliebig fein und semantisch sinnvoll zu beschreiben. Durch den Einsatz von DTDs oder XML-Schema-Definitionen lässt sich außerdem die Integrität der XML-Daten sicherstellen. Was noch fehlt, ist ein Verfahren zum Generieren von Zielformaten aus den XML-Daten.

An diesem Punkt setzt das XML-Konzept der ***Transformation*** an. Eine Transformation besteht darin, Daten oder bestimmte Teile davon aus einem XML-Quellformat in ein Zielformat zu konvertieren. Das Zielformat kann eine andere XML-Sprache sein, wie etwa XHTML für Webseiten. Es kann aber auch ein beliebiges anderes Klartextformat wie PDF (tagged), RTF, TIFF (tagged), TeX oder MIF sein. Da mittlerweile auch alle namhaften Office-Produkte nach und nach auf XML-basierte Formate als Default-Dateiformat umsteigen (OpenOffice auf das OpenDocument-Format und MS Office ab Office 2007 auf ein hauseigenes XML-Format), gewinnt das Konzept der Transformation aus individuell optimal gehaltenen XML-Daten in solche Zielformate weiter an Bedeutung.

Das häufigste Anwendungsgebiet für Transformationen dürfte allerdings das Konvertieren aus individuellen XML-Formaten nach HTML/XHTML sein. Deshalb konzentrieren sich die Beispiele in den nachfolgenden Abschnitten auf diese Quell-Ziel-Kombination.

14.7.1 Die Transformationssprache XSLT

XSLT ist eigentlich nur eine Komponente von XSL (*Extensible Stylesheet Language*). XSL ist ähnlich wie XML Schema selbst eine XML-Sprache, deren Aufgabe XML-intern ist. Die Aufgabe besteht darin, ein umfassendes Arsenal an Möglichkeiten zur Präsentation von XML-Daten zur Verfügung zu stellen. Zwar ist es auch möglich, XML-Daten mit Hilfe von CSS zu formatieren. XML-fähige Browser setzen die Resultate sogar leidlich um. CSS ist jedoch nicht wirklich für XML ausgelegt, sondern als reine Ergänzungssprache speziell für HTML gedacht. XSL dagegen ist von vorneherein in die XML-Welt integriert.

XSL besteht allerdings aus zwei Komponenten und damit eigentlich aus zwei Sprachen. Die eine Komponente ist die eigentliche Konkurrenzsprache zu CSS, also eine Sprache, mit deren Hilfe sich definieren lässt, wie XML-Daten auf welchen Ausgabegeräten dargestellt bzw. präsentiert werden sollen. Diese Komponente heißt XSL-FO, wobei das FO für *formatting objects* steht. Ebenso wie CSS stellt XSL-FO alles bereit, was für Layout und Formatierung benötigt wird, von der Textgestaltung über Ausrichtung, Hintergrundgestaltung bis hin zur freien Anordnung von Elementen.

Für XSLT dagegen gibt es in CSS kein Gegenstück. XSLT (das T steht für *Transformation*) beschreibt, wie Elemente, Attribute usw. einer XML-Quellsprache in Bestandteile einer Zielsprache übersetzt werden sollen. Da XML-Daten immer eine Baumstruktur darstellen, spricht man auch von Quellbaum und Zielbaum.

Angenommen, der folgende XML-Ausschnitt sei unser Quellbaum:

```
<member nr="19341" position="none">
   <given_name>Anna</given_name>
   <family_name>Lyse</family_name>
   <member_since>
      <yyyy>2003</yyyy><mm>10</mm><dd>18</dd>
   </member_since>
</member>
```

Weiter angenommen, aus dieser XML-Passage soll folgender HTML-Code (Zielbaum) für den Browser erzeugt werden.

```
<tr>
   <td>19341</td>
   <td>ordinary member</td>
   <td>Anna Lyse</td>
   <td>2003-10-18</td>
</tr>
```

Erkennbares Ziel ist, dass das Mitgliederverzeichnis in die Form einer HTML-Tabelle gebracht werden soll. Der Ausschnitt zeigt nur einen »Datensatz« des Mitgliederverzeichnisses, also die Daten eines bestimmten Mitglieds. Die Daten eines solchen Datensatzes

sollen jeweils in einer HTML-Tabellenzeile stehen. Erkennbar ist, dass Daten in andere Umgebungen gesetzt werden sollen. So wird die Mitgliedsnummer aus dem `nr`-Attribut des `member`-Elements zum Elementinhalt eines `td`-Elements. Der Inhalt des `position`-Attributs wird sogar ausgewertet und durch einen anderen Text ersetzt. Die Inhalte aus den Elementen `given_name` und `family_name` werden, durch ein Leerzeichen getrennt, in einem `td`-Element zusammengefügt. Auch die Inhalte aus den Elementen `yyyy`, `mm` und `dd` werden in einem `td`-Element zusammengefügt.

Adressierung mit XPath

Damit es möglich ist, Daten im Quellbaum gezielt zu selektieren, um sie im Zielbaum ebenso gezielt einsetzen zu können, ist es wichtig, die Daten genau adressieren zu können. Die Adressierungssyntax, die XSLT dazu verwendet, ist selbst Gegenstand einer Spezifikation des W3-Konsortiums und firmiert unter dem Namen XPath.

XPath erlaubt XML-Adressierungsangaben wie »den Wert des `nr`-Attributs vom dritten `member`-Element unterhalb von `members`«. In der verkürzten Schreibweise von XPath (es gibt auch eine ausführlichere Schreibweise) sieht das so aus:

```
/members/member[3]/@nr
```

Nichts Spektakuläres also, sondern einfach ein paar Syntaxregeln, die man sich merken muss. Wir werden noch genauer darauf eingehen.

XSLT ist allerdings eine Sprache, die, obwohl vom Typ her eine XML-basierte Auszeichnungssprache, teilweise Eigenschaften einer prozeduralen Programmiersprache hat. Kommen wir auf das Beispiel zurück, die Daten eines Mitgliederverzeichnisses in eine HTML-Tabelle zu transformieren. XSLT sorgt dafür, dass die Nutzdaten aus jedem einzelnen `member`-Element in je eine HTML-Tabellenzeile transformiert werden. Das XSLT-Dokument soll jedoch generisch sein, d.h., es soll nicht wissen, wie viele `member`-Elemente das Quelldokument aktuell hat. Zur Umsetzung in eine HTML-Tabelle benötigt es also so etwas wie eine Schleife mit der Bedingung »solange ein weiteres `member`-Element gefunden wird«. Und tatsächlich gibt es in XSLT Elemente, die zum Konstruieren solcher Schleifen dienen. An XPath stellt das jedoch ebenfalls Anforderungen. Denn wo es Schleifen gibt, sind auch Variablen für Zähler erforderlich und/oder Funktionen der Art »hol mir das `nr`-Attribut des `member`-Elements, welches im Schleifendurchgang das aktuelle ist«. Für solche und ähnliche Zwecke stellt XPath so genannte ***XPath-Funktionen*** bereit. Die wichtigsten davon werden wir kennen lernen

Links zu XSL, XSLT und XPath:

W3C-Übersicht zu XSL:
http://www.w3.org/Style/XSL/
XSLT 1.0 Empfehlung:
http://www.w3.org/TR/xslt
XML 2.0 Empfehlung (zum Redaktionszeitpunkt noch »Candidate«-Status):
http://www.w3.org/TR/xslt20/

XPath 1.0 Empfehlung:
http://www.w3.org/TR/xpath
XPath 1.0 Empfehlung (deutsche Übersetzung):
http://www.edition-w3c.de/TR/xpath
XPath 2.0 Empfehlung (zum Redaktionszeitpunkt noch »Candidate«-Status):
http://www.w3.org/TR/xpath20/
Wikipedia-Artikel über XSL:
http://de.wikipedia.org/wiki/XSL
Wikipedia-Artikel über XSLT:
http://de.wikipedia.org/wiki/XSLT
Wikipedia-Artikel über XPath:
http://de.wikipedia.org/wiki/XPath

14.7.2 XSLT-Prozessoren

Eine Datei mit einem XSLT-Dokument enthält also eine allgemeine (generische) Beschreibung, wie aus Daten eines XML-Quellbaums eine Ausgabe in einem anderen Zielformat erzeugt werden soll. Damit aus diesen beschreibenden Transformationsvorschriften eine tatsächliche Datenkonvertierung wird, ist natürlich Software erforderlich, die XSLT »versteht« und die darin enthaltenen Transformationsvorschriften auf konkrete Quelldaten anwenden kann. Eine Software, die dies leistet, wird als **XSLT-Prozessor** bezeichnet.

In Server-Umgebungen ist der XSLT-Prozessor in der Regel ein Programm, das wie ein Filter bzw. wie typische Unix/Linux-Shell-Programme arbeitet. Der XSLT-Prozessor erhält als Eingabe die Pfadangaben zu einem XML-Dokument, das transformiert werden soll, und zu einem XSLT-Dokument, das die Transformationsvorschriften enthält. Daraus erzeugt er Daten im Zielformat, die er am Ende ausgibt, wahlweise auf die Standardausgabe oder als Umlenkung in eine Ausgabedatei. Ein anderer Prozess, z.B. ein PHP-Script, das den Vorgang angestoßen hat, kann den Output des XSLT-Prozessors von der Standardausgabe auffangen und beispielsweise zum aufrufenden Webbrowser senden.

Links zu bekannten XSLT-Prozessoren

Xalan (en, OpenSource): *http://xml.apache.org/xalan-j/*
vom Apache-Projekt entwickelter, java-basierter XSLT-Prozessor (mittlerweile auch in einer C++-Implementierung verfügbar). Bestens geeignet zur Integration in eine Apache-Webserverumgebung

Saxon (en, OpenSource): *http://xml.apache.org/xalan-j/*
Schneller und unkomplizierter XSLT-Prozessor, auch als vorkompilierte Windows-Version (.NET-Umgebung erforderlich, läuft unter Windows als Filterprogramm in einem DOS-Fenster) erhältlich für den Einsatz an einem lokalen Windows-PC. Geeignet also zum lokalen Austesten.

Sablotron (en, OpenSource): *http://www.gingerall.com/charlie/ga/xml/p_sab.xml*
Schneller XSLT-Prozessor, auch als vorkompilierte Windows-Version (Expat-XML-Parser-Bibliothek erforderlich, erhältlich unter *http://sourceforge.net/projects/expat/*; läuft unter Windows als Filterprogramm in einem DOS-Fenster) erhältlich für den Einsatz an einem lokalen Windows-PC. Geeignet also zum lokalen Austesten.

14.7.3 Ein einfaches Transformationsbeispiel

Nach den vielen, zum Verständnis allerdings auch erforderlichen, Informationen wenden wir uns nun konkreten XSLT-Beispielen zu. Ein erstes, einfaches Beispiel soll die Arbeitsweise von XSLT demonstrieren. Gegeben sei folgender Quellbaum:

```
<?xml version="1.0" encoding=" UTF-8"?>
<book isbn="3-8273-2218-9">
   <author>Stefan Münz</author>
   <title>Professionelle Websites</title>
   <lang>de</lang>
   <content>
      <keywords>HTML, CSS, JavaScript, DOM, PHP, MySQL</keywords>
      <description>Einführungswerk für Web-Entwickler, die
      alle heute maßgeblichen Technologien und Aspekte der
      Erstellung professioneller Websites kennen und anwenden
      wollen.</description>
   </content>
</book>
```

Daraus soll eine sinnvolle HTML-Ausgabe erzeugt werden. Das XSLT-Dokument, das die Transformationsvorschriften beschreibt, soll den Dateinamen ***books.xsl*** erhalten. Die Dateiendung .xsl ist typisch für XSL(T)-Dokumente.

In der Datei ***books.xsl*** notieren wir zunächst das XSLT-Grundgerüst:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<!-- HIER FOLGT DER INHALT -->
</xsl:stylesheet>
```

Wie erkennbar, handelt es sich um ein typisches wohlgeformtes XML-Dokument, beginnend mit einer XML-Deklaration. Da innerhalb von XSLT Elemente und Attribute anderer Auszeichnungssprachen vorkommen, wird, um Namenskonflikte zu vermeiden, von vorneherein mit einem definierten Namensraum gearbeitet. Dieser erhält wie üblich über das `xmlns`-Attribut im Dokumentelement (`xsl:stylesheet`) den typischen Namensraumnamen `xsl`. Als Namensraumbezeichner wird der URN `http://www.w3.org/1999/XSL/Transform` angegeben.

Referenziert wird im Beispiel die XSLT-Version 1.0. Wenn Sie Version 2.0 referenzieren wollen, müssen Sie stattdessen notieren:

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Der Namensraumbezeichner bleibt also gleich, nur die Versionsangabe wird geändert.

Bereits das Dokumentelement selbst wird mit dem Namensraumnamen `xsl` als qualifizierter Name (`xsl:stylesheet`) ausgezeichnet.

Templates und Zugriffe auf Inhalte und Werte

An der Stelle im Dokument, an der vorläufig der Kommentartext HIER FOLGT DER INHALT notiert ist, wird nun folgender Inhalt notiert:

```
<xsl:template match="/">
   <html>
   <head>
   <title><xsl:value-of select="book/title" /></title>
   <meta http-equiv="content-type" content="text/html; charset= UTF-8" />
   </head>
   <body>
   <h1><xsl:value-of select="book/title" /></h1>
   <ul>
   <li><b>Autor:</b> <xsl:value-of select="book/author" /></li>
   <li><b>ISBN:</b> <xsl:value-of select="book/@isbn" /></li>
   <li><b>Kurzbeschreibung:</b><br />
   <xsl:value-of select="book/content/description" /></li>
   </ul>
   </body>
   </html>
</xsl:template>
```

Bei diesem Konstrukt handelt es sich um eine Template-Vorschrift (***template rule***) zur Transformation vom Quellbaum zum Zielbaum. Eine solche Template-Vorschrift wird in `<xsl:template...>...</xsl:template>` eingeschlossen.

Im einleitenden `<xsl:template>`-Tag wird ein Attribut `match=` notiert. Diesem Attribut wird zugewiesen, auf welchen »Knoten« des Quellbaums die Template-Vorschrift angewendet werden soll. Im obigen Beispiel wird `"/"` zugewiesen. Damit wird die abstrakte Dokumentwurzel (noch nicht das Dokumentelement) des Quellbaums adressiert. Die Adressierungssyntax ist ein so genannter XPath-Ausdruck (***XPath expression***). Auf diese Syntax kommen wir noch mehrfach und näher zurück.

Im Augenblick geht es darum zu verstehen, wie ein XSLT-Dokument aufgebaut ist. Eine Template-Vorschrift wie im obigen Beispiel kann als Elementinhalt gewünschten Code der Zielsprache enthalten. In unserem Beispiel besteht der Elementinhalt aus einem kompletten kleinen HTML-Dokument. An einigen Stellen des HTML-Dokuments sind jedoch wiederum `<xsl:value-of-select>`-Tags notiert. Diese erlauben den gezielten Zugriff auf Inhalte aus dem Quellbaum. Welche Inhalte jeweils genau selektiert werden, wird durch ein Attribut namens `select=` bestimmt. Diesem wird wieder ein XPath-Ausdruck zugewiesen, um den gewünschten Inhalt zu adressieren. Beispiel:

```
<title><xsl:value-of select="book/title" /></title>
```

Hier wird im Zielbaum (HTML) ein `title`-Element erzeugt. Als Inhalt wird der Elementinhalt aus dem `title`-Element des Quelldokuments geholt. Hier noch einmal der entsprechende Ausschnitt aus dem Quelldokument:

```
<book isbn="3-8273-2218-9">
   <author>Stefan Münz</author>
   <title>Professionelle Websites</title>
```

usw. Im erzeugten HTML-Dokument wird also am Ende stehen:

```
<title> Professionelle Websites</title>
```

Da wir uns innerhalb einer Template-Vorschrift für die abstrakte Dokumentwurzel (`"/"`) befinden, müssen wir die gewünschten Inhalte, auf die mit `<xsl:value-of-select>` zugegriffen werden soll, »aus Sicht« der abstrakten Dokumentwurzel adressieren. Mit `select="book/title"` wird beispielsweise der Inhalt des `title`-Elements adressiert, das ein Kindelement des Dokumentelements `book` ist.

Noch ein weiterer Ausschnitt aus der Template-Vorschrift lautet:

```
<li><b>ISBN:</b> <xsl:value-of select="book/@isbn" /></li>
```

Damit wird im HTML-Zielbaum ein Listenelement erzeugt. Es soll die ISBN-Nummer enthalten. Durch `select="book/@isbn"` wird der Wert des Attributs `isbn=` des Dokumentelements `book` adressiert.

Insgesamt wird aus dem Quellbaum unseres Beispiels mit Hilfe des vorgestellten XSLT-Dokuments folgendes HTML-Dokument erzeugt:

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Professionelle Websites</title>
   </head>
   <body>
      <h1>Professionelle Websites</h1>
      <ul>
         <li><b>Autor:</b>Stefan Münz
         </li>
         <li><b>ISBN:</b>3-8273-2218-9
         </li>
         <li><b>Kurzbeschreibung:</b><br>Einführungswerk für
            Web-Entwickler, die
            alle heute maßgeblichen Technologien und Aspekte der
            Erstellung professioneller Websites kennen und anwenden
            wollen.
         </li>
      </ul>
   </body>
</html>
```

Wenn Sie beispielsweise Saxon als XSLT-Prozessor installiert haben, können Sie an der Kommandozeile Folgendes eingeben:

```
transform book.xml book.xsl > book.htm
```

Dabei ist vorausgesetzt, dass die Programmdatei ***transform*** (***transform.exe***) im Pfad liegt. Ansonsten muss beim Aufruf der Pfadname mit notiert werden. Im Beispiel ist ***book.xml*** die Datei mit dem XML-Quellbaum-Dokument, ***book.xsl*** das XSLT-Dokument mit den

Transformationsvorschriften und die Angabe *> book.htm* lenkt den Output des XSLT-Prozessors, also das aus dem Quellbaum erzeugte HTML-Dokument, von der Standardausgabe in eine Datei namens *book.htm* um.

Einzelheiten zu den genauen Aufrufmöglichkeiten, möglichen Problemen und Fehlern entnehmen Sie der Dokumentation des XSLT-Prozessors, den Sie einsetzen.

14.7.4 Wichtige XSLT-Elemente

Von XSLT haben wir nun bereits das Dokumentelement `xsl:stylesheet` sowie die Elemente `xsl:template` und `xsl:value-of` kennen gelernt. Anhand des nachfolgenden Beispiels werden wir weitere Elemente kennen lernen. Zunächst das XML-Dokument, dessen Daten in HTML transformiert werden sollen (das Quelldokument):

```
<?xml version="1.0"?>
<members>
   <member nr="19341" position="none">
      <given_name>Anna</given_name>
      <family_name>Lyse</family_name>
      <member_since>
         <yyyy>2003</yyyy><mm>10</mm><dd>18</dd>
      </member_since>
   </member>
   <member nr="11294" position="committee">
      <given_name>Reiner</given_name>
      <family_name>Wahnsinn</family_name>
      <member_since>
         <yyyy>1998</yyyy><mm>02</mm><dd>23</dd>
      </member_since>
   </member>
   <member nr="20845" position="none">
      <given_name>Mira</given_name>
      <family_name>Mare</family_name>
      <member_since>
         <yyyy>2005</yyyy><mm>06</mm><dd>01</dd>
      </member_since>
   </member>
</members>
```

Die Daten dieses XML-Dokuments sind datensatzorientiert, d.h., es gibt sich wiederholende Elemente und Unterstrukturen – im Beispiel kommen drei `members`-Elemente vor. Für eine Transformation bietet es sich an, diese Daten in die Form einer HTML-Tabelle zu bringen. Das XSLT-Dokument für die Transformation soll natürlich immer anwendbar sein, egal ob im Quelldokument aktuell drei oder dreitausend member-Elemente gespeichert sind. Allerdings soll die Aufgabe darin bestehen, nur die Daten von Mitgliedern auszugeben, deren Mitgliedschaft vor 2005 begann.

Diese Aufgabenstellung lösen wir mit Hilfe des nachfolgenden XSLT-Stylesheet:

```
<?xml version="1.0" encoding=" UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
<xsl:template match="/">
   <html>
   <head>
   <title>Mitglieder</title>
   <meta http-equiv="content-type" content="text/html; charset= UTF-8" />
   </head>
   <body>
   <h1>Mitglieder</h1>
   <table border="1">
   <tr>
   <th style="text-align:left">Mitgliedsnummer</th>
   <th style="text-align:left">Mitgliedsname</th>
   <th style="text-align:left">Mitglied seit</th>
   </tr>
   <xsl:apply-templates />
   </table>
   </body>
   </html>
</xsl:template>
<xsl:template match="members">
  <xsl:for-each select="member">
    <xsl:if test="member_since/yyyy &lt; 2005">
      <tr>
      <td style="text-align:left"><xsl:value-of select="@nr" /></td>
      <td style="text-align:left"><xsl:value-of select="given_name" />
<xsl:value-of select="family_name" /></td>
      <td style="text-align:left"><xsl:value-of select="member_since/dd" /
>.<xsl:value-of select="member_since/mm" />.<xsl:value-of select="member_since/
yyyy" /></td>
  </tr>
</xsl:if>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Auf das zuvor gezeigte XML-Dokument angewendet, liefert dieses XSLT-Stylesheet folgenden HTML-Code:

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Mitglieder</title>
   </head>
   <body>
      <h1>Mitglieder</h1>
      <table border="1">
         <tr>
            <th style="text-align:left">Mitgliedsnummer</th>
            <th style="text-align:left">Mitgliedsname</th>
            <th style="text-align:left">Mitglied seit</th>
         </tr>
         <tr>
            <td style="text-align:left">19341</td>
```

```
            <td style="text-align:left">AnnaLyse</td>
            <td style="text-align:left">18.10.2003</td>
         </tr>
         <tr>
            <td style="text-align:left">11294</td>
            <td style="text-align:left">ReinerWahnsinn</td>
            <td style="text-align:left">23.02.1998</td>
         </tr>
      </table>
   </body>
</html>
```

Erkennbar ist, dass in der erzeugten HTML-Tabelle nur zwei der drei in XML gespeicherten Mitglieder auftauchen. Das dritte Mitglied wurde vom XSLT-Stylesheet herausgefiltert, weil sein Eintrittsdatum nicht vor 2005 lag.

Im Stylesheet fallen gleich mehrere Neuigkeiten auf: Statt eines einzigen `xsl:template`-Elements sind in diesem Fall zwei notiert. In der Praxis werden die Transformationsvorschriften meistens auf mehrere Template-Angaben verteilt. Wichtig ist, den Zusammenhang zwischen den Templates zu verstehen.

Mehrere Templates

Die Schnittstelle zwischen den Templates ist das Element `<xsl:apply-templates />`. In dieser Form notiert, bedeutet es so viel wie: »Wende an dieser Stelle alle anderen Templates an.« Um nur ein bestimmtes Template anzuwenden, kann das Element ein `select`-Attribut erhalten. Beispiel:

```
<xsl:apply-templates select="members"/>
```

Der Wert im `select`-Attribut ist wieder eine XPath-Pfadangabe.

Eine andere Möglichkeit besteht darin, Templates über einen Namen aufzurufen. Dazu wird anstelle von `<xsl:apply-templates select="..."/>` notiert:

```
<xsl:call-template name="template_members" />
```

Unter dem entsprechenden Namen muss das Template wie folgt definiert sein:

```
<xsl:template name="template_members">
  <!-- Template-Inhalt -->
</xsl:template>
```

Die Ausführung aufgerufener anderer Templates erfolgt genau an der Stelle, an der der Template-Aufruf notiert ist. In unserem Beispiel wird im ersten Template das HTML-Grundgerüst notiert und von der Ausgabetabelle nur das Tabellengrundgerüst sowie die Kopfzeile. An der Stelle, an der die Datenzeilen beginnen, ist der Template-Aufruf notiert.

Das Arbeiten mit Template-Namen ist praktisch, wenn die XPath-Adressierung sehr umständlich wäre.

Schleifen und Bedingungen

Das andere Template kümmert sich folglich um die Ausgabe der Datenzeilen. Verarbeitet werden dabei die `member`-Elemente aus dem XML-Quelldokument. Da im Quellbaum mehrere solche Elemente existieren, ist zur Verarbeitung ein Konstrukt erforderlich, das an eine foreach-Schleife aus Programmiersprachen erinnert. Und tatsächlich stellt XSLT mit `<xsl:for-each...>...</xsl:foreach>` ein solches Element zur Verfügung. Im `select`-Attribut dieses Elements wird der XPath-Pfad zu einem Set von Elementen angegeben, die auf gleicher Hierarchieebene nacheinander notiert sind.

In unserem Beispiel werden also mit dem `xsl:for-each`-Element der Reihe nach alle `member`-Elemente abgearbeitet. Die zusätzliche Aufgabenstellung bestand allerdings darin, nur solche Mitglieder aufzulisten, deren Mitgliedschaft vor 2005 begann. Solche Einschränkungen lassen sich wieder nur durch ein programmiertypisches Konstrukt lösen, nämlich durch eine Bedingung. Dazu bietet XSLT das Element `<xsl:if...>...</xsl:if>` an. Das Element erhält ein Attribut namens `test=`. Die Wertzuweisung an dieses Attribut ist eine Bedingung. Eine Bedingung kann Verschiedenes sein. In unserem Beispiel wird abgefragt, ob der Inhalt des aktuellen Quellbaumelements `yyyy` (das Mitgliedsbeitrittsjahr) kleiner als 2005 ist. Der Elementinhalt des `yyyy`-Elements wird dabei wieder über XPath-Syntax adressiert. Das Kleiner-als-Zeichen wird in HTML/XML-typischer Umschreibung (`<`) notiert, um beim Parser Interpretationskonflikte zu vermeiden. An der gleichen Stelle wären auch `>` für »größer als«, `<=` für »kleiner oder gleich«, `>=` für »größer oder gleich, `=` für »ist gleich« oder `!=` für »ist ungleich« erlaubt. Mehrere Bedingungen lassen sich durch die Schlüsselwörter `and` für »und« und `or` für »oder« (genauer: »entweder-oder«) verknüpfen. Auch Rechenoperationen, Zahlen, Zeichenketten oder Aufrufe von XPath-Funktionen dürfen in Bedingungen vorkommen.

Ist die Bedingung im Beispiel-Stylesheet erfüllt, wird eine HTML-Tabellenzeile mit den Mitgliederdaten des jeweils bearbeiteten `member`-Elements erzeugt. Über `<xsl:value-of select select="...">` mit XPath-Syntax wird auf die Inhalte der Unterelemente und Attribute zugegriffen.

Fallunterscheidungen

Um nur Mitglieder herauszufiltern, deren Eintrittsdatum vor 2005 liegt, haben wir uns im Beispiel des `xsl:if`-Elements bedient. Für einfache Bedingungen genügt das. Sollen jedoch mehrere mögliche Bedingungen mit unterschiedlichen Folgen unterschieden werden, so ist eine Fallunterscheidung erforderlich. Angenommen, im Beispiel sollen die Tabellenzeilen abhängig von der Länge der bisherigen Mitgliedschaft eine unterschiedliche Hintergrundfarbe erhalten. Dies ist wie folgt lösbar:

```
<xsl:template match="members">
  <xsl:for-each select="member">
    <xsl:choose>
      <xsl:when test="member_since/yyyy &gt; 2004">
         <tr style="background-color:#FFFFCC">
         <td><xsl:value-of select="@nr" /></td>
         <td><xsl:value-of select="given_name" />
```

```
            <xsl:value-of select="family_name" /></td>
          <td><xsl:value-of select="member_since/dd" />.
            <xsl:value-of select="member_since/mm" />.
            <xsl:value-of select="member_since/yyyy" /></td>
          </tr>
        </xsl:when>
        <xsl:when test="member_since/yyyy &gt;= 2000 and
                        member_since/yyyy &lt;= 2003">
          <tr style="background-color:#CCFFFF">
          <td><xsl:value-of select="@nr" /></td>
          <td><xsl:value-of select="given_name" />
            <xsl:value-of select="family_name" /></td>
          <td><xsl:value-of select="member_since/dd" />.
            <xsl:value-of select="member_since/mm" />.
            <xsl:value-of select="member_since/yyyy" /></td>
          </tr>
        </xsl:when>
        <xsl:otherwise>
          <tr style="background-color:#FFCCFF">
          <td><xsl:value-of select="@nr" /></td>
          <td><xsl:value-of select="given_name" />
            <xsl:value-of select="family_name" /></td>
          <td><xsl:value-of select="member_since/dd" />.
            <xsl:value-of select="member_since/mm" />.
            <xsl:value-of select="member_since/yyyy" /></td>
          </tr>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each>
  </xsl:template>
```

Das Beispiel zeigt den XSLT-Stylesheet-Ausschnitt für `<xsl:template match="members">...</xsl:template>`, innerhalb dessen die einzelnen `member`-Elemente in einer Schleife abgearbeitet werden.

Anstelle des `xsl:if`-Elements ist nun ein `xsl:choose`-Element notiert. Alles, was zwischen `<xsl:choose>` und `</xsl:choose>` steht, gehört zur Fallunterscheidung. Die einzelnen Fälle werden durch `<xsl:when...>...` `</xsl:when>` markiert. Der »Restfall« wird innerhalb von `<xsl:otherwise>...</xsl:otherwise>` behandelt. Ebenso wie bei `xsl:if` wird bei `xsl:when` mit Hilfe des `test`-Attributs eine Bedingung formuliert. Im Beispiel werden folgende Fälle und Konsequenzen unterschieden:

Tabellenzeilen von Mitgliedern mit Eintrittsdatum nach 2004 erhalten einen hellgelben Hintergrund (CSS-Definition im einleitenden `<tr>`-Tag!), solche von Mitgliedern, deren Eintrittsdatum zwischen 2000 und 2003 liegt, eine hellblaue Hintergrundfarbe und Zeilen mit älteren Mitgliedern einen hellgrünen Hintergrund.

Elemente und Attribute im Ergebnisbaum erzeugen

In der Regel genügt es, die Markup-Struktur des Zielformats einfach direkt zu notieren, wie in den bisherigen Beispielen. Wenn jedoch Daten aus dem Quellbaum in Attribute

übernommen werden sollen, kommt man damit ohne Syntaxverletzung nicht mehr weiter. Daher bietet XSLT die Möglichkeit an, Elemente und Attribute des Zielformats explizit zu definieren. Nehmen wir folgende XML-Datei als Quellbaum an:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<foto>
  <url>/images/fotos/hamburg/michel.png</url>
  <data>
    <title>Der Hamburger Michel</title>
    <author>Rainer Wahnsinn</author>
    <timestamp>2006-04-19 13:24</timestamp>
    <px-width>2272</px-width>
    <px-height>1704</px-height>
  </data>
</foto>
```

In diesen Daten stehen Inhalte wie die URL-Adresse des Fotos, der Bildtext (Titel) sowie Angaben zu Breite und Höhe in Elementinhalten. In den Zieldaten sollen diese Daten in einem HTML-img-Element untergebracht werden, und zwar als Wertzuweisung an die Attribute `src=`, `alt=`, `width=` und `height=`.

Folgendes XSLT-Stylesheet löst diese Aufgabe:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:attribute-set name="img_url">
  <xsl:attribute name="src">
    <xsl:value-of select="/foto/url" />
  </xsl:attribute>
</xsl:attribute-set>
<xsl:attribute-set name="img_width">
  <xsl:attribute name="width">
    <xsl:value-of select="/foto/data/px-width" />
  </xsl:attribute>
</xsl:attribute-set>
<xsl:attribute-set name="img_height">
  <xsl:attribute name="height">
    <xsl:value-of select="/foto/data/px-height" />
  </xsl:attribute>
</xsl:attribute-set>
<xsl:attribute-set name="img_alt">
  <xsl:attribute name="alt">
    <xsl:value-of select="/foto/data/title" />
  </xsl:attribute>
</xsl:attribute-set>

<xsl:template match="/">
   <html>
   <head>
   <title><xsl:value-of select="foto/data/title" /></title>
   <meta http-equiv="content-type" content="text/html; charset=utf-8" />
   </head>
   <body>
```

```
    <h1><xsl:value-of select="foto/data/title" /></h1>
    <p>
      <xsl:element name="img" use-attribute-sets="img_url img_width img_height
img_alt"></xsl:element>
    </p>
    </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

Das Stylesheet transformiert die zuvor gezeigte XML-Datei in folgenden HTML-Code:

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Der Hamburger Michel</title>
   </head>
   <body>
      <h1>Der Hamburger Michel</h1>
      <p><img src="/images/fotos/hamburg/michel.png" width="2272" height="1704"
alt="Der Hamburger Michel"></p>
   </body>
</html>
```

Im XSLT-Stylesheet haben wir mit Hilfe des `xsl:attribute-set`-Elements Attribute für den Zielbaum in HTML definiert. Diese Elemente müssen außerhalb von `xsl:template`-Elementen definiert werden, in der Regel so wie im obigen Beispiel am Dokumentanfang. Jedes `xsl:attribut-set`-Element erhält über das `name`-Attribut im Start-Tag einen Namen. Unter diesem Namen kann es innerhalb von `xsl:template`-Elementen verwendet werden. Als Elementinhalt wird ein `xsl:attribute`-Element notiert. Auch dieses Element muss ein `name`-Attribut im Start-Tag haben. Dessen Wert ist jedoch kein frei vergebbarer Name, sondern der Name des gewünschten Attributs im Zielbaum. Als Elementinhalt kann statischer Text notiert werden, oder, wie im obigen Beispiel, mit Hilfe des `xsl:value-of`-Elements ein bestimmter Inhalt aus dem Quellbaum geholt werden.

In unserem Beispiel werden auf diese Weise die Zielattribute mit den Namen `src`, `alt`, `width` und `height` definiert und mit den entsprechenden Inhalten aus den XML-Quelldaten versorgt.

Innerhalb der `xsl:template`-Definition werden die so definierten Attribute an geeigneter Stelle eingesetzt. Das geht jedoch nur, indem man auch betroffene Elemente über das entsprechende XSLT-Element definiert. Im Beispiel wird das HTML-`img`-Element auf diese Weise erzeugt. Das XSLT-Element dazu lautet `xsl:element`. Auch dieses Element muss ein Attribut `name=` erhalten, dem als Wert der Name des Zielelements zugewiesen wird. In unserem Fall also `img`. Über das zweite Attribut `use-attribute-sets` können ein oder mehrere (durch Leerzeichen getrennte) Attribute für das so definierte Element übernommen werden. Anzugeben sind dabei die Namen, die bei `<xsl:attribute-set name="...">` vergeben wurden.

Gleichnamige Elemente aus Quellbaum in Zielbaum kopieren

Nicht selten gibt es Elemente, die im Zieldokument den gleichen Namen haben wie in den XML-Quelldaten. Viele proprietäre XML-Formate zum Speichern von Textdaten benutzen beispielsweise durchaus HTML-konforme Namen wie `h1`, `p` oder `strong` für entsprechende Textauszeichnungen. Solche Elemente lassen sich mit XSLT besonders einfach kopieren. Ein Beispiel:

```
<xsl:template match="p">
<xsl:copy><xsl:value-of select="." /></xsl:copy>
</xsl:template>
```

Über das Template werden `p`-Elemente im XML-Quellbaum angesprochen. Diese werden durch das `xsl:copy`-Element, das innerhalb der Template-Definition notiert ist, zusammen mit dem `xsl:value`-Element und der Auswahl `select="."` (Inhalt des jeweiligen `p`-Elements) 1:1 in die Zieldaten kopiert.

Variablen einsetzen

Ein weiteres Feature von XSLT sind Variablen. Diese erlauben es, einen Wert (z. B. eine Zeichenkette oder eine Zahl) unter einem Namen zu speichern und an gewünschter Stelle in die Zieldaten einzufügen.

Allerdings haben XSLT-Variablen gegenüber Variablen, wie sie aus Programmiersprachen bekannt sind, den Nachteil, dass sie nur initialisierbar, aber nicht änderbar sind. Das Feature eignet sich also beispielsweise nicht für Schleifendurchlaufzähler und ähnliche Anwendungen. Sinnvoll ist der Einsatz, wenn Daten an einer Stelle ermittelt oder definiert werden sollen, um sie dann an mehreren Stellen wiederzuverwenden. Dadurch wird ein umfangreiches XSLT-Stylesheet besser wartbar. Dazu ein denkbarer Ausschnitt aus dem bereits bekannten Stylesheet zur Ausgabe eines Mitgliederverzeichnisses:

```
<xsl:variable name="member_count">
  <xsl:value-of select="count(/members/member)" />
</xsl:variable>
<xsl:template match="/">
   <html>
   <head>
   <title>Mitglieder (<xsl:value-of select="$member_count" />)</title>
   <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
   </head>
   <body>
   <h1>Mitglieder (<xsl:value-of select="$member_count" />)</h1>
   <!-- usw. -->
</xsl:template>
```

Mit Hilfe von `<xsl:variable...>...</xsl:variable>` wird eine Variable definiert. Den Namen bestimmen Sie mit dem Attribut `name=` . Die Wertzuweisung an die Variable kann entweder im Elementinhalt erfolgen – so wie in unserem Beispiel – oder das `xsl:variable`-Element wird als Standalone-Tag notiert und der Wert wird mit Hilfe des zusätzlichen Attributs `select=` zugewiesen. Für unser Beispiel hätte das genügt, aber die Variante, den Wert über den Elementinhalt zuzuweisen, erlaubt mehr Freiheiten. So wäre es etwa pro-

blemlos möglich, zwischen <xsl:variable> und </xsl:variable> ein xsl:choose-Element zu notieren, also eine Fallunterscheidung, und der Variable abhängig von den Gegebenheiten unterschiedliche Werte zuzuweisen.

Im Beispiel wird der Wert durch ein xsl:value-of-Element zugewiesen, genauer, durch dessen Angabe beim select-Attribut. Das ist im Beispiel ein Aufruf der XPath-Funktion count(), die in diesem Fall die Anzahl der vorhandenen member-Elemente im XML-Quellbaum ermittelt. Es gibt eine ganze Reihe solcher XPath-Funktionen, auf die wir im nächsten Abschnitt näher eingehen werden.

Variablen können sowohl außerhalb von xsl:template-Elementen notiert werden – wie im Beispiel – als auch innerhalb davon. Werden sie außerhalb notiert, ist ihr Geltungsbereich global im gesamten XSLT-Stylesheet. Werden sie innerhalb von xsl:template definiert, sind sie nur innerhalb der jeweiligen Template-Definition »bekannt«.

Auf eine definierte Variable kann an geeigneter Stelle über die Syntax $Variablenname zugegriffen werden, also durch Voranstellen eines Dollarzeichens. Im Beispiel sind zwei Tags (<xsl:value-of select="$member_count" /> notiert. Damit wird an der betreffenden Stelle die Anzahl der zuvor ermittelten member-Elemente ausgegeben.

14.7.5 XPath-Funktionen für die Transformation

Die Aufgabe der XPath-Syntax erschöpft sich nicht darin, festzulegen, wie Inhalte aus dem XML-Quellbaum adressiert werden. Ein weiterer Beitrag, den XPath leistet, sind eine Reihe von Funktionen. Eine davon haben wir bereits kennen gelernt: die Funktion count(), welche die Anzahl von gleichnamigen Unterelementen zu einem Ausgangsknoten im Quellbaum ermittelt. Nachfolgendes Beispiel stellt weitere XPath-Funktionen vor. Als XML-Quelldokument verwenden wir dazu folgendes Beispiel:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<products>
  <product nr="kr345">
    <title>Krawuse XL3</title>
    <price>
  <value offer="standard">39.90</value>
      <valid-from>2006-04-15</valid-from>
      <valid-until>2006-05-31</valid-until>
</price>
  </product>
  <product nr="hw8245">
    <title>Pirolle &quot;Abalim&quot;</title>
    <price>
  <value offer="special">24.95</value>
      <valid-from>2006-04-15</valid-from>
      <valid-until>2006-06-30</valid-until>
</price>
  </product>
  <product nr="cg674">
    <title>Jumelbrase XL</title>
    <price>
```

```
    <value offer="special">1295</value>
        <valid-from>2006-03-01</valid-from>
        <valid-until>2006-05-31</valid-until>
  </price>
    </product>
  </products>
```

Auf diesen XML-Quellbaum wenden wir nun folgendes XSLT-Stylesheet an:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:decimal-format name="german" decimal-separator=","
     grouping-separator="."/>
<xsl:attribute-set name="dark">
  <xsl:attribute name="style">background-color:#D0D0D0
  </xsl:attribute>
</xsl:attribute-set>
<xsl:attribute-set name="light">
  <xsl:attribute name="style">background-color:#F0F0F0
  </xsl:attribute>
</xsl:attribute-set>
<xsl:attribute-set name="special">
  <xsl:attribute name="style">background-color:#CCFFCC
  </xsl:attribute>
</xsl:attribute-set>
<xsl:template match="/">
   <html>
   <head>
   <title>Angebote</title>
   <meta http-equiv="content-type"
         content="text/html; charset=utf-8" />
   </head>
   <body>
   <h1>Angebote</h1>
   <table border="1">
   <tr>
   <th style="text-align:left">Produkt</th>
   <th style="text-align:left">Titel</th>
   <th style="text-align:right">Preis</th>
   <th style="text-align:center">gültig bis</th>
   </tr>
   <xsl:apply-templates />
   </table>
   </body>
   </html>
</xsl:template>
<xsl:template match="products">
  <xsl:for-each select="product">
    <xsl:choose>
      <xsl:when test="price/value/@offer='special'">
        <xsl:element name="tr" use-attribute-sets="special">
          <td><xsl:value-of select="@nr" /></td>
```

```
            <td><xsl:value-of select="title" /></td>
            <td style="text-align:right">&#x20AC; <xsl:value-of
        select="format-number(price/value,'#.##0,00','german')"/>
          </td>
            <td style="text-align:center"><xsl:value-of
        select="substring(price/valid-until, 9, 2)" />.
          <xsl:value-of
        select="substring(price/valid-until, 6, 2)" />.</td>
          </xsl:element>
        </xsl:when>
        <xsl:otherwise>
          <xsl:choose>
      <xsl:when test="position() mod 2 = 0">
            <xsl:element name="tr" use-attribute-sets="dark">
              <td><xsl:value-of select="@nr" /></td>
              <td><xsl:value-of select="title" /></td>
              <td style="text-align:right">&#x20AC; <xsl:value-of
        select="format-number(price/value, '#.##0,00','german')"/>
            </td>
              <td style="text-align:center"><xsl:value-of
        select="substring(price/valid-until, 9, 2)" />.
            <xsl:value-of
        select="substring(price/valid-until, 6, 2)" />.</td>
            </xsl:element>
          </xsl:when>
          <xsl:otherwise>
            <xsl:element name="tr" use-attribute-sets="light">
              <td><xsl:value-of select="@nr" /></td>
              <td><xsl:value-of select="title" /></td>
              <td style="text-align:right">&#x20AC; <xsl:value-of
        select="format-number(price/value,'#.##0,00','german')"/>
            </td>
              <td style="text-align:center"><xsl:value-of
        select="substring(price/valid-until, 9, 2)" />.
            <xsl:value-of
        select="substring(price/valid-until, 6, 2)" />.</td>
            </xsl:element>
          </xsl:otherwise>
              </xsl:choose>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each>
  </xsl:template>
  </xsl:stylesheet>
```

Wie aus dem Ergebnis-Screenshot deutlich wird, wird auch in diesem Fall eine HTML-Tabelle erzeugt, um die Originaldaten aufzubereiten. Optisch fällt dabei sofort die unterschiedliche Färbung der Tabellenzeilen auf. Das Beispiel erzeugt Tabellenzeilen, die abwechselnd einen helleren oder dunkleren Hintergrund haben, wie es in der Praxis gerne angewendet wird, um Tabelleninhalte optisch leichter erfassbar zu machen.

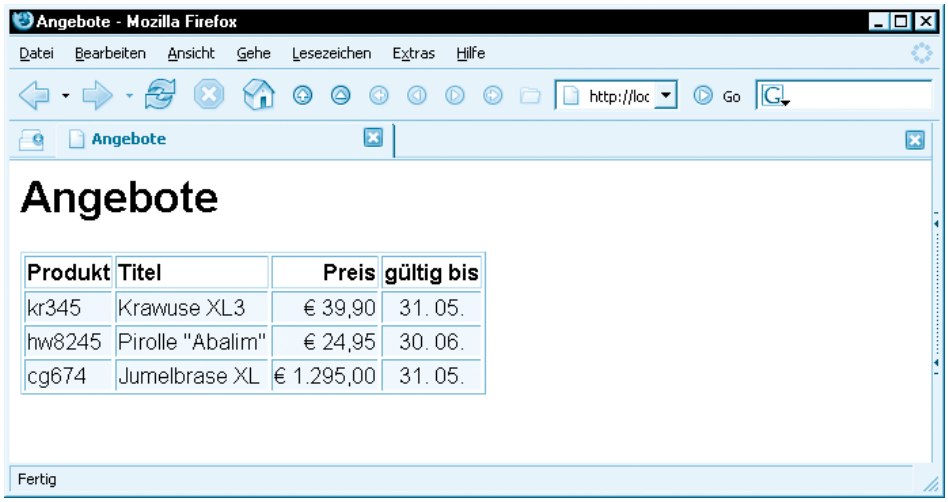


Abbildung 14.6: Ergebnis-Screenshot nach XSLT-Transformation

Dazu werden im XSLT-Stylesheet zu Beginn (vor den Template-Definitionen) die Variablen mit den Namen `dark` und `light` definiert. Als Wert erhalten sie eine geeignete CSS-Style-Angabe zur Hintergrundfarbe. In unserem Beispiel wird noch eine weitere Variable mit einem derartigen Wert definiert: die Variable `special`. Die damit definierte Hintergrundfarbe soll für Tabellenzeilen verwendet werden, in denen ein Produkt aufgelistet wird, dessen aktueller Preis ein Sonderangebot ist.

Innerhalb der `xsl:for-each`-Schleife, durch die alle `product`-Elemente für die HTML-Tabelle abgearbeitet werden, muss entschieden werden, welche Hintergrundfarbe die jeweils aktuelle Tabellenzeile erhält. Für den Wechsel zwischen hellerer und dunklerer Hintergrundfarbe kommt eine XPath-Funktion namens `position()` zum Einsatz:

```
<xsl:when test="position() mod 2 = 0">
```

Die Funktion `position()` liefert in einer Schleife über eine Reihe von Elementen den aktuellen Zähler, also das wievielte Element gerade abgearbeitet wird. Mit `mod 2 = 0` wird geprüft, ob eine Modulodivision (Restwert der Division) der aktuellen Zählerposition 0 ergibt. In diesem Fall wird im Beispiel die dunklere Hintergrundfarbe gesetzt. Für den anderen Fall, nämlich dass der Restwert 1 beträgt, wird der zugehörige `xsl:otherwise`-Zweig definiert. Dabei wird die hellere Hintergrundfarbe gesetzt. Die gesamte Unterscheidung steht jedoch ihrerseits in einem `xsl:otherwise`-Zweig. Denn vorher wird bereits mittels `xsl:when` abgefragt, ob das `offer`-Attribut im Element für die Preisangabe ein Sonderangebot (`offer="special"`) auszeichnet. In diesem Fall wird die spezielle Hintergrundfarbe angewendet.

Zwei weitere wichtige XPath-Funktionen, die in unserem Beispiel zum Einsatz kommen, sind `substring()` und `format-number()`. Mit Hilfe von `substring()` kann aus einer Zeichenkette eine Teilzeichenkette extrahiert werden. Im Beispiel verwenden wir die Funktion, um die Datumsangabe zur Preisgültigkeit anders zu formatieren als in den XML-Daten des Quellbaums. Dort stehen Werte wie 2006-05-31. Daraus soll die Angabe 31.05. werden.

Mit `substring(price/valid-until, 9, 2)` wird in diesem Beispiel die 31 extrahiert, also die Zeichenkette ab dem 9. Zeichen, und dann insgesamt zwei Zeichen. Mit `substring(price/valid-until, 6, 2)` wird auf gleiche Weise die 05 extrahiert. Die so extrahierten Werte lassen sich im Template für die Zielausgabe an der gewünschten Stelle einsetzen. Wenn Sie ähnliche Funktionen aus anderen Programmiersprachen wie JavaScript und/oder PHP kennen, müssen Sie beachten, dass die Zeichenkettenfunktionen von XPath nicht wie üblich 0-basiert arbeiten, sondern 1-basiert. Das erste Zeichen hat also die Indexnummer 1, das zweite die Indexnummer 2 usw.

Eine XPath-Funktion, die bei XSL-Transformationen ebenfalls gerne verwendet wird, ist die Funktion `format-number()`. Sie erlaubt es, Zahlenangaben wie etwa Preise in ein gewünschtes Format zu bringen. In unserem Beispiel finden wir folgende Anwendungen dieser Funktion:

```
format-number(price/value,'#.##0,00','german')
```

Zunächst wird der XPath-Pfad zu dem gewünschten Wert übergeben, in unserem Beispiel also das jeweilige value-Element unterhalb des `price`-Elements eines jeden `product`-Elements. Die zweite Angabe ist ein so genannter Formatstring. Im Beispiel wird festgelegt, dass alle Zahlen mit zwei Nachkommastellen dargestellt werden, dass das Dezimalzeichen ein Komma ist und dass außerdem Tausenderpunkte verwendet werden, wenn der Preis entsprechend hoch ist.

Eine 0 im Formatstring erzwingt eine Ziffer (im Beispiel wird also etwa ein Preis wie 95 Cent in das Format 0,95 gebracht. Würde der Formatstring `'0.000,00'` lauten, so würde ein Preis von 39 Euro und 90 Cent in der Form 0.039,90 dargestellt.

Ein Gatter (*#*) im Formatstring stellt eine mögliche Ziffer dar. Im Beispiel werden damit die Zehner-, die Hunderter- und die Tausenderstelle bezeichnet. Gatterzeichen können aber auch hinter dem Dezimalzeichen notiert werden. Dadurch lässt sich eine automatische Rundung definieren. So bewirkt beispielsweise `'###0,0###'`, angewendet auf 0.00035, die Ausgabe 0,0004.

Zusätzlich sollte die Funktion `format-number()` jedoch noch einen dritten Parameter erhalten. Dies ist der Name eines Formats, das zuvor definiert worden sein muss, und zwar mit Hilfe des XSLT-Elements `xsl:decimal-format`. In unserem Beispiel ist das Element gleich unterhalb vom Start-Tag des Dokumentelements `xsl:stylesheet` notiert. Es definiert einen Formatnamen sowie Angaben zum Dezimalzeichen sowie zum Tausendertrennzeichen.

Weitere XPath-Funktionen

Einige weitere nützliche XPath-Funktionen sollen zumindest noch erwähnt werden. Die Funktion `sum()` berechnet eine Summe von Zahlenwerten. In unserem Beispiel könnte z. B. am Ende der Ausgabetabelle notiert werden:

```
<p>Summe der Preise:
<xsl:value-of select="sum(product/price/value)" /></p>
```

Damit würde die Summe über die Werte aller `value`-Elemente gebildet, die innerhalb eines `product`-Elements und dort innerhalb eines `price`-Elements vorkommen.

Eine weitere praktische Funktion ist `generate-id()`. Damit lassen sich dokumentweit eindeutige Werte im Zielbaum erzeugen, was etwa für `id`- oder `name`-Attribute erforderlich sein kann. Angenommen, in einer `xsl:for-each`-Schleife werden XML-Quellbauminhalte abgearbeitet, aus denen in den HTML-Zieldaten Listeneinträge mit `id`-Attribut werden sollen. So etwas kann z. B. für JavaScript-Anwendungen sinnvoll sein. Beispiel:

```
<xsl:for-each select="member">
   <li id="{generate-id()}">
      <xsl:value-of select="given_name"/>
      <xsl:value-of select="family_name"/>
   </li>
</xsl:for-each>
```

In der HTML-Ausgabe des XSLT-Prozessors Saxon erscheint dann beispielsweise Code wie dieser:

```
<li id="#d2e3">Anna Lyse</li>
<li id="#d2e22">Reiner Wahnsinn</li>
<li id="#d2e41">Mira Mare</li>
```

Die Funktion `substring()` haben wir bereits kennen gelernt. Wenn aber nur überprüft werden soll, ob eine Zeichenkette mit einer bestimmten Teilzeichenkette beginnt, ist die Funktion `starts-with()` praktischer. Ein Beispiel:

```
<xsl:if test="starts-with(name/given_name, 'M')">
```

Damit wird überprüft, ob der Inhalt eines aktuellen `given_name`-Elements mit M beginnt. Die Funktion ist also sehr gut geeignet, um Daten zu selektieren.

Auch für Zahlen stehen noch weitere Funktionen bereit: `floor(Wert)` rundet den in `Wert` gespeicherten Zahlenwert zur Basisganzzahl ab, `ceiling(Wert)` rundet zur nächsthöheren Ganzzahl auf und `round(Wert)` rundet kaufmännisch. Beispiel:

```
<xsl:variable name="price" select="/products/product/price/value" />
<xsl:value-of select="floor($price)" />
```

Voraussetzung ist allerdings, dass der Wert als Zahl interpretiert werden kann. Wenn in den XML-Daten also z. B. 39,90 steht, schlägt die Anwendung von Funktionen wie `floor()` fehl, da das Komma nicht als Dezimalzeichen erkannt wird.

14.7.6 XPath-Adressierung bei der Transformation

Die korrekte XPath-Adressierung von Knoten und Inhalten im XML-Quellbaum ist die Voraussetzung für eine erfolgreiche Datenkonvertierung. Deshalb ist es wichtig, die Adressierungsmöglichkeiten genau zu kennen und gezielt anwenden zu können.

XPath-Adressierung wird innerhalb von XSLT beispielsweise benötigt bei:

- `<xsl:template match="[XPath-Adressierung]">`,
- `<xsl:apply-templates select="[XPath-Adressierung]" />`,
- `<xsl:for-each select="[XPath-Adressierung]">`,

- `<xsl:if test="[XPath-Adressierung][...]">`,
- `<xsl:when test="[XPath-Adressierung][...]">`,
- `<xsl:value-of select="[XPath-Adressierung]">`.

XML-Dokumente bestehen aus mehr oder weniger tief verschachtelten Elementen, die außerdem Attribute sowie Textinhalt enthalten können. Elemente, Attribute und Textinhalte stellen dabei aus Sicht von XPath so genannte **Knoten** dar. Knoten sind also identifizierbare Einheiten eines XML-Dokuments.

Damit solche Knoten adressierbar sind, muss jedoch auch ihre Beziehung untereinander berücksichtigt werden. So gibt es aus Sicht eines aktuell gegebenen Knotens beispielsweise einen Elternknoten (z. B. das Element, innerhalb dem ein aktuelles Element vorkommt), Kindknoten (Elemente oder Textinhalte, die innerhalb eines aktuellen Elements vorkommen), vorhergehende und nachfolgende Geschwisterknoten auf gleicher Ebene (wie etwa `li`-Elemente in einer HTML-Liste) usw.

Absolute und relative Adressierung

Durch verschachtelte Elemente entstehen, ähnlich wie in einer Verzeichnisstruktur auf einem Laufwerk, Pfade zu bestimmten Elementen. In XPath werden die Pfadelemente denn auch wie unter Linux/Unix die Verzeichnisse einer Pfadangabe durch Schrägstriche voneinander getrennt. Und genauso wie bei Adressierungen in einer Verzeichnisstruktur sind absolute und relative Pfadangaben möglich. Eine absolute XPath-Adressierung eines Elements ist beispielsweise:

```
/book/content/keywords
```

Eine relative Adressierung aus Sicht des keywords-Elements wäre etwa:

```
../description
```

Verkürzte und ausführliche Notation

Neben dieser üblicheren Notation, die wir auch in unseren bisherigen Beispielen ausschließlich verwendet haben, gibt es jedoch auch eine ausführlichere Notation in XPath. Dabei werden vor allem die Beziehungen der Pfadbestandteile explizit angegeben. Eine absolute Adressierung in dieser Notation wäre etwa:

```
./child::book/child::content/child::keywords
```

Eine relative Adressierung aus Sicht des keywords-Elements wäre etwa:

```
parent::content/child::description
```

Adressierung von Attributen

In der verkürzten Version werden Attribute von Elementen durch ein vorangestelltes @-Zeichen adressiert. Beispiel:

```
member/@nr
```

Damit wird der Wert des `nr`-Attributs eines member-Elements adressiert, d.h., in den XML-Quelldaten wird aus einem Markup wie `<member nr="7362">` der Wert 7362 ermittelt.

In der ausführlicheren Schreibweise sieht das wie folgt aus:

```
member/attribute::nr
```

Erweiterte Features bei der Adressierung

XPath bietet eine ganze Reihe speziellerer Notationen an, um Daten aus dem XML-Quellbaum gezielt zu filtern und auszuwählen. Vor allem in Verbindung mit XPath-Funktionen und Ausdrücken sind vielfältige Anwendungen möglich. Nachfolgende Tabelle zeigt einige Beispiele.

Beispiel	Bedeutung
item[position() = 3]	Wählt das dritte Kindelement des Namens `item` aus.
item[3]	Wählt das dritte Kindelement des Namens `item` aus.
item[last()]	Wählt das letzte Kindelement des Namens `item` aus.
item[last()-1]	Wählt das vorletzte Kindelement des Namens `item` aus.
item[@type='pdf']	Wählt das Kindelement `item` aus, jedoch nur, wenn es ein Attribut `type="pdf"` enthält.
item[@type and @id]	Wählt das Kindelement `item` aus, jedoch nur, wenn es sowohl ein Attribut `type=` als auch ein Attribut `id=` enthält (egal mit welchen Werten).
.	Wählt den aktuellen Knoten aus.

Tabelle 14.3: Weitere Möglichkeiten der XPath-Adressierung

14.8 PHP und XML

XML hat die Aufgabe, Daten semantisch sinnvoll zu beschreiben. Doch der Einsatzzweck von XML ist nicht nur die Dokumentation von Daten, sondern vor allem auch deren Verarbeitung in Anwendungen. Was Webanwendungen betrifft, so konzentrieren wir uns in diesem Buch auf PHP. Folglich fehlt noch die Verbindung, also wie Sie mit Hilfe von PHP XML-basierte Daten verarbeiten können.

PHP bietet dazu mehrere Wege an. Der Grund dafür ist vor allem, dass die Webentwicklerbedürfnisse gegenüber der XML-Datenverarbeitung sehr unterschiedlich sind. Einige wollen »nur mal eben« eine kleine XML-Datei mit Konfigurationsdaten einlesen. Andere wiederum wollen ein komplettes, XML-basiertes Content Management System errichten. Einige wollen XML-Dokumente auslesen und neue erzeugen, andere nur auslesen. Manche haben es mit XML-Dokumenten zu tun, die nur ein paar Kilobyte groß sind, andere mit welchen, deren Umfang zig Megabyte beträgt.

Bevor Sie sich für einen der möglichen, von PHP angebotenen Wege zur XML-Datenverarbeitung entscheiden, sollten Sie daher die Vor- und Nachteile der jeweiligen Wege kennen:

Weg 1: XML-Verarbeitung mit Expat-Parser

- Vorteil: auch für sehr umfangreiche XML-Dokumente geeignet.
- Vorteil: steht standardmäßig zur Verfügung.
- Nachteil: kann nur XML lesen, kein XML erzeugen.
- Nachteil: arbeitet nur ereignisorientiert (nur SAX-Schnittstelle, keine DOM-Schnittstelle)

Die XML-Verarbeitung über den Expat-Parser ist die älteste XML-Verarbeitungsvariante für PHP. Seit der Version 3.0.6. steht die Variante als PHP-Erweiterung zur Verfügung. Die entsprechenden PHP-Funktionen stehen standardmäßig zur Verfügung und verwenden die Expat-Bibliothek von Apache.

Weg 2: XML-Verarbeitung mit DOM bzw. DOM XML

- Vorteil: erlaubt den Direktzugriff auf die gesamte XML-Datenstruktur (DOM-Schnittstelle).
- Vorteil: kann XML lesen und erzeugen.
- Nachteil: nicht für sehr umfangreiche XML-Dokumente geeignet.
- Nachteil: keine vollständige Unterstützung eines bestimmten DOM-Standards garantiert.

Wenn Sie mit JavaScript und darüber mit dem Document Object Model (DOM) vertraut sind, werden Sie mit dieser Art der XML-Verarbeitung am schnellsten klar kommen. Die üblichen DOM-Methoden von `getElementById()`, `createElement()` usw. stehen zur Verfügung.

Seit PHP 5.x ist die DOM-Erweiterung fester Bestandteil von PHP. Für PHP 4.x steht nur die ältere DOM-XML-Erweiterung zur Verfügung, für die zusätzlich der XML-Parser libxml installiert werden musste. Die DOM-XML-Erweiterung wurde außerdem mehrfach überarbeitet und kam über das Stadium einer als experimentell eingestuften Erweiterung nie hinaus. Wenn Sie viel mit PHP und XML arbeiten, ist dies allein schon ein Grund, nicht mehr PHP 4.x, sondern PHP 5.x einzusetzen.

Weg 3: XML-Verarbeitung mit der SimpleXML-Klasse

- Vorteil: erlaubt den Direktzugriff auf die gesamte XML-Datenstruktur (DOM-ähnlich, jedoch keine DOM-Implementierung).
- Vorteil: kann XML lesen und erzeugen (erzeugen allerdings mit einigen Einschränkungen).
- Nachteil: nicht für sehr umfangreiche XML-Dokumente geeignet.
- Nachteil: keine echte DOM-Unterstützung.

SimpleXML steht ebenfalls seit PHP 5.x zur Verfügung und ist dort standardmäßig integriert. Diese Schnittstelle ist für Entwickler gedacht, die seltener mit XML in Berührung kommen und deren Aufgabe vornehmlich darin besteht, Daten aus vergleichsweise einfachen und überschaubaren XML-Dokumenten auszulesen oder in solche Dokumente zurückzuschreiben.

Weg 4: XML-Verarbeitung mit XMLReader/xmlwriter

- Vorteil: auch für sehr umfangreiche XML-Dokumente geeignet.
- Vorteil: kann XML lesen (XMLReader) und erzeugen (xmlwriter).
- Nachteil: erst seit PHP 5.1 standardmäßig verfügbar.
- Nachteil: unterstützt nicht die DOM-Schnittstelle, sondern arbeitet ähnlich wie bei der SAX-Schnittstelle ereignisorientiert.

Diese beiden neueren PHP-Klassen orientieren sich stark an den gleichnamigen .NET-Implementierungen von Microsoft. Es handelt sich am ehesten um eine Alternative zur Verarbeitung über den Expat-Parser, mit dem entscheidenden Vorteil, dass XML-Daten über die xmlwriter-Erweiterung auch erzeugt werden können.

Fazit

Die Vielfalt möglicher Verarbeitungswege sorgt derzeit mehr für Verwirrung, als dass sie wirklich jemandem nutzt. Das Problem ist jedoch kein PHP-eigenes. Unterschiedliche Anforderungen an das Parsen von XML-Daten sowie die hektische Weiterentwicklung der Technologien rund um XML sind hauptsächlich verantwortlich dafür.

In den nachfolgenden Abschnitten werden wir einige der Verarbeitungswege an kleinen Beispielen vorstellen. Dabei behandeln wir zunächst den einfachsten Weg, die Verarbeitung mit SimpleXML, anschließend den Weg über die DOM-Verarbeitung und schließlich die klassische Variante über den Expat-Parser.

14.8.1 XML-Verarbeitung mit der SimpleXML-Klasse

Zunächst das Listing der XML-Datei, die wir verarbeiten möchten:

```
<?xml version="1.0"?>
<members>
   <member nr="19341" position="none">
      <given_name>Anna</given_name>
      <family_name>Lyse</family_name>
      <member_since>
         <yyyy>2003</yyyy><mm>10</mm><dd>18</dd>
      </member_since>
   </member>
   <member nr="11294" position="committee">
      <given_name>Reiner</given_name>
      <family_name>Wahnsinn</family_name>
      <member_since>
```

```
            <yyyy>1998</yyyy><mm>02</mm><dd>23</dd>
         </member_since>
      </member>
      <member nr="20845" position="none">
         <given_name>Mira</given_name>
         <family_name>Mare</family_name>
         <member_since>
            <yyyy>2005</yyyy><mm>06</mm><dd>01</dd>
         </member_since>
      </member>
   </members>
```

Die Aufgabenstellung soll darin bestehen, alle Daten über das Mitglied Reiner Wahnsinn zu ermitteln und auszugeben. Wir kennen den Aufbau der XML-Datei, doch wir wissen nicht, als wievieltes `member`-Element dasjenige für Rainer Wahnsinn enthalten ist. So bleibt nur, die einzelnen `member`-Elemente in einer Schleife abzuarbeiten und jeweils herauszufinden, ob die Kindelemente `given_name` und `family_name` Übereinstimmungen mit Reiner bzw. Wahnsinn liefern.

Der PHP-Code lautet:

```
<?php
if(file_exists("members.xml"))
   $xml = simplexml_load_file("members.xml");
else
   exit("members.xml nicht gefunden!");
foreach ($xml->children() as $member) {
   if($member->given_name == "Reiner" and
      $member->family_name == "Wahnsinn") {
         $member_nr = $member['nr'];
         $member_name = $member->given_name." ".
                        $member->family_name;
         $member_since = $member->member_since->dd.".".
                         $member->member_since->mm.".".
                         $member->member_since->yyyy;
         echo "<p>Name: ", $member_name, "<br>";
         echo "Mitgliedsnummer: ", $member_nr, "<br>";
         echo "Mitglied seit: ", $member_since, "</p>";
   }
}
?>
```

Um eine XML-Datei zum Auslesen zu öffnen, wird die Funktion `simplexml_load_file()` verwendet. Als Parameter wird ihr der Pfadname zur gewünschten XML-Datei übergeben.

Falls die XML-Daten nicht als Datei vorliegen, sondern bereits in einer PHP-Variablen (z.B. weil sie zuvor bereits mit einer Standardfunktion zum Lesen von Dateien eingelesen wurde oder weil die XML-Daten auf anderem Wege empfangen wurden, etwa durch eine GET-Anfrage an einen entfernten Webserver), können Sie stattdessen die Funktion `simplexml_load_string()` verwenden.

Funktion	simplexml_load_file()
Aufgabe	Liest eine XML-Datei in ein SimpleXML-Objekt ein. Der Funktionsaufruf sollte einer Variablen zugewiesen werden.
Parameter	1. Die XML-Datei, wenn nötig mit absolutem Pfadnamen oder Pfadnamen aus Sicht des PHP-Scripts 2. (optional) Name einer eigenen Klasse, die von der SimpleXML-Klasse abgeleitet sein muss 3. (optional) Optionen für den Parser, trennen durch Senkrechtstriche Mögliche Angaben siehe *http://de.php.net/manual/de/ref.libxml.php#libxml.constants*
Rückgabewert	Die XML-Daten als SimpleXML-Objekt.

Tabelle 14.4: Steckbrief Funktion »simplexml_load_file()«

Funktion	simplexml_load_string()
Aufgabe	Liest XML-Daten, die als Zeichenkette vorliegen, in ein SimpleXML-Objekt ein. Der Funktionsaufruf sollte einer Variablen zugewiesen werden.
Parameter	1. Der Variablenname mit den XML-Daten 2. (optional) Name einer eigenen Klasse, die von der SimpleXML-Klasse abgeleitet sein muss 3. (optional) Optionen für den Parser, trennen durch Senkrechtstriche Mögliche Angaben siehe *http://de.php.net/manual/de/ref.libxml.php#libxml.constants*
Rückgabewert	Die XML-Daten als SimpleXML-Objekt.

Tabelle 14.5: Steckbrief Funktion »simplexml_load_string()«

Die Daten werden im Beispiel in eine Variable namens `$xml` eingelesen. Anschließend ist in `$xml` ein Objekt mit den XML-Daten gespeichert. Dazu ist es wichtig zu wissen, dass SimpleXML die gesamten XML-Daten als verschachtelte Objektstruktur abbildet. Über die Objektvariable `$xml` können wir im Beispiel auf die Daten und übrigen Methoden des SimpleXML-Objekts zugreifen.

Sehr praktisch ist die Repräsentation in Form verschachtelter Objekte, um Elemente in einer verschachtelten Elementstruktur direkt anzusprechen. Angenommen, die XML-Daten enthalten eine Elementfolge `<a><b><c></c></b></a>`. Über unsere `$xml`-Objektvariable könnten wir auf den Inhalt des `c`-Elements ganz einfach mit `$xml->a->b->c` zugreifen. Der Grund ist, dass jedes Kindelement wieder ein Unterobjekt des in `$xml` gespeicherten SimpleXML-Objekts ist.

In unserem Beispiel haben wir es jedoch mit einer unbekannten Anzahl von Kindelementen namens `member` zu tun. Eines davon speichert die Daten zu Reiner Wahnsinn, doch wir wissen nicht, welches. Deshalb durchlaufen wir die Kindelemente in einer foreach-Schleife. Die Gesamtheit der Kindelemente sprechen wir über die Methode `children()` an. Diese liefert alle Kindelemente aus Sicht des Objekts, auf das sie angewendet wird. Da in unserem Beispiel `$xml` den gesamten XML-Baum ab dem Dokumentelement enthält, liefert `$xml->children()` einen Array aus XMLSimple-Objekten für die einzelnen `member`-Elemente. Auf jedes dieser Objekte können wir innerhalb der `foreach`-Schleife über den Namen `$member` zugreifen.

Funktion	$SimpleXMLObjektVariable->children()
Aufgabe	Liefert einen Array aus SimpleXML-Objekten für alle Kindelemente aus Sicht des Elements, das im SimpleXML-Objekt `$SimpleXMLObjektVariable` gespeichert ist.
Parameter	1. (optional) ein Namensraumpräfix, sofern damit gearbeitet wird
Rückgabewert	Array aus SimpleXML-Objekten.

Tabelle 14.6: Steckbrief Funktion »children()«

In unserem Beispiel genügt dies bereits, um an die gewünschten Daten heranzukommen. Interessant an dem Beispiel ist noch, wie auf Attributwerte zugegriffen wird. Mit `$member['nr']` wird jeweils der Wert des `nr`-Attributs aus dem einleitenden `<member>`-Tag ermittelt. Die Attributwerte für ein Element, das in einem SimpleXML-Objekt gespeichert ist, stehen also als assoziativer Array zur Verfügung. Die Attributnamen sind dabei die Array-Schlüssel und die Attributwerte die Array-Werte.

Wenn Sie nicht wissen, welche Attribute ein Element hat, können Sie analog zur `children()`-Methode die `attributes()`-Methode anwenden. Beispiel:

```
foreach ($member->attributes() as $key => $value) {
   if($key == "nr")
      $member_nr = $value;
```

Funktion	$SimpleXMLObjektVariable->attributes()
Aufgabe	Liefert einen Array aus SimpleXML-Objekten für alle Attribute aus Sicht des Elements, das im SimpleXML-Objekt `$SimpleXMLObjektVariable` gespeichert ist.
Parameter	1. (optional) Attributwert
Rückgabewert	Assoziativer Array aus Attributen (Attributnamen als Schlüssel, Attributwerte als Werte)

Tabelle 14.7: Steckbrief Funktion »attributes()«

Eine weitere effiziente Möglichkeit, auf bestimmte XML-Daten gezielt zuzugreifen, ist der Zugriff über XPath-Syntax. Hierzu dient die `xpath()`-Methode. Angenommen, die XML-Daten enthalten eine Elementfolge `<a><b><c></c></b></a>`. Über unsere `$xml`-Objektvariable könnten wir auf den Inhalt des `c`-Elements dann auch mit `$xml->xpath("/a/b/c ")` zugreifen. Näheres zur XPath-Syntax siehe ***Abschnitt 14.7.6***.

Funktion	$SimpleXMLObjektVariable->xpath()
Aufgabe	Liefert einen Array aus SimpleXML-Objekten mit gefundenen XML-Knoten, die im angegebenen XPath-Pfad liegen.
Parameter	1. Der XPath-Pfad
Rückgabewert	Array aus SimpleXML-Objekten

Tabelle 14.8: Steckbrief Funktion »xpath()«

Dass mit SimpleXML auch das Erzeugen und Schreiben von XML-Daten möglich ist, wurde bereits angesprochen. Der Schlüssel hierzu ist die Methode `asXML()`. Allerdings setzt diese bereits ein existierendes SimpleXML-Objekt voraus. Möglichkeiten wie bei DOM, um Elemente, Attribute und andere XML-Knoten zu erzeugen und dann an gewünschter Stelle in einer XML-Struktur einzuhängen, sind mit dieser Methode nicht realisierbar. Sinnvoll ist die Methode, wenn einzelne Daten geändert werden sollen. Dazu noch mal ein Beispiel:

```
<?php
if(file_exists("members.xml"))
   $xml = simplexml_load_file("members.xml");
else
   exit("members.xml nicht gefunden!");
foreach ($xml->children() as $member) {
   if($member->given_name == "Reiner" and
      $member->family_name == "Wahnsinn")
         $member->given_name = "Rainer";
}
$xml->asXML("members.xml");
?>
```

Das Beispiel sucht wie schon zuvor nach Reiner Wahnsinn. Bei gefundenen Daten wird jedoch »Reiner« durch »Rainer« ersetzt. Am Ende wird der gesamte XML-Baum, der ja in `$xml` gespeichert ist, in die Datei *members.xml* zurückgeschrieben.

Funktion	**$SimpleXMLObjektVariable->asXML()**
Aufgabe	Erzeugt XML-Daten als Zeichenkette aus einem SimpleXML-Objekt.
Parameter	1. (optional) Dateiname (gegebenenfalls mit Pfad) einer XML-Datei. Wenn angegeben, werden die XML-Daten in die Datei geschrieben. Wenn weggelassen, werden die XML-Daten als Zeichenkette zurückgegeben.
Rückgabewert	Die XML-Daten als Zeichenkette

Tabelle 14.9: Steckbrief Funktion »asXML()«

Wenn Sie die XML-Daten nicht oder nicht gleich in eine XML-Datei zurückschreiben wollen, können Sie die Methode auch ohne Parameter aufrufen. In diesem Fall sollten Sie den Aufruf jedoch einer Variablen zuweisen. Darin sind anschließend die XML-Daten als Zeichenkette gespeichert.

14.8.2 XML-Verarbeitung mit der DOM-Klasse (libXML)

Zum besseren Vergleich wollen wir wieder die gleichen XML-Daten verarbeiten:

```
<?xml version="1.0"?>
<members>
   <member nr="19341" position="none">
      <given_name>Anna</given_name>
      <family_name>Lyse</family_name>
```

```
      <member_since>
         <yyyy>2003</yyyy><mm>10</mm><dd>18</dd>
      </member_since>
   </member>
   <member nr="11294" position="committee">
      <given_name>Reiner</given_name>
      <family_name>Wahnsinn</family_name>
      <member_since>
         <yyyy>1998</yyyy><mm>02</mm><dd>23</dd>
      </member_since>
   </member>
   <member nr="20845" position="none">
      <given_name>Mira</given_name>
      <family_name>Mare</family_name>
      <member_since>
         <yyyy>2005</yyyy><mm>06</mm><dd>01</dd>
      </member_since>
   </member>
</members>
```

Auch die Aufgabenstellung soll die gleiche bleiben: alle Daten über das Mitglied Reiner Wahnsinn zu ermitteln und auszugeben.

Mit der DOM-Klasse von PHP 5.x können wir diese Aufgabenstellung wie folgt lösen:

```
<?php
if(file_exists("members.xml")) {
$dom = new DOMDocument("1.0", "UTF-8");
if(!$dom)
      exit("Konnte kein DOM-Objekt erzeugen!");
if(!$dom->load("members.xml"))
      exit("members.xml konnte nicht eingelesen werden!");
$dom->preserveWhiteSpace = false;
   //$path = new DOMXPath($dom);

}
else
   exit("members.xml nicht gefunden!");

$member_elems = $dom->getElementsByTagName('member');

for($i = 1; $i <= $member_elems->length; $i++) {
   $given_name = get_node_value($dom,
                 "/members/member[".$i."]/given_name");
   $family_name = get_node_value($dom,
                  "/members/member[".$i."]/family_name");
   if($given_name == "Reiner" and $family_name == "Wahnsinn") {
      $member_nr = get_node_value($dom,
                 "/members/member[".$i."]/@nr");
      $member_since =
         get_node_value($dom,
         "/members/member[".$i."]/member_since/dd") .
         get_node_value($dom,
```

```
            "/members/member[".$i."]/member_since/mm") .
            get_node_value($dom,
            "/members/member[".$i."]/member_since/yyyy");
         echo "<p>Name: $given_name $family_name<br>";
         echo "Mitgliedsnummer: ", $member_nr, "<br>";
         echo "Mitglied seit: ", $member_since, "</p>";
      }
   }

   function get_node($dom, $xpath) {
      $path = new DOMXPath($dom);
      $nodes = @$path->query($xpath);
      if(!is_object($nodes))
         return(false);
      else
         return($nodes->item(0));
   }

   function get_node_value($dom, $xpath) {
      $path = new DOMXPath($dom);
      $nodes = @$path->query($xpath);
      if(!is_object($nodes))
         return(false);
      else
         return($nodes->item(0)->nodeValue);
   }
   ?>
```

Wie unschwer erkennbar ist, ist der Code-Aufwand zum Erreichen der gleichen Lösung wie zuvor deutlich höher. Das ist der Preis für die umfangreicheren Möglichkeiten bei der Verarbeitung mit der DOM-Klasse.

Unser Beispiel erzeugt zunächst mit Hilfe der Konstruktorfunktion `new DOMDocument()` ein neues, leeres Objekt für den DOM-Zugriff auf eine XML-Baumstruktur. Das Objekt wird in der Variablen `$dom` gespeichert. Als Parameter werden der Konstruktorfunktion die gewünschte XML-Version (derzeit 1.0 oder 1.1) sowie die Angabe zur Zeichenkodierung übergeben. Beide Parameter sollten mit denen übereinstimmen, die in der XML-Deklarationsanweisung `<?xml ...?>` der zu verarbeitenden XML-Daten notiert sind.

Das so erzeugte DOM-Objekt verfügt über eine Methode namens `load()` zum Lesen von XML-Daten. Mit Hilfe von `$dom->load("members.xml")` lesen wir also die Datei ***members.xml*** ein und füllen das zuvor erzeugte DOM-Objekt mit den Daten dieser Datei.

Funktionen	**$DOM_Objektvariable->load()** **$DOM_Objektvariable->loadXML()**
Aufgabe	load() liest eine XML-Datei in ein DOM-Objekt ein. loadXML() liest eine Zeichenkette mit XML-Daten in ein DOM-Objekt.

Tabelle 14.10: Steckbrief DOM-Methoden »load ()« und »loadXML()«

Parameter	1. Bei load() die XML-Datei, wenn nötig mit absolutem Pfadnamen oder Pfadnamen aus Sicht des PHP-Scripts 2. Bei loadXML() die Zeichenkette mit dem XML-Code 3. (optional) Optionen für den Parser, trennen durch Senkrechtstriche Mögliche Angaben siehe *http://de.php.net/manual/de/ref.libxml.php#libxml.constants*
Rückgabewert	`true` bei Erfolg, `false` im Fehlerfall

Tabelle 14.10: Steckbrief DOM-Methoden »load ()« und »loadXML()« (Fortsetzung)

Die nächste erwähnenswerte Anweisung in unserem Beispiel ist die Zeile:

```
$member_elems = $dom->getElementsByTagName('member');
```

Auf das DOM-Objekt, das in der Variablen `$dom` gespeichert ist, sind die aus JavaScript/DOM bekannten Funktionen für den schnellen Elementzugriff anwendbar. Im Beispiel werden auf diese Weise alle `member`-Elemente der XML-Daten ermittelt.

Funktionen	**$DOM_Objektvariable->getElementById()** **$DOM_Objektvariable->getElementsByTagName()** **$DOM_Objektvariable->getElementsByTagNameNS()**
Aufgabe	`getElementById()` liefert ein bestimmtes Element mit dokumentweit eindeutigem id-Attribut. `getElementsByTagName()` liefert eine Liste aller Elemente mit einem bestimmten Elementnamen. Die Namespace-Variante berücksichtigt den Namensraum.
Parameter	1. Bei `getElementByTagId()` der Wert des id-Attributs, bei `getElementsByTagName()` der Elementname
Rückgabewert	Bei `getElementById()` ein DOM-Elementobjekt, bei `getElementsByTagName()` eine DOM-Element-Objektliste.

Tabelle 14.11: Steckbrief DOM-Methoden »getElementById()« und »getElementsByTagName()«

Auf ein DOM-Elementobjekt, wie es `getElementById()` im Erfolgsfall liefert, sind die bekannten DOM-Eigenschaften und -Methoden anwendbar (siehe Tabelle weiter unten). Bei DOM-Element-Objektlisten, wie sie `getElementsByTagName()` zurückgibt, liegt der Fall etwas schwieriger. Um die damit erzeugte Ergebnismenge zu verarbeiten, ist in der Regel eine Schleife erforderlich, so auch in unserem Beispiel. Die Schleifenbedingung lautet dort:

```
for($i = 1; $i <= $member_elems->length; $i++)
```

Die Variable `$member_elems` enthält die DOM-Element-Objektliste. Eine solche Liste kennt nur die Eigenschaft `length` (Anzahl der Listenelemente). Diese Eigenschaft nutzen wir für die Konstruktion einer klassischen `for`-Zählschleife. Über die Zählervariable `$i` ist dann der Zugriff auf die einzelnen `member`-Elemente und deren Kindelemente möglich.

Für unsere Aufgabenstellung müssen wir zunächst gezielt auf die Werte der Kindelemente `given_name` und `family_name` zugreifen, um das `member`-Element für Reiner Wahnsinn herauszufiltern. Dazu ist eine Adressierung dieser Elemente erforderlich, was mit Hilfe von

XPath-Syntax möglich ist. Da das Handling für XPath-Zugriffe innerhalb der DOM-Klasse von PHP etwas umständlich ist, haben wir dafür zwei allgemeine Funktionen geschrieben, die am Ende des Beispiel-Listings notiert sind: `get_node()` und `get_node_value()`. Mit `get_node()` kann auf einen Knoten als solchen zugegriffen werden, also auf ein Element, ein Attribut usw. Zurückgeliefert wird dann ein DOM-Knotenobjekt, welches im Fall von Elementen auch gleichzeitig ein DOM-Elementobjekt ist. Mit `get_node_value()` kann dagegen gezielt auf den Inhalt eines Knotens zugegriffen werden, was bei adressierten Elementen deren Inhalt ist und bei adressierten Attributen deren Wertzuweisung.

Zum besseren Verständnis lohnt es sich, den Inhalt der beiden ähnlich aufgebauten Funktionen nachzuvollziehen. Beide Funktionen erwarten zwei Parameter, nämlich eine DOM-Objektvariable, mit der sie arbeiten können (in unserem Beispiel können wir also `$dom` übergeben), und als zweiten Parameter eine Zeichenkette, die eine Adressierung gemäß XPath-Syntax darstellt.

Für XPath-Zugriffe muss zunächst ein neues Objekt erzeugt werden. Dies tun die beiden Funktionen mit der Zeile:

```
$path = new DOMXPath($dom);
```

Die Konstruktorfunktion für ein solches Objekt, `DOMXPath()`, erwartet als Parameter die Objektvariable der DOM-Klassen-Instanz. In der Variablen `$path` wird dann ein XPath-Objekt gespeichert.

Die wohl zentrale Methode des XPath-Objekts ist die Methode `query()`. Ihr wird als Parameter eine Zeichenkette übergeben, die eine XPath-Adressierung darstellt. In unserem Beispiel wird die Variable übergeben, die den Funktionen `get_node ()` bzw. `get_node_value()` für diesen Zweck übergeben wurde.

Funktion	**$XPAth_Objektvariable->query()**
Aufgabe	Liefert eine Liste von DOM-Knoten, die der XPath-Adressierung entsprechen.
Parameter	1. die gewünschte XPath-Adressierung als Zeichenkette
Rückgabewert	Eine DOM-Knoten-Objektliste

Tabelle 14.12: Steckbrief XPath-Methode »query()«

Der Rückgabewert der `query()`-Methode ist ein Objekt mit einer Liste von gefundenen DOM-Knoten, die auf die Adressierung passen. In unseren beiden Beispielfunktionen setzen wir voraus, dass nach einem nur einmal vorhandenen Knoten gesucht wird bzw. dass die XPath-Adressierung so aussieht ist, dass die Ergebnisliste nur einen Knoten enthält. Auf diesen ersten Ergebnisknoten greifen die Funktionen mit der Objektmethode `item()` zu. Durch `item(0)` wird der erste Knoten der Ergebnisliste angesprochen. Im Fall von `get_node()` wird dieser Knoten zurückgegeben. Im Fall von `get_node_value()` wird auf die Knoteneigenschaft `nodeValue` zugegriffen, um den Wert eines Knotens (Elementinhalt, Attributwert usw.) zurückzugeben.

Mit `$given_name = get_node_value($dom, "/members/member[".$i."]/given_name")` können wir innerhalb der `for`-Schleife unseres Beispiels per XPath-Syntax auf den Inhalt des Elements `given_name` jedes einzelnen `member`-Elements zugreifen. Auf gleiche Weise ermittelt wir den Inhalt von `family_name`. Wird die Bedingung in der Abfrage `if($given_name == "Reiner" and $family_name == "Wahnsinn")` erfüllt, werden die restlichen Daten zum Eintrag ermittelt und ausgegeben.

Methoden und Eigenschaften für DOM-Knotenobjekte

Ein DOM-Knotenobjekt erhalten Sie wie beschrieben beispielsweise durch Aufruf von `getElementById()` oder durch Adressierung mit XPath. Auf solche Objekte sind zahlreiche DOM-Methoden und DOM-Eigenschaften anwendbar, die das Auslesen, Ändern und Erzeugen von XML-Daten ermöglichen.

Nachfolgende Tabellen listen wichtige Methoden und Eigenschaften auf, die auf solche Objekte anwendbar sind.

Methode	Anwendbar auf	Bedeutung/Beispiel

appendChild()	Alle Knoten, die Unterknoten haben können	Hängt einen Knoten in die XML-Struktur ein.
		`$node = $dom->createElement ("example");`
		`$newnode = $dom->appendChild($node);`

cloneNode()	Alle Knoten	Erzeugt eine Kopie eines Knotens.
		`$copy = $elem_node->cloneNode(true);`
		`true bedeutet: rekursiv kopieren.`

getAttribute()	Elementknoten	Ermittelt den Wert eines Attributs. `echo $elem_node->getAttribute("nr");`
getAttribute-Node()	Elementknoten	Ermittelt einen Attributknoten. `$nr = $elem_node->getAttributeNode("nr");`
hasAttribute()	Elementknoten	Prüft, ob ein Attribut gesetzt ist. `if($elem_node->hasAttribute ("nr"))`
hasAttributes()	Alle Knoten	Prüft, ob der Knoten Attribute hat. `if($node->hasAttributes())`
hasChildNodes()	Alle Knoten	Prüft, ob der Knoten Kindknoten hat. `if($node->hasChildNodes())`

insertBefore()	Alle Knoten	Fügt einen Knoten vor dem aktuellen ein.
		`$element = $dom->createElement('newnode');`
		`$elem_node->insertBefore($element);`

removeAttribute()	Elementknoten	Entfernt ein Attribut. `$elem_node->removeAttribute("nr");`
removeChild()	Alle Knoten mit Kindknoten	Entfernt einen Kindknoten. `$elem_node->removeChild($child_elem_node);`

Tabelle 14.13: Wichtige Methoden für DOM-Knotenobjekte

Methode	Anwendbar auf	Bedeutung/Beispiel

replaceChild()	Alle Knoten mit Kindknoten	Ersetzt einen Kindknoten durch einen anderen.
		`$element = $dom->createElement('newnode');`
		`$elem_node->replaceChild($child_elem_node, $element);`

setAttribute()	Elementknoten	Fügt ein Attribut hinzu. `$elem_node->setAttribute("nr", "49395");`

Tabelle 14.13: Wichtige Methoden für DOM-Knotenobjekte (Fortsetzung)

Methode	Anwendbar auf	Bedeutung/Beispiel
tagName	Elementknoten	Ermittelt den Elementnamen. `$elem_name = $elem_node->tagName;`
nodeName	Alle Knoten	Ermittelt den Namen eines Knotens. `$node_name = $node_object->nodeName;`
nodeValue	Alle Knoten	Ermittelt den Wert oder Inhalt eines Knotens. `echo $elem_node->nodeValue;`
parentNode	Alle Knoten	Nimmt auf den Elternknoten des aktuellen Knotens Bezug. `echo $elem_node=>$parentNode->tagName;`
childNodes	Alle Knoten	Liefert eine Knoten-Objektliste aller Kindknoten. `echo $elem_node=>$childNodes->length;`
firstChild	Alle Knoten	Liefert den ersten Kindknoten. `echo $elem_node->firstChild->tagName;`
lastChild	Alle Knoten	Liefert den letzten Kindknoten. `echo $elem_node->lastChild->tagName;`
previousSibling	Alle Knoten	Liefert den unmittelbar vorhergehenden Knoten auf gleicher Ebene. `echo $elem_node->previousSibling->tagName;`
nextSibling	Alle Knoten	Liefert den unmittelbar vorhergehenden Knoten auf gleicher Ebene. `echo $elem_node->nextSibling->tagName;`

Tabelle 14.14: Wichtige Eigenschaften für DOM-Knotenobjekte

Weitere Informationen zum DOM finden Sie innerhalb dieses Buchs in ***Abschnitt 7.4***.

Links zu weiteren Informationen:

W3C-Übersicht zu DOM:
http://www.w3.org/DOM/
DOM 1.0 (W3C-Empfehlung):
http://www.w3.org/TR/REC-DOM-Level-1/
DOM 2.0 Core (W3C-Empfehlung):
http://www.w3.org/TR/DOM-Level-2-Core/core.html
DOM-Funktionen in PHP:
http://de.php.net/manual/de/ref.dom.php

XML-Daten erzeugen und schreiben

Bestens geeignet ist die DOM-Klasse von PHP 5, um aus strukturierten Daten XML-Code zu erzeugen. Um das zu demonstrieren, nehmen wir ein Beispiel mit Daten einer Bestellung, an die das PHP-Script z. B. durch Formulareingaben eines Benutzers gekommen ist:

```
<?php
$data['customer'] = array(
  'given_name' => "Anna",
  'family_name' => "Lyse",
  'mail' => "anna_lyse@example.org",
  'customer_id' => 404035);
$data['order'] = array(
  'product_id' => "XR5934",
  'price' => 39.90,
  'tax_percent' => 16,
  'dispatch_mode' => "cash on delivery");


$dom = new DOMDocument("1.0", "UTF-8");
if(!$dom)
   exit("Konnte kein DOM-Objekt erzeugen!");
$dom->preserveWhiteSpace = false;

$root_elem = $dom->createElement('order');
$customer_elem = $dom->createElement('customer');
$name_elem = $dom->createElement('name');
$given_name_elem = $dom->createElement('given_name',
                   $data['customer']['given_name']);
$family_name_elem = $dom->createElement('family_name',
                   $data['customer']['family_name']);
$mail_elem = $dom->createElement('mail',
                   $data['customer']['mail']);
$details_elem = $dom->createElement('details');
$product_elem = $dom->createElement('product');
$price_elem = $dom->createElement('price',
                   $data['order']['price']);
$tax_percent_elem = $dom->createElement('tax_percent',
                   $data['order']['tax_percent']);
$dispatch_mode_elem = $dom->createElement('dispatch_mode',
                   $data['order']['dispatch_mode']);

$root = $dom->appendChild($root_elem);
$customer = $root->appendChild($customer_elem);
$customer->setAttribute("cid", $data['customer']['customer_id']);
$name = $customer->appendChild($name_elem);
$given_name = $name->appendChild($given_name_elem);
$family_name = $name->appendChild($family_name_elem);
$mail = $customer->appendChild($mail_elem);
$details = $root->appendChild($details_elem);
$product = $details->appendChild($product_elem);
$product->setAttribute("pid", $data['order']['product_id']);
$price = $details->appendChild($price_elem);
```

```
$tax_percent = $details->appendChild($tax_percent_elem);
$dispatch_mode = $details->appendChild($dispatch_mode_elem);

$dom->save("order_".time().".xml");
?>
```

Im oberen Teil des Beispielscripts sind zwei Arrays mit den Ausgangsdaten notiert. Diese Daten sollen in eine XML-Datei gespeichert werden. Der XML-Code, der aus den Daten zu erzeugen ist, soll am Ende so aussehen:

```
<?xml version="1.0" encoding="UTF-8"?>
<order>
  <customer cid="404035">
    <name>
      <given_name>Anna</given_name>
      <family_name>Lyse</family_name>
    </name>
    <mail>anna_lyse@example.org</mail>
  </customer>
  <details>
    <product pid="XR5934"/>
    <price>39.90</price>
    <tax_percent>16</tax_percent>
    <dispatch_mode>cash on delivery</dispatch_mode>
  </details>
</order>
```

Im weiterem Verlauf des obigen Beispielscripts wird genau dieser XML-Code erzeugt. Dazu erzeugen wir zunächst ein leeres, neues DOM-Objekt mit:

```
$dom = new DOMDocument("1.0", "UTF-8");
```

Daran anschließend werden erst einmal alle Elemente erzeugt, die im XML-Code am Ende vorkommen sollen. Mit der DOM-Methode `createElement()` werden Elemente zunächst nur erzeugt, aber noch nicht in die XML-Struktur des Dokuments eingehängt. Deshalb ist die Methode auch direkt über die Objektvariable des DOM-Objekts ansprechbar, im Beispiel also über die Variable `$dom`.

Funktion	**$DOM_Objektvariable->createElement()**
Aufgabe	Erzeugt einen Elementknoten, ohne ihn in die XML-Struktur einzuhängen.
Parameter	1. der Elementname 2. (optional) der Textinhalt des Elements
Rückgabewert	Ein Elementknotenobjekt

Tabelle 14.15: Steckbrief XPath-Methode »createElement()«

Wenn ein Element erzeugt werden konnte, gibt `createElement()` ein Elementknotenobjekt für das neu erzeugte Element zurück. In unserem Beispiel fangen wir diese Rückgabe jeweils in einer Variablen auf. Das ermöglicht es im nächsten Abschnitt des Scripts, die XML-Struktur aus den erzeugten Elementen zusammenzubauen.

Das Zusammenbauen des XML-Codes geschieht typischerweise über die DOM-Methoden `appendChild()` und `setAttribute()`. Die erste dieser Anweisungen im Script lautet:

```
$root = $dom->appendChild($root_elem);
```

Da `appendChild()` hier direkt auf die Objektvariable des DOM-Objekts angewendet wird, wird dadurch das Dokumentelement »eingehängt«. Auch `appendChild()` gibt ein Knotenobjekt zurück, nämlich das des in die XML-Struktur eingehängten Knotens. Auch diesen Rückgabewert fangen wir in Variablen auf, denn diese Variablen sind unsere »Marker« innerhalb der XML-Struktur. An der nachfolgenden Anweisung wird das deutlich:

```
$customer = $root->appendChild($customer_elem);
```

Dadurch wird der Elementknoten mit dem Namen `$customer_elem`, der ja weiter oben durch `createElement()` erzeugt wurde, als Kindknoten an den Elementknoten eingehängt, der in `$root` gespeichert ist, und dies ist der Knoten des Dokumentelements, das zuvor mit `appendChild()` eingehängt wurde. Nach diesem Prinzip können wir nun bequem die gesamte gewünschte XML-Elementstruktur erzeugen.

Funktion	**$Knotenobjektvariable->appendChild()**
Aufgabe	Hängt ein Kindelement in die XML-Struktur ein.
Parameter	1. das einzuhängende Knotenobjekt
Rückgabewert	Ein Knotenobjekt des eingehängten Knotens

Tabelle 14.16: Steckbrief XPath-Methode »appendChild()«

Auch zwei Attribute kommen in unserer Wunschdatenstruktur vor. Dazu verwenden wir die DOM-Methode `setAttribute()`. Angewendet wird sie jeweils auf denjenigen Elementknoten, in dessen Element das Attribut vorkommen soll.

Funktion	**$Knotenobjektvariable->setAttribute()**
Aufgabe	Fügt einem Elementknoten ein Attribut hinzu.
Parameter	1. der Attributname 2. der Wert des Attributs
Rückgabewert	Ein Knotenobjekt des eingehängten Attributknotens

Tabelle 14.17: Steckbrief XPath-Methode »setAttribute()«

Es wäre übrigens ebenso möglich gewesen, die Attribute im oberen Scriptteil zuzuweisen, also nach dem Erzeugen der Elemente.

Am Ende des Scripts, nachdem die gesamte XML-Struktur zusammengebaut wurde, bleibt nur noch, diese in eine Datei zu speichern. Dazu steht die DOM-Methode `save()` zur Verfügung.

Funktionen	**$DOM_Objektvariable->save()** **$DOM_Objektvariable->saveXML()**
Aufgabe	save() speichert ein DOM-Objekt in einer XML-Datei. saveXML() erzeugt aus dem DOM-Objekt eine Zeichenkette.
Parameter	1. bei save() die XML-Datei, wenn nötig mit absolutem Pfadnamen oder Pfadnamen aus Sicht des PHP-Scripts. Bei saveXML() optional eine Knotenobjektvariable. Wenn nicht angegeben, wird die gesamte XML-Struktur als Zeichenkette zurückgegeben, wenn angegeben, nur die XML-Unterstruktur des angegebenen Knotenobjekts.
Rückgabewert	Bei save() die Anzahl geschriebener Bytes bei Erfolg oder `false` im Fehlerfall Bei saveXML() die Zeichenkette mit dem XML-Code oder `false` im Fehlerfall

Tabelle 14.18: Steckbrief DOM-Methoden »save()« und »saveXML()«

Im Beispiel lautet die entsprechende Anweisung:

```
$dom->save("order_".time().".xml");
```

Damit wird eine XML-Datei mit der Endung .xml erzeugt. Der Dateiname beginnt mit »order_«, gefolgt von dem Rückgabewert der PHP-Funktion `time()`.

14.8.3 XML-Verarbeitung mit der XML-Klasse (Expat)

Wenn Sie bereits daran gewöhnt sind, XML mit der DOM-Schnittstelle zu verarbeiten, wird Ihnen das Konzept von Expat anfangs etwas befremdlich vorkommen. Expat arbeitet ereignisorientiert. Das bedeutet, der Code einer XML-Datei wird der Reihe nach eingelesen. Stößt der Expat-Parser dabei auf Elemente, Attribute usw., registriert er ein entsprechendes Ereignis. Für Ereignisse können Sie in PHP Handler-Funktionen bereitstellen, um die Ereignisse zu verarbeiten.

Zum besseren Vergleich mit den anderen XML-Verarbeitungsmethoden gehen wir wieder von folgendem XML-Dokument aus:

```
<?xml version="1.0"?>
<members>
   <member nr="19341" position="none">
      <given_name>Anna</given_name>
      <family_name>Lyse</family_name>
      <member_since>
         <yyyy>2003</yyyy><mm>10</mm><dd>18</dd>
      </member_since>
   </member>
   <member nr="11294" position="committee">
      <given_name>Reiner</given_name>
      <family_name>Wahnsinn</family_name>
      <member_since>
         <yyyy>1998</yyyy><mm>02</mm><dd>23</dd>
      </member_since>
   </member>
   <member nr="20845" position="none">
      <given_name>Mira</given_name>
```

```
      <family_name>Mare</family_name>
      <member_since>
         <yyyy>2005</yyyy><mm>06</mm><dd>01</dd>
      </member_since>
   </member>
</members>
```

Die Aufgabenstellung soll wieder darin bestehen, alle Daten über das Mitglied Reiner Wahnsinn zu ermitteln und auszugeben.

Der Code des PHP-Scripts lautet diesmal:

```
<?php

$elem = "";
$member = array(
  'nr' => "",
  'given_name' => "",
  'family_name' => "",
  'dd' => "",
  'mm' => "",
  'yyyy' => "");
$look_for = array(
  'given_name' => "Rainer",
  'family_name' => "Wahnsinn",
);

if(file_exists("members.xml")) {
   $xml_parser = xml_parser_create();
   xml_parser_set_option($xml_parser,
                         XML_OPTION_CASE_FOLDING, false);
   xml_set_element_handler($xml_parser,
                           "startElement", "endElement");
   xml_set_character_data_handler($xml_parser,
                                  "characterData");
}
else
   exit("members.xml nicht gefunden!");

$fp = fopen("members.xml", "r");
while($data = fread($fp, 4096))
   if(!xml_parse($xml_parser, $data, feof($fp)))
      exit(sprintf("XML-Fehler: %s in Zeile %d",
        xml_error_string(xml_get_error_code($xml_parser)),
        xml_get_current_line_number($xml_parser)));
xml_parser_free($xml_parser);
fclose($fp);

function startElement($parser, $name, $attrs) {
   global $member, $elem;
   $elem = $name;
   if($name == "member")
      foreach($attrs as $key => $value)
```

```
        if($key == "nr")
           $member['nr'] = $value;
}

function endElement($parser, $name) {
   global $member, $elem, $look_for;
   $elem = "";
   if($name == "member") {
      if($member['given_name'] == $look_for['given_name']
      and $member['family_name'] == $look_for['family_name'])
         print_data();
      $member['nr'] = "";
      $member['given_name'] = "";
      $member['family_name'] = "";
      $member['dd'] = "";
      $member['mm'] = "";
      $member['yyyy'] = "";
   }
}

function characterData($parser, $data) {
   global $member, $elem;
   $elems = array_keys($member);
   if(in_array($elem, $elems))
      $member[$elem] = $data;
}

function print_data() {
   global $member;
   echo "<p>Name: ", $member['given_name'], " ",
        $member['family_name'], "<br>";
   echo "Mitgliedsnummer: ", $member['nr'], "<br>";
   echo "Mitglied seit: ", $member['dd'], ".",
        $member['mm'], ".",$member['yyyy'] ,"</p>";
}
?>
```

Zu Beginn des Scripts werden zunächst einige globale Variablen deklariert. In `$elem` wird später der Name des jeweils abgearbeiteten Elements gespeichert. Das ist nötig, da eine Funktion, die sonst keine Kenntnis darüber hat, welches Element gerade aktuell bearbeitet wird, dies wissen muss. Im Array `$member` werden bei der Abarbeitung des XML-Codes die Daten des jeweils aktuellen `member`-Elements festgehalten. Im Array `$look_for` schließlich werden die Daten für die Suchbedingung vorgegeben.

Die XML-Verarbeitung beginnt mit:

```
$xml_parser = xml_parser_create();
```

Die Funktion `xml_parser_create()` erzeugt eine Prozessinstanz des Expat-Parser und gibt eine Ressource zurück, die im Beispiel in der Variablen `$xml_parser` gespeichert wird. Diese Ressource wird als Parameter für die Funktionen benötigt, die als Nächstes aufgerufen werden.

Funktion	xml_parser_create()
Aufgabe	Erzeugt eine Instanz des XML-Parsers.
Parameter	1. (optional) eine Angabe zur Zeichenkodierung
Rückgabewert	Ein Ressourcen-Handler für die Parser-Instanz

Tabelle 14.19: Steckbrief Funktion »xml_parser_create()«

```
xml_parser_set_option($xml_parser,
                      XML_OPTION_CASE_FOLDING, false);
```

Damit wird eine wichtige Option für den XML-Parser gesetzt: Er soll Groß- und Kleinschreibung bei Element- und Attributnamen lassen wie sie ist. Mit `true` als letztem Parameterwert würde er alle Element- und Attributnamen in Großbuchstaben konvertieren.

Funktion	xml_parser_set_option()
Aufgabe	Setzt Optionen für den XML-Parser.
Parameters	1. Ressourcenvariable des Parsers 2. Name der Option (möglich sind: `XML_OPTION_CASE_FOLDING`: true = Großschreibung, false = Groß-/Kleinschreibung lassen wie in XML-Daten `XML_OPTION_SKIP_TAGSTART`: (Integer) Ab dem wievielten Zeichen ein Elementname übertragen werden soll – sinnvoll bei Namensraumpräfixen `XML_OPTION_SKIP_WHITE`: true = Leerraumzeichen zwischen Elementen ignorieren, false = als Text behandeln `ML_OPTION_TARGET_ENCODING`: Zeichenkodierung der Zieldaten: ISO-8859-1, US-ASCII oder UTF-8) 3. zu setzender Wert für die Option
Rückgabewert	`true`, wenn die Option gesetzt werden konnte, `false`, wenn nicht.

Tabelle 14.20: Steckbrief Funktion »xml_parser_set_option()«

```
xml_set_element_handler($xml_parser,
                        "startElement", "endElement");
```

Damit beginnt die Ereignisbehandlung für den XML-Parser, und zwar für die Ereignisse »Start-Tag eines Elements« und »End-Tag eines Elements«. Neben der Parser-Ressourcenvariable werden zwei Namen von Funktionen übergeben. Diese Funktionen müssen im weiteren PHP-Quelltext vorkommen. In unserem Beispiel wird beim Ereignis »Start-Tag eines Elements« die Funktion `startElement()` aufgerufen und beim Ereignis »End-Tag eines Elements« die Funktion `endElement()`.

Funktion	xml_set_element_handler()
Aufgabe	Benennt die Handler-Funktionen für die Ereignisse »Start-Tag« und »End-Tag«.
Parameter	1. Ressourcenvariable des Parsers 2. Name der Funktion zum Verarbeiten von Start-Tags 3. Name der Funktion zum Verarbeiten von End-Tags
Rückgabewert	`true` bei Erfolg, andernfalls `false`

Tabelle 14.21: Steckbrief Funktion »xml_set_element_handler()«

```
xml_set_character_data_handler($xml_parser,
                               "characterData");
```

Damit wird die Ereignisbehandlung für das Ereignis »Textinhalt« gesetzt. Der zweite Parameter ist wieder der Name einer Funktion, die im Quelltext existieren muss – in unserem Beispiel die Funktion `characterData()`.

Die Funktionen `xml_set_element_handler()` und `xml_set_character_data_handler()` gehören zu den Funktionen des Expat-XML-Handlers und kommen in dieser Form praktisch in allen Scripts vor, die mit dem Parser arbeiten.

Funktion	xml_set_character_data_handler()
Aufgabe	Benennt die Handler-Funktionen für das Ereignis »Textinhalt von Elementen«.
Parameter	1. Ressourcenvariable des Parsers 2. Name der Funktion zum Verarbeiten von Textinhalten
Rückgabewert	`true` bei Erfolg, andernfalls `false`

Tabelle 14.22: Steckbrief Funktion »xml_set_character data_handler()«

Im nächsten Code-Block unseres Beispielscripts wird die XML-Datei eingelesen und an den Parser übergeben. Der Parser stellt keine eigenen Funktionen dafür zur Verfügung, so dass wir auf Standard-PHP-Funktionen zum Lesen von Dateien zurückgreifen müssen. Beim »Füttern« des Expat-Parser ist es üblich, Dateien so einzulesen wie in unserem Beispiel, also mit `fread()` in sinnvoll großen Blöcken, hier jeweils 4 Kilobyte.

Während des blockweisen Einlesens der Datei wird jedes Mal die Parser-Funktion `xml_parse()` aufgerufen. Sie bekommt wieder die Ressourcenvariable übergeben, die bei `xml_parser_create()` initialisiert wurde. Im zweiten Parameter werden die aktuell von `fread()` eingelesenen Daten übergeben und als dritter Parameter wird ein `true`/`false`-Wert übergeben, der angibt, ob das Ende der XML-Daten erreicht ist. Diesen Parameter versorgen wir durch Aufruf von `feof()`.

Funktion	xml_parse()
Aufgabe	Parst den Inhalt von XML-Daten.
Parameter	1. Ressourcenvariable des Parsers 2. (optional) `true`, um anzugeben, dass das Ende der XML-Daten erreicht ist, oder `false`, um anzugeben, dass es noch nicht erreicht ist
Rückgabewert	`true` bei erfolgreichem Parsen, andernfalls `false`

Tabelle 14.23: Steckbrief Funktion »xml_parse()«

Die Funktion `xml_parse()` gibt, wenn Fehler beim Parsen auftreten, den Wert `false` zurück. In diesem Fall kann über die Funktion `xml_get_error_code()` der aufgetretene Fehler ermittelt werden und über `xml_get_current_line_number()` die aktuelle Zeilennummer der XML-Datei.

Funktionen	xml_get_error_code() xml_error_string()
Aufgabe	`xml_error_code()` liefert die Fehlernummer des aufgetretenen Fehlers. `xml_error_string()` liefert die Fehlermeldung des aufgetretenen Fehlers.
Parameter	Bei `xml_get_error_code()`: 1. Ressourcenvariable des Parsers. Bei `xml_error_string()`: 1. gültige Fehlernummer, die zunächst mit `xml_get_error_code()` ermittelt werden kann.
Rückgabewert	`xml_get_error_code()` gibt eine Fehlernummer zurück. `xml_error_string()` gibt eine Fehlermeldung zurück.

Tabelle 14.24: Steckbrief Funktionen »xml_get_error_code()/xml_error_string«

Funktion	xml_get_current_line_number()
Aufgabe	Liefert die aktuell verarbeitete Zeilennummer der XML-Daten.
Parameter	1. Ressourcenvariable des Parsers
Rückgabewert	Nummer der verarbeiteten Zeile

Tabelle 14.25: Steckbrief Funktion »xml_get_current_line_number()«

Ist das Einlesen erfolgreich beendet, sollte die Parser-Ressource durch Aufruf von `xml_parser_free()` wieder freigegeben werden – ähnlich wie Dateiressourcen durch `fclose()` wieder freigegeben werden.

Funktion	xml_parser_free()
Aufgabe	Gibt die Parser-Ressource wieder frei.
Parameter	1. Ressourcenvariable des Parsers
Rückgabewert	`true` bei Erfolg, andernfalls `false`

Tabelle 14.26: Steckbrief Funktion »xml_parser_free()«

Der Rest unseres Beispielscripts besteht aus vier Funktionen. Die drei ersten davon sind die Handler-Funktionen für den XML-Parser. Sie tragen die zuvor bestimmten Namen `startElement()`, `endElement()` und `characterData()`. Alle drei Funktionen bekommen als ersten Parameter wieder die Ressourcenvariable des XML-Parsers übergeben. Die Funktionen für Elementbehandlung müssen als zweiten Parameter eine Variable übergeben bekommen, in der der Parser bei Aufruf der Funktionen den Namen des ereignisauslösenden Elements übergibt – im Beispiel `$name`. Die Funktion zum Behandeln von Start-Tags von Elementen bekommt außerdem einen dritten Parameter übergeben – im Beispiel `$attrs`. Darin übergibt der Parser einen assoziativen Array mit Attributen und ihren Werten, sofern in dem ereignisauslösenden Start-Tag welche vorkommen. Die Schlüssel dieses Arrays sind die Attributnamen.

Was die Ereignis-Handler-Funktionen tun, ist aufgabenabhängig. In unserem Fall sollen aus dem Datenstrom die Daten für ein bestimmtes Mitglied herausgefiltert werden. Dazu versorgen wir den globalen Array `$member` jeweils mit aktuellen Daten. Wenn ein Start-Tag `<member>` in den verarbeiteten Daten vorkommt, fängt die Handler-Funktion `startElement()` dies mit `if($name == "member")` ab. In diesem Fall werden zunächst die Attribute abgefragt und der Wert des Attributs `nr=` wird im globalen Array `$member` gespeichert.

In jedem Fall speichert `startElement()` das aktuell beginnende Element in der globalen Variablen `$elems`.

Die Textinhalte der XML-Daten werden vom XML-Parser an die Funktion `characterData()` übergeben. Diese kann durch die stets aktuell gehaltene globale Variable `$elems` überprüfen, welcher Elementname gerade aktuell ist. Den vergleicht sie mit den Schlüsseln des Arrays `$member`. Ist in `$elem` beispielsweise aktuell der Wert `given_name` gespeichert, findet die Funktion den passenden Array-Schlüssel `"given_name"` im Array `$member` und speichert die aktuell vom Parser erhaltenen Zeichendaten in diesem Array-Element.

Auf diese Weise füllt sich pro `member`-Element der XML-Daten der Array `$member` mit den entsprechenden Daten. Am Ende eines `member`-Elements ist der Array also gefüllt. Deshalb wird in der Handler-Funktion `endElement()` wieder mit `if($name == "member")` geprüft, ob es sich um ein `</member>`-Tag handelt. Falls ja, werden die `$member`-Array-Werte für `given_name` und `family_name` mit den Suchdaten verglichen, die in `$look_for` gespeichert sind. Ist das der Fall, haben wir das gesuchte Mitglied gefunden. Die Funktion `print_data()` wird aufgerufen. Sie gibt die Daten des gefundenen Mitglieds aus.

14.8.4 XSLT-Funktionen

```
Auch was XSLT betrifft, hat PHP für einige Verwirrung gesorgt. In der PHP-
Version 4.x waren zwei Lösungen implementiert: Die eine bediente sich der
DOMXML-Bibliothek und verwendete libxslt als XSLT-Prozessor, die andere dagegen
setzte auf dem XSLT-Prozessor Sablotron auf und stellte eigene XSLT-Funktionen
zur Verfügung.
```

Es wird jedoch Zeit, sich langsam von den alten Modellen zu verabschieden. In PHP 5 wird nur noch der libxslt-Prozessor (Version 2) unterstützt und der Zugang erfolgt nicht

mehr über die veraltete DOMXML-Klasse, sondern über die DOM-Klasse, die wir in ***Abschnitt 14.8.3*** vorgestellt haben.

Wie bei XSLT üblich und in ***Abschnitt 14.7*** beschrieben, benötigen Sie XML-Ausgangsdaten und ein XSLT-Stylesheet. Der Rest ist in PHP 5.x relativ einfach. Im nachfolgenden Beispiel nehmen wir an, dass eine Datei namens ***members.xml*** mit Hilfe eines Stylesheet, das in ***members.xsl*** gespeichert ist, in HTML-Code transformiert wird. Das PHP-Script stößt die Transformation an und gibt den dabei erzeugten HTML-Code an den aufrufenden Browser aus:

```
$xsl = new DomDocument();
$xsl->load("members.xsl");
$xml = new DomDocument();
$xml->load("members.xml");
$proc = new XsltProcessor();
$proc->importStylesheet($xsl);
$html = $proc->transformToDoc($xml);
echo $html->saveXML();
```

Zunächst wird ein neues Objekt der DOM-Klasse erzeugt und in der Objektvariablen `$xsl` gespeichert. Darin wird anschließend mit Hilfe der `load()`-Methode das XSLT-Stylesheet geladen, also der Inhalt von ***members.xsl***. Als Nächstes wird nach dem gleichen Schema noch ein DOM-Objekt erzeugt, diesmal für die XML-Quelldaten, gespeichert in ***members.xml***.

Damit sind alle Daten geladen, die benötigt werden und der XSLT-Prozessor kann gestartet werden. Zunächst muss eine Instanz des XSLT-Prozessors erzeugt werden. Dies geschieht durch Aufruf der Konstruktormethode `XsltProcessor()`. Die Funktion liefert eine Objektvariable als Handler für die erzeugte Prozessorinstanz zurück. Im Beispiel ist dies die Variable `$proc`. Über diese Objektvariable rufen wir als Nächstes die Funktion `importStylesheet()` auf. Damit wird das gewünschte XSLT-Stylesheet an den Prozessor übergeben. Als Parameter wird `importStylesheet()` die Objektvariable des XSLT-Stylesheet-Objekts übergeben.

Durch Aufruf der Methode `transformToDoc()` wird der eigentliche Übersetzungsvorgang angestoßen. Die Methode `transformToDoc()` bekommt dabei als Parameter die Objektvariable des XML-Quelldokuments übergeben. Ihr Rückgabewert ist ebenfalls ein Objekt, und zwar ein DOM-Objekt mit den erzeugten Ergebnisdaten. Dieses Objekt speichern wir im Beispiel in der Variablen `$html`, da unser XSLT-Stylesheet die XML-Quelldaten in HTML-Code übersetzt.

Mit Hilfe der DOM-Klassen-Methode `saveXML()` wird wie bereits bekannt der in einem DOM-Objekt gespeicherte XML-Code in eine durchgängige Zeichenkette umgewandelt. Diese Methode wenden wir auf die Objektvariable `$html` an, um den HTML-Code als Zeichenkette zu erhalten. Mit `echo` geben wir die zurückgegebene Zeichenkette schließlich an den Browser aus.

14.8.5 Praxisbeispiel: News-Script für RSS-Feed-Funktion

Nachdem wir nun einen Einblick in die XML-Verarbeitung mit PHP gewonnen haben, soll das Wissen nun noch anhand von Praxisbeispielen vertieft werden. In ***Abschnitt 14.4*** haben wir die Technik der RSS-Feeds aus XML- und Anwendersicht vorgestellt. Wenn Sie als Website-Betreiber selbst regelmäßige News anbieten und dafür einen RSS-Feed bereitstellen wollen, erledigen Sie das am besten über ein Script, so dass die XML-Daten des RSS-Feed automatisch aktualisiert oder generiert werden.

Wir nehmen dazu an, dass es bereits ein Script gibt, mit dessen Hilfe Sie Ihre News verwalten. Ein solches Script könnte sich beispielsweise im Aufbau an dem Script orientieren, das wir in ***Abschnitt 143.5.2*** dieses Buchs vorgestellt haben. Dort ging es um die Verwaltung von Produktdaten. Nach dem gleichen Schema lassen sich natürlich auch News verwalten.

Wir nehmen also an, alle News werden in einer MySQL-Datenbank gespeichert. Als Betreiber haben Sie eine nur intern zugängliche Weboberfläche, über die Sie News hinzufügen, ändern oder löschen können.

Doch wie kommen die so gespeicherten Daten als RSS-Feed zum Benutzer? Die Lösung besteht darin, die XML-Daten des RSS-Feeds nicht statisch als XML-Datei zu speichern, sondern mit Hilfe von PHP dynamisch aus der Datenbank zu generieren. Hinter dem URI, den der Benutzer in seinem RSS-Reader eingeben muss, um den RSS-Feed zu abonnieren, verbirgt sich dann keine XML-Datei, sondern ein PHP-Script, das XML-Code im RSS-Format erzeugt und sendet.

Die Aufgabe des PHP-Scripts besteht also darin, die MySQL-Datenbank mit den News auszulesen und die neuesten News (welche genau das sind, wird noch zu diskutieren sein) in ein dynamisch generiertes XML-Dokument zu bringen. Dazu muss das Script zunächst einmal die Struktur der Datenbanktabelle(n) verstehen, in der/denen die News gespeichert sind. In unserem Beispiel nehmen wir an, dass alle News in einer Datenbank namens ***site*** und dort in einer Tabelle namens ***news*** gespeichert sind. Für die Tabelle ***news*** nehmen wir folgenden Aufbau an:

Feldname	Datentyp	Bedeutung
nid	unsigned integer	Interne Datensatzlaufnummer; automatisch verwaltetes Feld
language	varchar(8)	Sprachenkürzel zur Bezeichnung der Landessprache der News; Pflichtfeld
title	varchar(255)	Titel der News; Pflichtfeld
short_text	blob	Kurz- bzw. Anlesertext der News; Pflichtfeld
long_text	blob	Langtext der News; freiwillige Angabe
pub_date	date	Offiziell ausgewiesenes Publikationsdatum

Tabelle 14.27: Aufbau der angenommenen Datenbanktabelle ***news***

Damit haben wir bereits alles, was wir an Voraussetzungen benötigen, und können uns dem Inhalt des Beispielscripts zuwenden:

```
<?php

// Variablen
$db_host = "localhost";
$db_user = "siteuser";
$db_passwd = "";
$db_name = "site";
$news_table = "news";
$feed_days_back = 10;
$feed_language = "de";
$feed_title = "Unsere tollen News";
$feed_url = "http://mein.server/rss.php";
$feed_description =
   "Fast täglich fallen Säcke mit Reis um. Wir sind dabei.";
$feed_master = "my.mail@example.org";
$webnews_url = "http://mein.server/news.php?news=";

// Datenermittlung
$dbh = mysql_connect($db_host, $db_user, $db_passwd);
if(!$dbh)
   die("Keine Datenbankverbindung!");
$db = mysql_select_db($db_name);
$from_date =
   date("Y-m-d", time() - ($feed_days_back * 24 * 60 * 60));
$sql = "select nid, title, short_text, pub_date from news";
$sql .= " where language = '".$feed_language."'";
$sql .= " and pub_date >= '".$from_date."'";
$sql .= " order by pub_date DESC";
$res = mysql_query($sql, $dbh);
if(!$res)
   die("Fehler beim Einlesen der News");
$news_data = array();
while($record = mysql_fetch_array($res, MYSQL_ASSOC))
   $news_data[] = $record;

// XML-Generierung RSS-Feed
$dom = new DOMDocument("1.0", "UTF-8");
if(!$dom)
   die("Konnte kein DOM-Objekt erzeugen!");
//$dom->preserveWhiteSpace = false;
$rss = add_element("rss", $dom);
$rss->setAttribute("version", "2.0");
$channel = add_element("channel", $rss);
add_value_element("title", $feed_title, $channel);
add_value_element("link", $feed_url, $channel);
add_value_element("description", $feed_description, $channel);
add_value_element("language", $feed_language, $channel);
add_value_element("pubDate",
   gmdate('D, d M Y H:i:s')." GMT", $channel);
add_value_element("webMaster", $feed_master, $channel);
```

```
for($i = 0; $i < count($news_data); $i++) {
   $item[$i] = add_element("item", $channel);
   add_value_element("title",
      $news_data[$i]['title'], $item[$i]);
  add_value_element("description",
      $news_data[$i]['short_text'], $item[$i]);
  add_value_element("pubDate",
      $news_data[$i]['pub_date'], $item[$i]);
  add_value_element("link",
      $webnews_url.$news_data[$i]['nid'], $item[$i]);
}

// Ausgabe der RSS-Daten
header("Expires: Sat, 1 Jan 2000 00:00:00 GMT");
header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
header("Cache-Control: no-store, no-cache, must-revalidate");
header("Content-Type: text/xml");
echo $dom->saveXML();

// Funktion zum Erzeugen/Einfügen eines Elements
// mit Elementinhalt
function add_element($name, $ref_obj) {
   global $dom;
   $elem = $dom->createElement($name);
   $tree_elem = $ref_obj->appendChild($elem);
   return($tree_elem);
}

// Funktion zum Erzeugen/Einfügen eines Elements
// mit Texttinhalt
function add_value_element($name, $value, $ref_obj) {
   global $dom;
   $utf8_value = utf8_encode($value);
   $elem = $dom->createElement($name, $utf8_value);
   $tree_elem = $ref_obj->appendChild($elem);
   return($tree_elem);
}
?>
```

Zunächst initialisieren wir einige Variablen, die Sie für den Einsatz des Scripts anpassen müssen. Dazu gehören die Variablen `$db_host`, `$db_user`, `$db_passwd`, `$db_name` und `$new_table`, in denen die Parameter für die MySQL-Datenbankverbindung gesetzt werden.

In der Variablen `$feed_days_back` können Sie festlegen, welcher Zeitraum der gespeicherten News im Newsfeed abgebildet werden soll. Der Wert bedeutet: Es werden nur News in den Feed übernommen, die in den letzten n Tagen veröffentlicht wurden. Auch die restlichen Variablen, beginnend mit `$feed_`, sollten Sie anpassen.

In der Variablen `$webnews_url` gehen wir davon aus, dass die News auch auf Ihrer Website präsentiert werden. Außerdem gehen wir davon aus, dass dahinter ein Script steckt, das in der Lage ist, jede einzelne News anhand ihrer nid-Nummer aus der Datenbank zu holen und separat anzuzeigen. Die Beispiel-Basis-URL `http://mein.server/news.php?news=`

bedeutet: Es gibt ein anderes, hier nicht behandeltes PHP-Script namens *news.php*, das einen GET-Parameter namens `news` versteht. Dabei erwartet es einen numerischen Wert, z. B. `news=13`. Aus diesem Wert generiert es eine Datenbankabfrage und sucht die News heraus, die im Datenbankfeld `nid` den Wert 13 hat. Die entsprechende News wird dann mit Kurz- und Langtext angezeigt. Unser Newsfeed dagegen ist so angelegt, dass nur der Kurztext angezeigt wird. RSS-Reader bieten jedoch einen Link zur zugehörigen HTML-Version einer News im Web an. Um diese Links mit Zieladressen zu versorgen, benötigen wir die URI-Basis, die in `$webnews_url` gespeichert ist.

Im anschließenden Block des Scripts werden die News aus der MySQL-Datenbank gelesen. Beim Konstruieren des dafür erforderlichen `select`-Statement ist sichtbar, dass einerseits nur News der zentral eingestellten Sprache selektiert werden und andererseits nur News, die nicht älter sind als 10 Tage (im Beispiel bei `$feed_days_back` festgelegt). Das `select`-Statement besorgt auch gleich die für die News-Präsentation übliche Sortierung. Der neueste News-Eintrag wird dadurch am Ende der oberste sein, der zweitneueste der zweitoberste usw.

Alle gefundenen News werden in den Array `$news_data` eingelesen. Dieser Array ist damit die Datenbasis für die Ausgabe.

Im dritten Block des Scripts beginnt die Generierung des XML-Codes im RSS-Format. Wir entscheiden uns für die Verarbeitung mit der DOM-Klasse von PHP 5, die wir in ***Abschnitt 14.8.3*** vorgestellt haben. Das erforderliche DOM-Objekt wird in der Variablen `$dom` gespeichert.

Um mit den DOM-Methoden ein Element in die XML-Zieldaten zu bekommen, sind bekanntlich immer zwei Schritte nötig: Zunächst muss das Element mit `createElement()` erzeugt werden und dann muss es mit einer geeigneten Methode wie `appendChild()` in die XML-Struktur eingehängt werden. Da wir einige Elemente zu erzeugen haben, verlagern wir dieses Prozedere in die beiden Funktionen `add_element()` und `add_value_element()`. Beide sind am Ende des Scripts notiert. Die Funktionen unterscheiden sich nur dadurch, dass `add_element()` für Elemente gedacht ist, deren Inhalt aus Kindelementen besteht, und `add_value_element()` für Elemente, deren Inhalt aus Textdaten (#PCDATA) besteht. Beide Funktionen erwarten als ersten Parameter den Namen des gewünschten neuen Elements und als letzten Parameter die Objektvariable eines Bezugsobjekts im XML-Zielbaum. Die Funktion `add_value_element()` erwartet außerdem als mittleren Parameter den gewünschten Textinhalt des Elements.

Mit diesen beiden Funktionen werden nun die XML-Daten erzeugt. Zunächst wird das Dokumentelement `rss` erzeugt. Deshalb wird als Bezugselement die Objektvariable `$dom` des DOM-Objekts übergeben. Der nächste Aufruf von `add_element()` fügt das `channel`-Element ein, das ein Kindelement des `rss`-Elements ist. Deshalb wird jetzt die Objektvariable `$rss`, die zuvor beim Einfügen des `rss`-Elements von `add_element()` versorgt wurde, übergeben. So entsteht die Verschachtelungsstruktur der Elemente.

Die nächsten Elemente sind inhaltstragende Elemente innerhalb des `channel`-Elements. Deshalb wird für sie die Funktion `add_value_element()` aufgerufen und als Bezugsobjekt-

variable wird jeweils `$channel` übergeben. Als zweiter Parameter werden die gewünschten Daten für den Elementinhalt übergeben.

Die eigentlichen News-Einträge eines RSS-Kanals werden bekanntlich in Elementen namens `item` gespeichert. Die `item`-Elemente sind ebenfalls Kindelemente des `channel`-Elements. Da `$news_data` ein Array mit allen ermittelten News-Einträgen aus unserer Datenbank ist, setzen wir eine `for`-Schleife ein, um den Array abzuarbeiten.

Innerhalb der `for`-Schleife fügen wir zunächst jeweils das neue `item`-Element ein und speichern das von `add_element()` zurückgegebene Objekt in `$item[$i]`. Die übrigen Elemente, die innerhalb der `for`-Schleife eingefügt werden, sind inhaltstragende Kindelemente des jeweiligen `item`-Elements. Deshalb werden sie wieder mit `add_value_item()` eingefügt. Als dritter Parameter wird dabei jeweils `$item[$i]` übergeben, wodurch die Elemente als Kindelemente des aktuell verarbeiteten `item`-Elements eingehängt werden. Jedes `item`-Element erhält auf diese Weise die Unterelemente `title`, `description`, `short_text`, `pubDate` und `link`.

Nach Abarbeitung der `for`-Schleife sind die XML-Daten im RSS-Format fertig erzeugt. Nun müssen sie nur noch ausgegeben bzw. an den wartenden Client gesendet werden. Wir nehmen dazu an, dass die Daten über das HTTP-Protokoll versendet werden. Durch entsprechende Aufrufe der `header()`-Funktion von PHP erzwingen wir, dass ein Client bei Aufruf die Daten neu aus dem Cache holt und dass die Daten mit dem Mime-Type `text/xml` übertragen werden.

Am Ende genügt ein echo-Aufruf, der den Rückgabewert der Methode `saveXML()` ausgibt.

14.8.6 Praxisbeispiel: Google-Suche als SOAP-Webservice

Der Suchdienst Google expandiert seit geraumer Zeit weit über seine ursprünglichen Grenzen einer Suchmaschine hinaus und zeigt sich dabei durchaus geberfreundlich. Während spektakuläre Erweiterungen wie Google-Maps im Blickpunkt der Öffentlichkeit stehen, führen andere Extra-Features eher ein Dasein im Verborgenen. Eines davon ist der Webservice, der es erlaubt, die Google-Suche ohne die Google-Website anzusprechen und die Suchergebnisse direkt in eigene Webseiten mit eigenem Layout zu integrieren.

Um den Webservice nutzen zu können, ist es wie bei den meisten bereits existierenden Webservices erforderlich, einen Benutzerzugang einzurichten. Dazu rufen Sie die Adresse *http://www.google.com/apis/* auf. Klicken Sie dort auf den Link ***Create Account*** und folgen Sie der weiteren Benutzerführung. Wenn Sie als Benutzer bei Google angemeldet sind, können Sie sich über die Adresse *http://api.google.com/createkey* einen Zugangsschlüssel generieren lassen. Diesen Schlüssel benötigen Sie in Ihrem PHP-Script.

Wichtig zu wissen ist, dass der ansonsten freie Webservice derzeit auf 1000 Abfragen pro Tag beschränkt ist. Werden mehr Anfragen gestartet, reagiert der Webservice mit einer SOAP-Fault-Message (siehe ***Abschnitt 14.5.4***). Weitere Einschränkungen bestehen darin, dass pro Aufruf maximal zehn Treffer »geholt« werden können. Jedes Weiterblättern zu den zehn nächsten Treffern gilt wieder als neue Abfrage. Maximal werden Daten bis zum 1000. Suchtreffer ausgegeben. Untersuchungen haben außerdem ergeben, dass es Unter-

schiede zwischen den vom Webservice gelieferten Suchtreffern und denen der Google-Standard-Weboberfläche gibt. Das alles sollte Sie jedoch nicht davon abhalten, den Service einmal auszuprobieren.

In unserem Beispiel möchten wir einfach eine normale Google-Suche realisieren, wobei wir die Suchergebnisse im eigenen Layout präsentieren wollen. Dazu wählen wir folgendes Konzept:

- Ein HTML-Template sorgt für den Rahmen.
- Ein zentrales PHP-Script namens *google.php* übernimmt das gesamte Handling. Wenn ein Suchwunsch aus dem Formular anliegt, wird der Webservice von Google aufgerufen. Unterhalb des Suchformulars werden die Suchtreffer angezeigt.

Seit Produktversion 5.0 bietet PHP erfreulicherweise SOAP-Unterstützung an. Derzeit muss dazu jedoch der XML-Parser *libxml 2.5.4* oder höher installiert sein und PHP muss mit der Option `--enable-soap` kompiliert sein. In der *php.ini* muss in der Abteilung für Extensions die Erweiterung für SOAP aktiviert sein.

Mit folgendem kleinen PHP-Script können Sie testen, ob SOAP in Ihrer PHP-Installation verfügbar ist:

```
<?php
if(class_exists(SoapClient))
  exit("SOAP ist verfügbar");
else
  exit("SOAP ist leider nicht verfügbar");
?>
```

In unserem Beispiel greifen wir auf frühere Beispiele in diesem Buch zurück. So werden wir anhand des Winkellayouts (beschrieben in ***Abschnitt 5.1.3***) demonstrieren, wie sich der Google-Webservice in ein eigenes Layout einpassen lässt. Die Stylesheet-Datei *winkel.css* übernehmen wir wie in 5.1.3 beschrieben. Den HTML-Code notieren wir wie oben angekündigt in einem Template. Dessen Code lautet:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
   <head>
      <meta http-equiv="content-type" content="text/html;
         charset=ISO-8859-1">
      <title>Google-Suche</title>
      <link rel="stylesheet" type="text/css" href="winkel.css">
   </head>
   <body>
      <div id="top_section">
         <img src="winkellogo.gif" id="logo" alt="Logo">
         <form id="search" action="google.php" method="get">
            <input type="hidden" id="offset"
                        name="offset" value="[%offset%]">
            <span id="search_label">Suche in Google:</span>
            <input type="text" id="search_text"
```

```
            name="search_text" value="[%search%]">
           <input type="submit" src="ok.gif"
            id="search_button" value="OK">
        </form>
      </div>
      <div id="main_section">
        <div id="navigation">
          <img id="navigation_pic" src="navi.gif"
           alt="Navigation">
        </div>
        <div id="content">
       [%content%]
       <hr style="border: solid 1px rgb(175,35,35)">
       <p>[%prev%][%prevnext%][%next%]</p>
        </div>
      </div>
    </body>
</html>
```

Das Template bindet die Stylesheetdatei ***winkel.css*** ein, um den HTML-Code mit einem Seitenlayout zu unterlegen. Wie schon in den Template-Beispielen von Kapitel 12 arbeiten wir auch diesmal wieder mit Platzhaltervariablen, die im Template in der Form `[%Variablenname%]` notiert sind. Zu den Aufgaben des PHP-Scripts gehört es, diese Variablen mit Inhalten zu versorgen.

Folgende Variablen kommen im Template vor:

- `[%offset%]`
  Der Wert eines hidden-Felds im Suchformular, in dem sich das Script merkt, welcher Ausschnitt der Suchergebnisse angezeigt werden soll (»Treffer 1 bis 10«, »Treffer 11 bis 20« usw.). Das Feld wird den 0-basierten Index des ersten auszugebenden Treffers speichern, so dass also 0 so viel bedeutet wie »Treffer 1 bis 10 anzeigen«, 10 so viel wie »Treffer 11 bis 20 anzeigen« usw.

- `[%search%]`
  Der Wert des Suchfelds. Ruft sich das Script selbst mit einem in dem Suchfeld eingegebenen Suchausdruck auf, fügt es den Suchausdruck als Vorbelegung in das Suchfeld ein. Auch das ist eine wichtige Funktionalität, da der Benutzer in den Suchergebnissen blättern können soll.

- `[%content%]`
  Dieser Platzhalter wird vom Script, falls keine Suchanfrage vorliegt, mit einem Text zur Suchaufforderung ersetzt. Falls eine Suchanfrage vorliegt, werden an dieser Stelle die Suchergebnisse eingefügt, die Google liefert.

- `[%prev%][%prevnext%][%next%]`
  Diese drei Variablen dienen zur Platzierung der Links zum Blättern in den Suchtreffern.

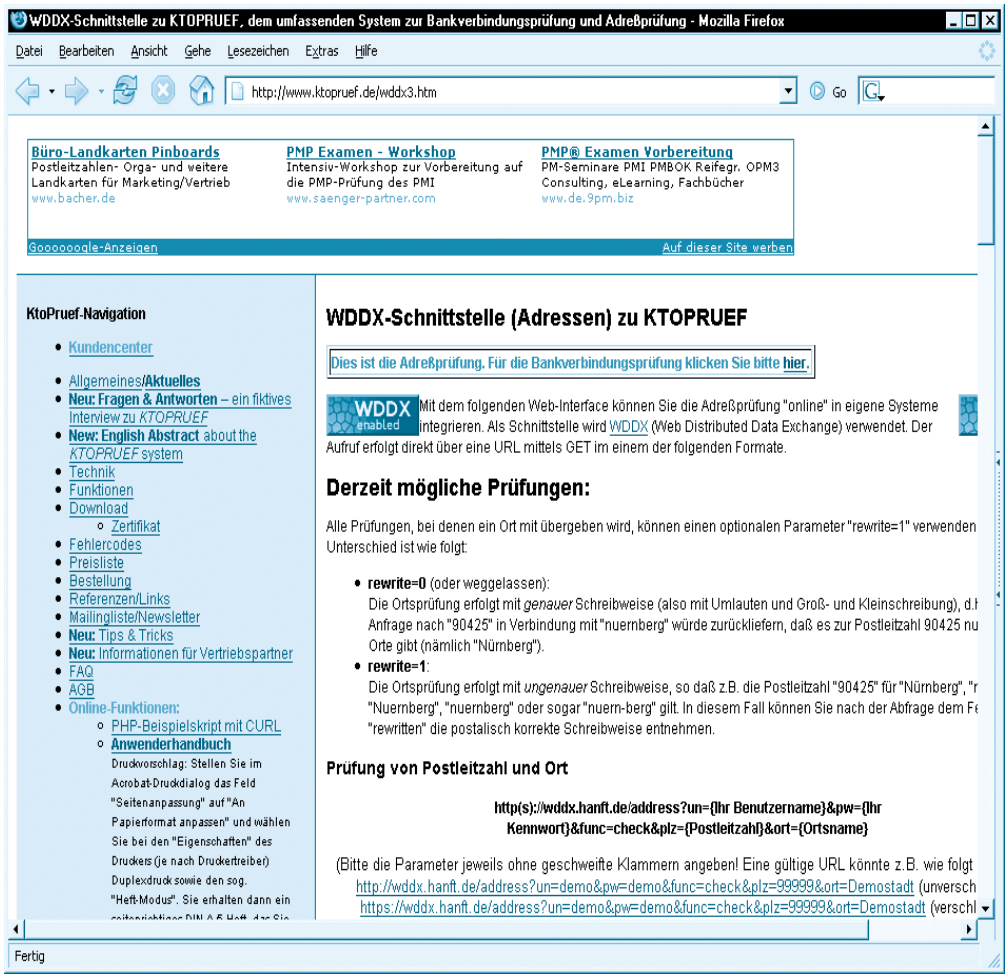


Abbildung 14.7: Google-Suche im eigenen Weblayout

Der vollständige Code unseres PHP-Scripts lautet:

```
<?php

// Template einlesen
$layout_template = "google.tpl";
$page = file_get_contents($layout_template);
$est_total_hits = 0;
$found_hits = 0;
$offset = 0;
$prev = "";
$next = "";
$prevnext = "";

// sinnvolle Optionen einstellen
ini_set("soap.wsdl_cache_enabled", "1");
```

```
ini_set("soap.wsdl_cache_dir", "/tmp");
ini_set("soap.wsdl_cache_ttl", "86400");

// Wenn eine Suche vorliegt
if(isset($_GET['search_text']) and isset($_GET['offset'])) {
   if(get_magic_quotes_gpc())
      $_GET['search_text'] = stripslashes($_GET['search_text']);
  $content = get_google_results();
  $page = str_replace("[%search%]",
          htmlspecialchars($_GET['search_text']), $page);
  $page = str_replace("[%content%]", $content, $page);
  if((int) $_GET['offset'] > 10) {
     $offset = (int) $_GET['offset'] - 10;
     $prev = "&amp;offset=".$offset;

  }
  else {
     if((int) $_GET['offset'] > 0)
        $prev = "&amp;offset=0";
  }
  if($est_total_hits > (int) $_GET['offset']) {
     $offset = (int) $_GET['offset'] + 10;
     $next = "&amp;offset=".$offset;
  }
  if(!empty($prev)) {
     $search = urldecode(htmlspecialchars($_GET['search_text']));
     $prevlink = "<a href=\"google.php?search_text=";
     $prevlink .= $search.$prev."\">&lt;&lt;</a>";
     $prev = $prevlink;
     $prevnext = " Ergebnisse ";
  }
  if(!empty($next)) {
     $search = urldecode(htmlspecialchars($_GET['search_text']));
     $nextlink = "<a href=\"google.php?search_text=";
     $nextlink .= $search.$next."\">&gt;&gt;</a>";
     $next = $nextlink;
     $prevnext = " Ergebnisse ";
  }
}
else {
  $page = str_replace("[%search%]", "", $page);
  $page = str_replace("[%content%]",
            "<p>Suchen Sie in Google!</p>", $page);
}
$page = str_replace("[%prev%]", $prev, $page);
$page = str_replace("[%prevnext%]", $prevnext, $page);
$page = str_replace("[%next%]", $next, $page);
$page = str_replace("[%offset%]", $offset, $page);
echo $page;

// Funktion zum Ermitteln der Google-Suchergebnisse
function get_google_results() {
   global $found_hits, $est_total_hits;
```

```
    $get_offset = (int) $_GET['offset'];
    // SOAP-Client erzeugen
    $client =
       new SoapClient("http://api.google.com/GoogleSearch.wsdl",
           array('trace' => 1,
                 'exceptions' => 0));
    // Suchbegriff in UTF-8 umwandeln
    $search_term = utf8_encode($_GET['search_text']);
    // Google-Suche über SOAP starten
    $result = $client->doGoogleSearch(
       "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX", // API-Key
       $search_term,   // Suchausdruck
       $get_offset,    // Zeige Ergebnisse ab 1. Treffer
       10,         // Anzahl Treffer - 10 ist das Maximum
       FALSE,      // ähnliche Treffer löschen
       '',         // keine Themenbeschränkung
       false,      // auch Treffer "nur für Erwachsene" anzeigen
       '',         // keine Beschränkung auf bestimmte Sprache
       '','');     // diese beiden Parameter immer so übergeben
    // Prüfen ob SOAP-Antwort ein Fault ist
    if(is_soap_fault($result)) {
       $html = "<p>Webservice-Fehler ";
       $html .= $result->faultcode.": ";
       $html .= $result->faultstring."</p>";
       return($html);
    }
    $found_hits = count($result->resultElements);
    $est_total_hits = $result->estimatedTotalResultsCount;
    if($found_hits == 0) {
       $html = "<p>Google hat keine passenden ";
       $html .= " Ergebnisse gefunden:</p>\n";
       return($html);
    }
    $html = "<p>Ergebnis ".$result->startIndex." bis ".$result->endIndex;
    $html .= " von isg. ".$result->estimatedTotalResultsCount;
    $html .= " Ergebnissen :</p>\n<ul>\n";
    foreach($result->resultElements as $hit) {
       $html .= "<li><a href=\"".$hit->URL."\">";
       $html .= utf8_decode($hit->title)."</a><br>";
       $html .= utf8_decode($hit->snippet)."<br>";
       $html .= "<small style=\"font-style: italic\">";
       $html .= $hit->URL."</small></li>";
    }
    $html .= "</ul>\n";
    return($html);
}
?>
```

Im ersten Scriptblock werden einige Variablen initialisiert. In `$page` wird die Template-Datei eingelesen. Die weiteren Variablen erläutern wir weiter unten noch.

Als Nächstes werden einige SOAP-spezifische Einstellungen für PHP vorgenommen. Es handelt sich um Werte, die normalerweise in der *php.ini* gesetzt sind und mit `ini_set()` für das aktuelle Script angepasst werden:

```
ini_set("soap.wsdl_cache_enabled", "1");
```

Damit wird die WSDL-Datei des Webservice clientseitig (also auf dem Rechner, auf dem unser PHP-Script läuft) gespeichert, so dass sie nicht bei jedem Aufruf neu übertragen werden muss. Die WSDL-Datei des Google-Webservice ist zwar nur ein paar Kilobyte groß, doch bei vielen Aufrufen kann auf diese Weise einiges an Datenübertragung gespart werden.

```
ini_set("soap.wsdl_cache_dir", "/tmp");
```

Diese Anweisung legt ein Verzeichnis fest, in der die zwischengespeicherte WSDL-Datei abgelegt werden soll. Eine absolute Pfadangabe sollte den Verzeichnispfad benennen.

```
ini_set("soap.wsdl_cache_ttl", "86400");
```

Damit wird die Lebensdauer der zwischengespeicherten WSDL-Datei begrenzt. Der Wert bedeutet Sekunden. 86400 ist die Sekundenanzahl eines Tages. Jeden Tag wird die lokale Cache-Kopie der WSDL-Datei also einmal erneuert (natürlich nur, wenn das PHP-Script angestoßen wird).

Im nächsten Scriptblock wird zunächst abgefragt, ob beim Scriptaufruf die beiden GET-Parameter `search_text` und `offset` übergeben wurden. Wenn ja, ist dies das Signal für das Script, den Webservice von Google anzustoßen. Dies geschieht in der Zeile:

```
$content = get_google_results();
```

Wir rufen also eine Subroutine bzw. Funktion auf, in die wir die eigentliche Kommunikation mit dem Google-Webservice ausgelagert haben. Die Funktion `get_google_results()` ist in der zweiten Hälfte des Scripts notiert. Innerhalb der Funktion können Sie sehen, wie die SOAP-Unterstützung von PHP funktioniert. Alles ist im Grunde sehr einfach und Sie müssen gar keine Details zum Aufbau der XML-basierten SOAP-Messages wissen, die PHP aus Ihren Anweisungen generiert. Auch die vom Service-Server erhaltenen Message-Inhalte stehen gleich in Variablenform zur Verfügung, so dass überhaupt kein Parsen von XML nötig ist.

Mit der Konstruktormethode `new SoapClient()` wird ein neuer SOAP-Client-Prozess in PHP erzeugt. Wenn das Objekt erfolgreich erzeugt werden kann, wird es in unserem Beispiel in der Objektvariablen `$client` gespeichert.

Die Methode `SoapClient()` erwartet ein oder zwei Parameter. Der erste ist der URI der WSDL-Datei, die den gewünschten Webservice beschreibt (Details dazu siehe ***Abschnitt 14.5.5***). Im Fall des hier gewünschten Google-Webservice ist dies der URI `http://api.google.com/GoogleSearch.wsdl`. Diese Adresse können Sie übrigens auch im Browser aufrufen. Abhängig von den Fähigkeiten und Einstellungen des Browsers wird dann der XML-Quelltext der Datei angezeigt oder die Datei wird zum Download angeboten (in letzterem Fall können Sie sie nach dem Download mit einem Betrachter oder Editor öff-

nen). Ein Blick in die WSDL-Datei von Google lohnt sich, denn sie enthält die komplette Beschreibung aller Service-Features.

Der zweite Parameter, den `SoapClient()` erwartet, ist dann optional, wenn beim ersten Parameter eine WSDL-Datei angegeben wird. Es ist auch möglich, anstelle des WSDL-URI den Wert `null` zu übergeben (`new SoapClient(null, ...)`. Wird `null` übergeben, ist der zweite Parameter verpflichtend.

Wenn er angegeben wird, muss der zweite Parameter aus einem assoziativen Array bestehen. Folgende Schlüssel kann der Array unter anderem besitzen:

- `[%soap_version%]`
  Angabe, in welcher SOAP-Version PHP mit dem Webservice kommunizieren soll (1.1 oder 1.2)
- `[%login%]` und `[%password%]`
  Benutzername und Passwort für die Anmeldung, falls der Webservice HTTP-Authentifizierung erfordert
- `[%compression%]`
  Ermöglicht die komprimierte Übertragung der SOAP-Messages, um die zu übertragenden Datenmengen zu reduzieren. Eine typische Angabe lautet: `SOAP_COMPRESSION_ACCEPT | SOAP_COMPRESSION_GZIP` (damit werden die beiden vordefinierten Konstanten für GZIP-komprimierte Übertragung und Akzeptieren solcher ankommenden Daten eingeschaltet).
- `[%encoding%]`
  Innerhalb des Scripts verwendete Zeichenkodierung. Je nach Angabe werden zu übertragende Zeichenketten dann automatisch in SOAP-gemäßes UTF-8 konvertiert.
- `[%trace%]`
  Ermöglicht die Rückverfolgung der zuletzt stattgefundenen SOAP-Kommunikation. Kann auf `true` bzw. `1` für ja oder `false` bzw. `0` für nein gesetzt werden.
- `[%exceptions%]`
  Ermöglicht das Abfangen von SOAP-Fault-Messages. Kann auf `true` bzw. `1` für ja oder `false` bzw. `0` für nein gesetzt werden.
- In unserem Beispiel haben wir `trace` auf `1` und `exceptions` auf `0` gesetzt. Letzteres ist wichtig, da wir die Fault-Messages nicht automatisch abfangen wollen, sondern eine script-eigene Fehlerbehandlung dafür einsetzen.
- Der Suchausdruck, der an den Google-Webservice übergeben wird, muss in jedem Fall in UTF-8-Kodierung umgewandelt werden, bevor er an den Webservice gesendet wird. Dies erreichen wir, indem wir den Formularfeldinhalt aus `$_GET['search_text']` an die Funktion `utf8_encode()` übergeben.
- Als Nächstes folgt das Kernstück der Webservice-Kommunikation. Über die Objektvariable `$client` wird eine Methode namens `doGoogleSearch()` aufgerufen. Der Name dieser Methode kommt jedoch nicht von PHP, sondern aus der WSDL-Datei von Google.

Dort steht verzeichnet, welche Funktionen ein Webservice-Client beim Webservice aufrufen kann. Eine davon ist beim Google-Webservice die Funktion `doGoogleSearch()`.

- Durch Aufruf dieser Methode über die Objektvariable des SOAP-Client-Objekts wird die SOAP-Kommunikation zwischen PHP-Script angestoßen. PHP erzeugt aus den Parametern, die der Methode `doGoogleSearch()` beim Aufruf übergeben werden, SOAP-konformen XML-Code. Wie die SOAP-Messages dabei im Einzelnen aufgebaut sein müssen, entnimmt das Script wiederum der zuvor eingelesenen WSDL-Datei. Auch die Parameter, die `doGoogleSearch` erwartet, sind in der WSDL-Datei definiert. Folgende Parameter werden übergeben:

- Parametername: `key`
  Dies ist der Schlüssel, den Sie wie weiter oben beschrieben von Google erhalten, wenn Sie sich zur Nutzung des Webservice anmelden. Die XXX-Folge in unserem Beispielscript müssen Sie durch die Zeichenfolge Ihres Key ersetzen.

- Parametername: `start`
  Dies ist ein 0-basierter Index dafür, ab dem wievielten Treffer Suchergebnisse zurückgeliefert werden sollen. Da wir in unserem Script das Blättern in den Suchergebnissen ermöglichen, verwalten wir den aktuellen Offset in einer Variablen namens `$get_offset`.

- Parametername: `maxResults`
  Die Anzahl der zu liefernden Suchtreffer für den aktuellen Aufruf. Leider erlaubt Google hier maximal zehn. Durch unsere Blätterfunktion ermöglichen wir es dem Benutzer, die nächsten oder vorherigen zehn Treffer anzuzeigen.

- Parametername: `filter`
  `TRUE`/`FALSE`-Wert, ob ähnliche Treffer angezeigt werden sollen. `FALSE` bedeutet: keine Einschränkung, also alle Treffer liefern.

- Parametername: `restricts`
  Suche auf eine bestimmte Kategorie beschränken: etwas unausgereifter Parameter, da Google bislang nur vier Kategorien dieser Art erkennt: `unclesam` für US-Government, `linux` für Linux, `mac` für Macintosh und `bsd` für FreeBSD. Übergeben Sie daher im Normalfall eine leere Zeichenkette wie in unserem Beispiel.

- Parametername: `safeSearch`
  `TRUE`/`FALSE`-Wert, ob Treffer mit Links zu Zielen angezeigt werden sollen, deren Inhalt nicht für Kinder und Jugendliche geeignet ist. Google hat ein eigenes Verwaltungssystem zur Erkennung nichtjugendfreier Inhalte. Mit `TRUE` werden Treffer zu solchen Zielen nicht übertragen, mit `FALSE` werden sie mit übertragen.

- Parametername: `lr`
  Dies steht für language restriction und ermöglicht es, die Suchtreffer auf Ziele zu beschränken, deren Inhalt in einer bestimmten Landessprache verfasst ist. Übergeben Sie, wenn Sie keine Sprachbeschränkung wünschen wie in unserem Beispiel eine leere Zeichenkette oder übergeben Sie einen Wert nach dem Muster `lang_[Sprachenkürzel]`, also z. B. `lang_de` für Deutsch, `lang_en` für Englisch, `lang_fr` für Französisch, `lang_es` für Spanisch oder `lang_tr` für Türkisch.

- Parameternamen: `ie` und `oe`:
  Diese Parameter müssen noch übergeben werden, werden jedoch nicht mehr ausgewertet. Übergeben Sie daher stets wie in unserem Beispiel für die beiden letzten Parameter eine leere Zeichenkette.

- Die Ergebnisse, die der Webservice von Google durch Aufruf von `doGoogleSearch()` via SOAP zurückliefert, wandelt PHP automatisch in ein »Ergebnisobjekt« um. Im Beispiel fangen wir dieses Ergebnisobjekt in der Variablen `$result` auf. Das Ergebnisobjekt verfügt über eine ganze Reihe von Eigenschaften, die mit Werten gefüllt sind. Welche Eigenschaften das sind und wie sie heißen, wird wiederum durch den Webservice bzw. die entsprechenden Definitionen in der WSDL-Datei bestimmt. Auch hier lohnt es sich also wieder, als Programmierer einen Blick in die WSDL-Datei zu werfen, um zu erfahren, welche Ergebnisdaten verfügbar sind.

- Zunächst prüfen wir in unserem Script jedoch, ob das Ergebnisobjekt auch tatsächlich Ergebnisse enthält oder ob der Webservice aus irgendeinem Grund eine Fault-Message, also einen Fehler zurückgeliefert hat. Dazu steht die objektunabhängige PHP-Funktion `is_soap_fault()` zur Verfügung. Als Parameter übergeben wir ihr die Ergebnisobjektvariable, in unserem Fall also `$result`. Die Funktion liefert `true` zurück, wenn ein Fehler aufgetreten ist. In diesem Fall speichert unser Script in der Variablen `$html`, die das Suchergebnis HTML-formatiert speichern soll, nur eine entsprechende Fehlermeldung. Dabei greifen wir über die Objektvariable `$result` auf die Eigenschaften `faultcode` (für die Fehlernummer) und `faultstring` (für den Fehlermeldungstext) zu, um dem Benutzer die entsprechenden Details anzuzeigen.

- Wenn kein Fehler aufgetreten ist, überprüfen wir als Nächstes, wie viele Ergebnisse überhaupt übermittelt wurden. Falls null Ergebnisse übermittelt wurden, reagieren wir ähnlich wie bei einem SOAP-Fehler und speichern in `$html` nur eine entsprechende Mitteilung an den Benutzer.

- Wenn dagegen Ergebnisse übertragen wurden, speichern wir in `$html` zunächst die üblichen Benutzerinformationen, nämlich Treffer wie viel bis wie viel von insgesamt wie viel gerade ausgegeben werden. An die entsprechenden Werte kommen wir durch Abrufen von Eigenschaften des Objekts, das in `$result` gespeichert ist. `$result->startIndex` gibt an, ab Treffer wie viel die Ausgabe gilt, `$result->endIndex`, bis Treffer wie viel und `$result->estimatedTotalResultsCount`, wie viele Treffer insgesamt gefunden wurden. Das `estimated` signalisiert, dass Google bei hohen Trefferzahlen hier nur gerundete Ungefährwerte ausgibt.

- Die eigentlichen Treffer sind in `$result->resultElements` gespeichert. In einer `foreach`-Schleife arbeiten wir die Treffer ab und fügen die gewünschten Daten zu jedem Treffer unserer Variablen `$html` hinzu. Jedes Trefferelement stellt dabei wiederum ein Unterobjekt dar, das wir innerhalb der `foreach`-Schleife über die Durchlaufvariable `$hit` ansprechen.

- Welche Eigenschaften ein Trefferobjekt haben kann, steht – wieder einmal – in der WSDL-Datei. Auf folgende Eigenschaften können Sie zugreifen:
  - `URL`
    Die Internetadresse des Suchtreffers.
  - `title`
    Der Titel des Suchtreffers (bei HTML-Inhalten der Inhalt des HTML-title-Elements des Verweisziels).
  - `snippet`
    Der Umgebungstext des Suchtreffers, wie ihn Google auch bei seiner Weboberfläche üblicherweise bei jedem Suchtreffer mit anzeigt.
  - `cachedSize`
    Anzahl Kilobyte des Verweisziels (ohne darin referenzierte Inhalte).
  - `relatedInformationPresent`
    Gibt den Google-Zusatz-Feature-Link »Ähnliche Seiten« bei Suchtreffern mit aus, für die dieses Feature verfügbar ist
  - `summary`
    Falls der Suchtreffer im redaktionell gepflegten Google-Webverzeichnis vorkommt, wird der dort gespeicherte Beschreibungstext angegeben.
  - `directoryTitle`
    Falls der Suchtreffer im redaktionell gepflegten Google-Webverzeichnis vorkommt, wird der dort gespeicherte Titel angegeben.
- Am Ende unserer Funktion `get_google_results` geben wir die mit den Suchtreffern gefüllte Variable `$html` zurück.
- Im oberen Teil des Scripts, wo die Funktion `get_google_results()` aufgerufen wird, wird dieser Inhalt in der Variablen `$content` aufgefangen.
- Der Rest des oberen Script-Teils besteht darin, die bereits beschriebenen Platzhaltervariablen `[%search%]`, `[%offset%]`, `[%content%]`, `[%prev%]`, `[%prevnext%]` und `[%next%]` durch aktuelle Inhalte zu ersetzen. Dabei wird vor allem etwas Aufwand betrieben, um die Blätterfunktion »intelligent« zu machen, d.h., beispielsweise nur dann einen Rückwärtsblättern-Link anzuzeigen, wenn nicht ab dem ersten Suchtreffer ausgegeben wird. Auf die Einzelheiten gehen wir hier nicht mehr näher ein, da dies alles Standard-PHP ist.

14.8.7 Praxisbeispiel: Plausibilitätsprüfung mit WDDX

Gerade bei Webshops ist es wichtig, vom Anwender fehlerfreie Daten bei Adress- und Bankverbindungsangaben zu bekommen. Denn Rücksendungen oder Rücklastschriften kosten Geld. Unter *http://www.ktopruef.de/* hat sich ein Anbieter darauf spezialisiert, solche Plausibilitätsprüfungen als Online-Service zur Verfügung zu stellen. Dazu sind natürlich

Datenaustauschschnittstellen erforderlich. Für Webanwendungen wird eine WDDX-Schnittstelle angeboten.

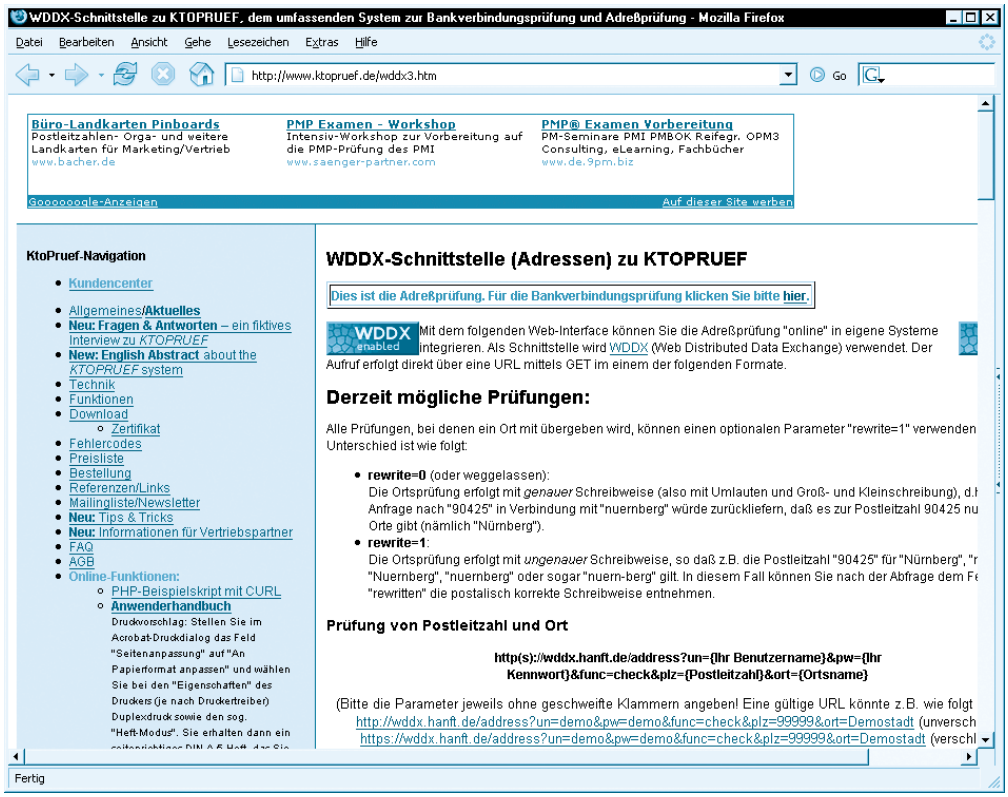


Abbildung 14.8: WDDX-Datenschnittstelle von ktopruef.de

Überprüfbar sind bei diesem Service Bankleitzahlen, Kreditkartennummern sowie Adressangaben (Kombination von Postleitzahlen und Ort, Ort und Straße usw.). Natürlich ist auf diese Weise nicht überprüfbar, ob ein Benutzer in einem Formular seine echte Bankverbindung angegeben hat – wohl aber, ob die angegebene Bankleitzahl existiert.

Der Service von ktopruef.de ist kostenpflichtig. Preise und Details finden Sie auf den Anbieterseiten. Das mag Ihnen zwar ärgerlich erscheinen, doch die meisten heute existierenden Webservices sind kostenpflichtig. Möglicherweise ist das allerdings auch der Grund, warum die Webservices noch nicht wirklich erfolgreich und stärker verbreitet sind.

Damit immerhin ein Test möglich ist, bietet ktopruef.de die Möglichkeit an, eine kostenlose Demoabfrage zu starten. Das genügt, um entsprechende Scripts auszutesten. Auf der Seite *http://www.ktopruef.de/demo.htm* werden die Details beschrieben.

In unserem Beispiel wollen wir eine Adressplausibilitätsprüfung realisieren. Der Service von ktopruef.de funktioniert so, dass die Input-Daten einfach über HTTP-Aufruf als GET-

Parameter an den Service übergeben werden. Dieser liefert das Ergebnis – ebenfalls über HTTP – als WDDX-Dokument zurück. Das auswertende Script muss die Daten des WDDX-Dokuments auslesen und verstehen, um dann entsprechend reagieren zu können.

Zum Testen verwenden wir folgende HTML-Datei mit einem Formular zur Adresseingabe:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Eingabeformular</title>
</head>
<body style="font-size:90%; font-family:sans-serif">
<form action="process.php" method="post"
      style="background-color:#E0E0E0; padding:6px">
<table>
<tr>
<th style="text-align:right">Name:</th>
<td><input type="text" name="name" style="width: 300px"></td>
</tr>
<tr>
<th style="text-align:right">Straße Hausnummer:</th>
<td><input type="text" name="strasse" style="width: 260px">
<input type="text" name="hnr" style="width: 32px"></td>
</tr>
<tr>
<th style="text-align:right">PLZ Ort:</th>
<td><input type="text" name="plz" style="width: 47px">
<input type="text" name="ort" style="width: 245px"></td>
</tr>
<tr>
<th style="text-align:right">Daten:</th>
<td><input type="submit" style="width:100px" value="senden"></td>
</tr>
</table>
</form>
</body>
</html>
```

Das Formular überträgt seine Daten, wenn der Benutzer es absendet, an eine Scriptdatei namens ***process.php***. Deren Inhalt sieht wie folgt aus:

```
<?php
if(!isset($_POST['name']) or
   !isset($_POST['strasse']) or
   !isset($_POST['hnr']) or
   !isset($_POST['plz']) or
   !isset($_POST['ort']))
      die("Eingabedaten unvollständig");
```

```
if(check_plz_ort() == true)
   do_something_else();
else
   die("Eingabedaten nicht plausibel");

function check_plz_ort() {
   $plz = urlencode($_POST['plz']);
   $ort = urlencode($_POST['ort']);
   $url = "http://wddx.hanft.de/";
   $url .= "address?un=demo&pw=demo&func=check&plz=";
   $url .= $plz."&ort=".$ort;
   $wddx_doc = file_get_contents($url);
   $wddx_array = wddx_deserialize($wddx_doc);
   if($wddx_array['result'] != 0)
      return(false);
   else
      return(true);
}

function do_something_else() {
   // Hier könnte die weitere Datenverarbeitung
   // fortgesetzt werden
   echo "Die Eingaben waren in Ordnung";
}
?>
```

Das Script prüft zunächst, ob die via POST übergebenen Formulardaten alle vorhanden sind. Falls nicht, wird mit einer Fehlermeldung abgebrochen.

Als Nächstes wird bereits geprüft, ob die Daten plausibel sind. Dazu wird die Funktion `check_plz_ort()` aufgerufen. Diese ist weiter unten im Script notiert und enthält die gesamte Logik, um die Formulardaten an den Service von ktopruef.de zu übergeben und das per WDDX erhaltene Prüfergebnis auszuwerten.

Zunächst werden die POST-Daten, die überprüft werden müssen, durch Aufruf von `urlencode()` URL-gerecht kodiert. Dann wird die URL-Adressse des Service-Anbieters konstruiert. Der Domain-Name und die zu übergebenden GET-Parameter sind auf den Anbieterseiten dokumentiert. Wir übergeben die aus dem HTML-Formular erhaltenen Werte für Postleitzahl und Ort.

Nachdem die Aufrufadresse feststeht, kann der Aufruf erfolgen:

```
$wddx_doc = file_get_contents($url);
```

Damit wird die Funktion `file_get_contents()` auf die entfernte URL-Adresse angewendet. Das funktioniert in dieser Form nur, wenn PHP entsprechend konfiguriert ist (in der *php.ini* muss die Direktive *allow_url_fopen* den Wert 1 oder true enthalten).

Die aufgerufene Adresse sendet über HTTP ein WDDX-Dokument zurück. Dessen Inhalt ist nach erfolgreichem Einlesen der URL-Adresse in der Auffangvariablen `$wddx_doc` gespeichert.

Angenommen, der Anwender hat im Formular »80331 München« für Postleitzahl und Ort angegeben. Diese PLZ/Ort-Kombination existiert zwar wirklich, doch in der Demoversion von ktopruef.de wird zu Testzwecken einzig und allein die Kombination 99999 Demostadt als »wirklich existent« eingestuft. Alles andere wird als nicht plausible Kombination betrachtet. Der Inhalt des WDDX-Dokuments lautet in unserem »Fehlerfall«:

```
<!DOCTYPE wddxPacket SYSTEM 'wddx_0100.dtd'>
<wddxPacket version='1.0'><header/><data><struct>
<var name='result'><string>3</string></var>
<var name='resulttext'><string>Postleitzahl 80331 gilt als ungültig in der Demo-
Version</string></var>
</struct></data></wddxPacket>
```

Ein solcher Inhalt steht also in unserer Variablen `$wddx_doc`. Nun müssen wir den XML-Inhalt jedoch gar nicht selbst zerlegen, denn PHP bietet von Haus aus die Unterstützung der WDDX-Schnittstelle an. Deshalb genügt ein Aufruf der Funktion `wddx_deserialize ($wddx_doc)`, um die in `$wddx_doc` gespeicherten XML-Daten in einen PHP-Array zu überführen. Dieser Array enthält die `var`-Elemente der WDDX-Daten. Die Werte der `name`-Attribute sind dabei die Array-Schlüssel und der Textinhalt der inneren Datentypelemente sind die zugehörigen Werte. Komplexe WDDX-Variablen werden in Form verschachtelter Arrays abgebildet.

In unserem Fall sieht ein Dump des Arrays, der nach Aufruf von `wddx_deserialize()` aus den obigen WDDX-Daten gebildet wird, wie folgt aus:

```
array(2) {
  ["result"]=>
  string(1) "3"
  ["resulttext"]=>
  string(56) "Postleitzahl 80331 gilt als ungültig in der Demo-Version"
}
```

Dieser Array wird in unserem Beispiel in der Auffangvariablen `$wddx_array` gespeichert. Da wir im Beispiel nur prüfen wollen, ob die Kombination von Postleitzahl und Ort korrekt, also plausibel ist, genügt die Abfrage `if($wddx_array['result'] != 0)`. Denn nur wenn das Array-Element mit dem Schlüssel `'result'` den Wert 0 hat, ist die Kombination korrekt (Einzelheiten dazu müssen Sie der Dokumentation des Anbieters entnehmen).

WDDX-Dokumente erzeugen

Neben der Funktion `wddx_deserialize()` bietet PHP noch weitere WDDX-Funktionen an. Die wichtigste davon möchten wir zum Abschluss kurz vorstellen. Ihre Aufgabe ist der umgekehrte Fall, nämlich aus intern ermittelten Daten selbst ein WDDX-Dokument zu erzeugen. Dies ist also der Fall, wenn Ihr Script der »Service-Server« ist, der von einem anderen, entfernten Script abgefragt wird. Dazu ein kleines Code-Beispiel:

```
<?php
$data['member_id'] = 4503;
$data['name']['given_name'] = "Reiner";
$data['name']['family_name'] = "Wahnsinn";
$data['member_since'] = "2001-06-25";
```

```
echo wddx_serialize_vars("data");
?>
```

Was in diesem Beispiel im verschachtelten Array `$data` steht, sind Werte, die Ihr Script z. B. durch Input, das es beim Aufruf erhalten hat, in einer Datenbank gefunden haben könnte. Das Erzeugen und Senden des WDDX-Dokuments reduziert sich auf den Aufruf der Funktion `wddx_serialize_vars()`. Der Funktion werden Namen von Variablen übergeben (nicht die Variablen selbst!). Sie können beliebig viele Variablennamen übergeben, getrennt durch Kommata. Da wir in unserem Beispiel nur den Array `$data` ins WDDX-Dokument bringen wollen, müssen wir nur dessen Namen, also `"data"` übergeben.

Den Aufruf der Funktion `wddx_serialize_vars()` notieren wir hinter einem `echo`-Konstrukt. Damit wird der von der Funktion erzeugte XML-Code an den aufrufenden Client gesendet. Das obige Script erzeugt folgendes WDDX-Dokument:

```
<wddxPacket version='1.0'>
<header/>
<data>
  <struct>
    <var name='data'>
      <struct>
        <var name='member_id'><number>4503</number></var>
        <var name='name'>
        <struct>
          <var name='given_name'><string>Reiner</string></var>
          <var name='family_name'><string>Wahnsinn</string></var>
        </struct>
        </var>
        <var name='member_since'><string>2001-06-25</string></var>
      </struct>
    </var>
  </struct>
</data>
</wddxPacket>
```

Ein entferntes Script, das unser PHP-Script aufruft, kann diese Daten wiederum parsen und verarbeiten. Wenn das entfernte Script auch ein PHP-Script ist, kann es dazu einfach die Funktion `wddx_deserialize()` verwenden. Das entfernte Script kann aber ebenso gut in Perl, Python, Java oder einer anderen Programmiersprache geschrieben sein. Dank der WDDX-Schnittstelle ist der Datenaustausch über die Grenzen der Programmiersprachen hinweg kein Problem.

Teil 7 – Betrieb von Websites

Professionelle Webseiten zu entwickeln, ist eine Sache – solche Webseiten erfolgreich zu unterhalten, eine andere. Vor allem rechtliche und wirtschaftliche Aspekte gehören dazu, aber auch die Analyse des Benutzerverhaltens und die Kunst, die richtigen Schlüsse daraus zu ziehen. In diesem Abschnitt werden wichtige Themen aus diesem Bereich behandelt.

15 Site-Verwaltung

In diesem Kapitel geht es um die rechtlichen und wirtschaftlichen Aspekte beim Betrieb von Websites.

15.1 Rechte und Pflichten eines Site-Anbieters

Websites sind längst ein vielfältig juristischer Gegenstand geworden. Spektakuläre Streitigkeiten um Domain-Namen, Hyperlinks oder Datensammlungen, neue oder neu gefasste Gesetze wie das Teledienstegesetz (TDG), Abmahnwellen und andere Vorfälle haben den rechtlichen Aspekt von Websites zwar ins Bewusstsein gerückt, doch immer noch herrscht viel Rechtsunsicherheit. Das liegt zum einen daran, dass Internet und World Wide Web einfach viele neuartige Realitäten mit sich gebracht haben, die juristisch erst einmal bearbeitet werden müssen. Zum anderen liegt es aber auch daran, dass viele Website-Anbieter keine existierenden Unternehmen sind, die als definierte Rechtsform auftreten und sich in einem dafür maßgeblichen rechtlichen Rahmen bewegen.

15.1.1 Die Rechtslage

In jedem Fall hat jeder, der Inhalte im Web veröffentlicht, Rechte und Pflichten, egal ob es sich um eine kleine private Homepage, um einen Shop oder um ein großes Portal handelt. Welche Rechte und Pflichten das genau sind, hängt von der Staatsbürgerschaft des Betreibers und von den maßgeblichen Gesetzen und der Rechtspraxis im betreffenden Land ab.

Selbst das ist übrigens eine Errungenschaft, die nicht immer selbstverständlich war. In den Anfangsjahren des Web wurden beispielsweise immer wieder Provider für Dinge belangt, die gar nicht in ihrem Verantwortungsbereich lagen (»du bietest Kunden Internetzugang an, also bist du verantwortlich, wenn irgendwo im Internet Kinderpornos auftauchen«).

Im vorliegenden Kapitel sprechen wir die Rechtslage aus deutscher Sicht an. Wir beschränken uns auf die Rechte und Pflichten von Site-Anbietern. Netzbetreiber und Provider müssen noch andere, hier nicht besprochene Regelungen beachten.



Abbildung 15.1: eRecht24.de, ein Informationsportal für Website-Betreiber

Relevante Rechtsbereiche

Als Betreiber einer Website können Sie sowohl mit dem Strafrecht als auch mit dem Zivilrecht in Konflikt geraten.

Für alle Inhalte, die Sie im Web publizieren oder verantwortlich publizieren lassen, gelten Sie im Rechtsdeutschen als »Verbreiter von Schriften«.

Strafrechtlich können Sie in dieser Lage vor allem folgende Verstöße begehen:

- Verletzung der öffentlichen Ordnung, insbesondere Volksverhetzung (§130 StGB) und Anleitung zu Straftaten (§ 130a StGB)
- Verstoß gegen Pornografiegesetze (§ 184 StGB)
- Beleidigung, üble Nachrede und Verleumdung (§ 185-188 StGB)
- Verstöße im Sinne des Nebenstrafrechts, das sich aus dem Urhebergesetz, dem Markengesetz, dem Patentgesetz, dem Gesetz des unlauteren Wettbewerbs und dem Datenschutzgesetz ergeben kann

Zivilrechtlich können Sie z. B. belangt werden wegen:

- Verstoß in Bezug auf fremdes geistiges Eigentum (fremdes Urheberrecht)

- Verletzungshandlung gegen registrierte Marken und Patente
- Verstoß gegen Regelungen zur vergleichenden Werbung, gegen berufsspezifische Werbeverbote, gegen Preisbindungsregelungen usw.
- Verletzungen im Vertragsrecht (relevant bei Verkaufsabschlüssen, Mietabschlüssen usw. über das Web)
- Verstoß gegen fremde Persönlichkeitsrechte (z.B. durch Rufschädigung, Veröffentlichung privater Schriften anderer, darunter auch E-Mails)
- Verstoß gegen das Produkthaftungsgesetz (Haftung bei Falschinformationen, die nachweislich Schäden verursachen)

Wichtige Gesetze für »Neue Medien«

Als Anbieter von Websites müssen Sie im deutschen Rechtsraum vor allem folgende neueren Gesetze kennen:

- ***Informations- und Kommunikationsdienstegesetz (IuKDG):*** Dieses Gesetz ist maßgeblich für elektronische Angebote zur Information oder Kommunikation, soweit nicht die redaktionelle Gestaltung zur Meinungsbildung für die Allgemeinheit im Vordergrund steht (Datendienste, zum Beispiel Verkehrs-, Wetter-, Umwelt- und Börsendaten, Verbreitung von Informationen über Waren und Dienstleistungsangebote) sowie für Warenangebote und Dienstleistungen in elektronisch abrufbaren Datenbanken mit interaktivem Zugriff und unmittelbarer Bestellmöglichkeit.
- ***Teledienstegesetz (TDG):*** Dieses Gesetz ist ein Auszug aus dem IuKDG und enthält den Artikel 1 daraus. Das Gesetz regelt aus Anbietersicht die beiden wichtigen Punkte Verantwortlichkeit (TDG, § 5) und Anbieterkennzeichnung (TDG, § 6).
- ***Teledienste-Datenschutzgesetz (TDDSG):*** Dieses Gesetz ist ein Auszug aus dem IuKDG und enthält den Artikel 2 daraus. Das Gesetz ist vor allem für Anbieter von Online-Shops wichtig, betrifft aber auch alle Anbieter, die aus anderen Gründen Benutzerdaten sammeln, z.B. um für ein Forum nur registrierte Benutzer zuzulassen. Es regelt die Grundsätze für die Verarbeitung personenbezogener Daten (TDDSG, § 3), datenschutzrechtliche Pflichten des Diensteanbieters (TDDSG, § 4), Bestandsdaten (TDDSG, § 5), Nutzungs- und Abrechnungsdaten (TDDSG, § 6), das Auskunftsrecht des Nutzers (TDDSG, § 7) und die allgemeine Datenschutzkontrolle (TDDSG, § 8). Besonders wichtig ist dabei Absatz 4 des § 3, wo das Prinzip der so genannten »Datensparsamkeit« formuliert wird. In Formularen dürfen Site-Anbieter von Benutzern nur so viele Daten erfassen wie für den Betrieb des Webangebots erforderlich sind. Bei Sites mit Benutzerregistrierung bedeutet das beispielsweise, dass allenfalls Benutzername, Passwort und Mailadresse Pflichtfelder sein dürfen, nicht jedoch Angaben zu Adresse, Telefonnummer, Geburtsdatum oder Ähnlichem.
- ***Mediendienstestaatsvertrag (MdStV):*** Um einen Staatsvertrag und nicht um ein Bundesgesetz handelt es sich dabei deshalb, weil er die kulturelle Länderhoheit betrifft und deshalb ein von den Ländern einzeln unterzeichnetes Dokument darstellt. Die

darin formulierten Paragraphen ähneln in vielen Fällen den Paragraphen aus TDG und TDDSG. Der MdStV enthält aber darüber hinaus noch einige weiterreichende Regelungen. So wird in § 6 die Impressumspflicht explizit formuliert und in § 7 die Verpflichtung zur wahrheitsgemäßen Wiedergabe von Information und die Sorgfaltspflicht bei der Recherche. In § 9 wird für die Werbung festgelegt, dass diese als solche kenntlich gemacht werden muss, und § 10 postuliert das Prinzip der Gegendarstellung. Dieses verpflichtet den Anbieter, Widerspruch von Lesern/Anwendern gegen eine Tatsachenbehauptung der eigenen Redaktion zu veröffentlichen.

Problematisch ist in vielen Fällen die Einordnung einer Site. Für Online-Ausgaben von bekannten Zeitungen oder Magazinen sind zweifellos alle genannten Gesetze maßgeblich. Die folgenden Einordnungen gelten als einigermaßen sicher.

Teledienste sind:

- Webbasierte Shops
- Diskussionsforen, Mailinglisten und Ähnliches
- Datendienste wie Wetter-Services, Land- und Stadtkartendienste, Börseninformationsdienste usw.
- Online-Spiele-Sites
- Suchmaschinen und Linkverzeichnisse

Mediendienste sind:

- Broadcasting-Sender, die unter anderem das Web für Services wie Teleshopping benutzen
- Verteildienste wie Nachrichtenagenturen
- Öffentliche Textdienste wie Videotext
- Abrufdienste; dies ist wohl die problematischste Kategorie, da hierunter alle Angebote fallen, die Inhalte in Wort, Bild oder Ton für ein breites Publikum anbieten. Der Unterschied zum Teledienst wird damit begründet, dass das Zielpublikum die breite Masse ist, während Teledienste auf ein Individualpublikum zielen. Die Grenzen sind jedoch zweifellos fließend.

Weiterführender Link:

Gesetzessammlung (Gesetzestexte) Online-Recht:
http://www.netlaw.de/gesetze/

15.1.2 Impressumspflicht

Hinter dem Schlagwort »Impressumspflicht« verbirgt sich der juristische Begriff der »Anbieterkennung«. Seit 2002 ist jede Website verpflichtet, auch eine Anbieterkennung, also ein Impressum, zu veröffentlichen. Basis ist $ 6 des Teledienstgesetzes (TDG) in seiner Fassung vom 21.12.2001.

Ein Impressum muss folgende Daten enthalten:

- Name und Anschrift des Betreibers. Bei juristischen Personen wie Gesellschaften, Vereinen usw. sollten die geschäftsführenden Gesellschafter bzw. Vertretungsberechtigten namentlich genannt werden. Bei der Anschrift muss es sich um eine echte Adresse handeln. Postfachangaben sind nicht zulässig.
- Angaben, die eine schnelle elektronische Kontaktaufnahme und unmittelbare Kommunikation mit dem Betreiber ermöglichen. Das bedeutet in der Praxis vor allem die Angabe einer gültigen und vom Betreiber tatsächlich benutzten Mailadresse. Auch andere Kommunikationsadressen wie Messenger-Nummern usw. sind zusätzlich möglich.

Dies sind die Mindestangaben, die selbst eine private Homepage erfüllen muss, um den Anforderungen der Anbieterkennung zu genügen. Darüber hinaus bestehen je nach Gegebenheiten folgende Auskunftspflichten im Impressum:

- Falls das Website-Angebot Teil einer Tätigkeit ist, die einer behördlichen Zulassung oder Aufsichtspflicht unterliegt, muss die zuständige Behörde genannt werden.
- Wenn der Betreiber im Handelsregister, Vereinsregister, Partnerschaftsregister oder Genossenschaftsregister eingetragen ist, muss er angeben, in welchem Register er unter welcher Registernummer eingetragen ist.
- Wenn das Website-Angebot Teil einer beruflichen Tätigkeit ist und der Beruf einen gesetzlich geregelten Ausbildungsabschluss oder Ähnliches erfordert, muss darauf hingewiesen werden. Das heißt, Titel, genaue Berufsbezeichnung usw. müssen angegeben werden.
- Wenn der Betreiber über eine Umsatzsteueridentifikationsnummer verfügt, muss er die Nummer angeben.

Je nach Berufsstand oder Status können noch weitere Angaben erforderlich sein. Im Zweifelsfall gilt: lieber eine Angabe zu viel als eine zu wenig.

Geschäftsmäßige Teledienste

Das Teledienstgesetz (TDG) spricht – nicht nur in Zusammenhang mit der Anbieterkennung – immer wieder von »geschäftsmäßigen Telediensten«. Die Auslegung von »geschäftsmäßig« ist dabei sehr weit zu verstehen. Nicht nur unmittelbar kommerziell orientierte Webangebote wie Shops fallen darunter, sondern auch jede Art von Angebot, das die Bekanntheit des Anbieters steigert und ihm damit sekundär geschäftlichen Nutzen bringen kann. Es muss nicht einmal eine Gewinnabsicht bestehen. Daher werden auch Vereine oder viele Gesellschaften des bürgerlichen Rechts als »geschäftsmäßig« bewertet.

Erreichbarkeit des Impressums

Laut Gesetz muss die Anbieterkennung leicht erkennbar, unmittelbar erreichbar und ständig verfügbar sein. In der Praxis bedeutet das, dass die Impressumsseite auf jeder Webseite eines Webangebots direkt verlinkt sein sollte. Da die meisten Websites über

irgendeine Form von fester Navigation verfügen, sollte das Impressum am besten ein eigener Punkt in der Navigationsstruktur sein. Wenn die Navigation auf JavaScript basiert, sollte sichergestellt werden, dass Links zum Impressum auch ohne JavaScript funktionieren.

Auch der Fußbereich einer Webseite kann dazu dienen, einen Link zum Impressum unterzubringen. Ein solcher Link ist allemal sinnvoller als die auch hierzulande üblichen, aber juristisch überhaupt nicht relevanten ©-Angaben. Das geistige Urheberrecht greift ohnehin und das amerikanische Copyright, welches durch das ©-Zeichen ausgedrückt wird, ist bei deutschen Sites unmaßgeblich.

Links zu Impressums im Web

Angela Merkel: *http://www.angela-merkel.de/impressum/30.htm*
Der SPIEGEL: *http://www.spiegel.de/extra/0,1518,162848,00.html*
TAZ: *http://www.taz.de/pt/.etc/nf/impressum*
Wikipedia: *http://de.wikipedia.org/wiki/Wikipedia:Impressum*
Adalbert-Stifter-Verein: *http://www.asv-muen.de/de/main/impressum.htm*
Deutsches Forschungsnetz: *http://www.dfn.de/content/de/impressum/*
Firma Putzmunter: *http://www.putzmunter-berlin.de/impress.htm*
Wolfgang Wiese: *http://www.xwolf.de/impressum.shtml*

15.1.3 Vorvertragliche Widerrufsbelehrung

Während die Allgemeinen Geschäftsbedingungen (AGB) von allen Anbietern, die mit einer Website unmittelbar eine Geschäftstätigkeit ausüben (z. B. Shop-Betreiber), meist ohne Aufforderung bereitgestellt werden, ist dies bei der so genannten »vorvertraglichen Widerrufsbelehrung« nicht so selbstverständlich. Diese gehört jedoch zu den Pflichten eines jeden Webshop-Betreibers, auch dann, wenn es sich nur um nebenberuflich betriebenes Kleingewerbe handelt.

Im Gegensatz zu den Allgemeinen Geschäftsbedingungen, die in aller Regel aufrufbar sind, dem Anwender aber nicht vor Abschluss eines Vertrags aufgezwungen werden, ist das bei der vorvertraglichen Widerrufsbelehrung Vorschrift. Diese muss dem Kunden, bevor er eine Bestellung endgültig tätigt, zwangsweise vorgesetzt werden, ähnlich, wie bei Softwareinstallationen eine Dialogseite den Lizenzvertrag enthält, der meist mit einer expliziten Aktion (z. B. Anklicken eines Radio-Buttons »ich akzeptiere«) verbunden ist.

Basis ist vor allem § 312 des BGB. Danach hat ein Unternehmer den Verbraucher rechtzeitig vor Abschluss des Fernabsatzvertrages klar und verständlich über dessen Widerrufsrechte zu informieren. Dabei ist keine Widergabe von Gesetzestexten verlangt, sondern einfach der Hinweis darauf, dass der Kunde das Recht hat, vom Kaufvertrag innerhalb einer gesetzlich geregelten Frist (derzeit zwei Wochen) ohne Angabe von Gründen zurückzutreten.

Diese Information muss dem Benutzer eines webbasierten Shops, der gerade dabei ist, eine Ware zu bestellen, aufgezwungen werden. Zwischen Warenkorbübersicht, Eingabe

von Lieferadresse, in jedem Fall aber vor der endgültigen Kaufbestätigung muss dieser Hinweis angezeigt werden, und zwar so vordergründig, dass der Kunde praktisch keine Möglichkeit hat, den Hinweis zu ignorieren. Der Rechtssprechung zufolge sollte der Hinweis auf derjenigen Webseite stehen, auf Grund derer sich der Kunde zum Kauf entschließt.

Wortlautvorschlag

Ein Hinweis zur Widerrufsbelehrung könnte folgenden Text haben:

```
Widerrufsbelehrung - Widerrufsrecht:
Sie können den hier zustande kommenden Vertrag innerhalb von zwei Wochen ohne
Angabe von Gründen in Textform (z.B. Brief, Fax, E-Mail) oder durch Rücksendung
der Sache widerrufen. Die Frist beginnt frühestens mit Erhalt dieser Belehrung.
Zur Wahrung der Widerrufsfrist genügt die rechtzeitige Absendung des Widerrufs
oder der Sache. Der Widerruf ist zu richten an:
[Name/Anschrift]
Sie können den Widerruf auch per Telefax unter der Nummer [Faxnummer] oder per
E-Mail unter [Mailadresse] erklären.
Können Sie uns die erhaltene Ware nicht oder nur in verschlechtertem Zustand
zurückgeben, müssen Sie uns insoweit ggf. Wertersatz leisten. Dies gilt nicht,
wenn die Verschlechterung ausschließlich auf eine Warenprüfung - wie etwa auch
im Ladengeschäft üblich - zurückzuführen ist.
Paketversandfähige Waren können Sie auf unsere Kosten und Gefahr an die folgende
Adresse zurückzusenden:
[Name/Anschrift]
Nicht paketversandfähige Waren werden bei Ihnen abgeholt.
```

Bei Internetauktionen, etwa in eBay, muss zwischen privaten Versteigerungen (»alte Gegenstände loswerden«) und geschäftsmäßig betriebenen Versteigerungen unterschieden werden. Die Grenzen können dabei durchaus fließend sein, etwa wenn Erbgut im größeren Stil versteigert wird. Auf jeden Fall unterliegt jeder, der bei eBay einen echten Shop betreibt, in vollem Umfang den Regelungen des Fernabsatzgesetzes und damit auch der Vorschrift der vorvertraglichen Widerrufsbelehrung.

15.1.4 Urheberrecht

Websites unterliegen in mehrfacher Hinsicht dem Schutz geistigen Eigentums:

- Websites gelten als Medium zur Verbreitung von Wort, Bild und Ton, je nachdem, was alles enthalten ist. Alles, was darunter fällt, unterliegt dem Urheberrecht.
- Websites gelten als Computerprogramm und sind als solche urheberrechtlich geschützt.
- Websites können je nach Inhalt und Präsentationsform auch als individuelle Datensammlung gelten und sind als solche urheberrechtlich geschützt.

Um Urheberschutz zu genießen, reicht es, wenn ein »Werk« vorliegt, das in eigener geistiger Leistung erstellt wurde. Zwar wird der Begriff »Werk« zunächst auf Werke der Kunst

angewendet, wie literarische, bildnerische oder musikalische Werke, doch ist der Begriff längst ausgedehnt worden, unter anderem auch auf Computerprogramme und Datensammlungen. Entscheidend für die Einstufung als »Werk« ist jedoch letztlich die so genannte Schöpfungshöhe, die im Falle eines Rechtsstreits fallspezifisch bewertet werden muss.

Das Urheberrecht muss nirgends beantragt oder eingetragen werden. Es gilt automatisch.

Das Urheberrecht gilt nur für tatsächlich realisierte Werke. Für bloße Ideen (z.B. Designideen) gilt es nicht.

Eigene und fremde Urheberschaft

Wer HTML, CSS, Flashmovies, PHP-Scripts, Datenbanken usw. völlig autonom und selbständig entwickelt und dabei auch keine fremden Bilder, Texte oder Sounds verwendet, ist geistiger Urheber der so geschaffenen Website.

Wird eine Website von mehreren Personen erstellt, können diese als Miturheber alle über die Verwertungs- und Veröffentlichungsrechte bestimmen. In vielen Fällen ist es allerdings so, dass ein Auftraggeber seine Mitarbeiter vertraglich verpflichtet, ihm die Nutzungsrechte am fertigen Werk zu überlassen. Deshalb haben Firmenmitarbeiter, aber auch freie Webdesigner und Webagenturen in der Regel keine Nutzungs- oder Verwertungsrechte an den von ihnen in Auftragsarbeit erstellten Webseiten, obwohl sie nach wie vor als deren Urheber gelten.

Eine Website genießt nicht nur Urheberrecht, sondern kann auch ganz leicht gegen fremdes Urheberrecht verstoßen:

- Jede vom Originalurheber nicht explizit genehmigte Verwendung von fremden Texten, Bildern oder Sounds stellt einen Verstoß gegen das Urheberrecht dar.
- Das Einbinden von Ressourcen von fremden Servern über absolute URI-Adressierung z.B. in img- oder frame-Elementen stellt einen Verstoß gegen das Urheberrecht dar, sofern der Urheber dieser Nutzung nicht ausdrücklich zugestimmt hat.
- Das Verwenden von Code aus fremden Scripts (egal ob clientseitig wie JavaScript oder serverseitig wie PHP) stellt einen Verstoß gegen das Urheberrecht dar, sofern der Originalentwickler keine ausdrückliche Genehmigung zur Verwendung des Codes erteilt hat.
- HTML- und CSS-Code gilt an sich nicht als schützenswert, wohl aber das Design einer Website.

Nachweis der Urheberschaft

Bei elektronischen Daten ist der Nachweis der Urheberschaft oft nicht so einfach zu erbringen wie bei Werken, die über einen Verlag oder eine Agentur publiziert werden. So kam es in der Vergangenheit bei Rechtsstreitigkeiten um »Designklau« schon zu gegenseitigen Vorwürfen der Parteien, weil die Klägerseite gar nicht beweisen konnte, dass ihr

Werk zeitlich zuerst entstanden war. Dateistempel sind dabei unerheblich, da sie sich leicht manipulieren lassen.

Eine mögliche Vorkehrmaßnahme besteht darin, die Daten einer fertig erstellten Website auf einen nicht mehr weiter beschreibbaren Datenträger wie CD/DVD-ROM zu brennen und sich diesen selber per Post als Einschreiben zuzusenden. Der Einschreibbeleg gilt dann als »Zeitstempel«.

15.1.5 Haftung für Inhalte und Links

Ein weiteres wichtiges Thema für Anbieter von Websites ist das der Haftung. Vor allem zwei Arten von Haftung stehen im Vordergrund:

- Haftung für Inhalte
- Haftung für Links

Daneben können je nach Gegenstand der Website andere Arten von Haftung relevant sein, wie z. B. die Produkthaftung.

Haftung für Inhalte

In diesem Zusammenhang besteht vor allem die Gefahr des Verstoßes gegen strafrechtliche Paragraphen. Wer auf Webseiten zu Straftaten oder Volksverhetzung aufruft, wird belangt werden. Problematisch kann die Lage auch sehr schnell für Anbieter von Erotikinhalten werden. Wer solche Inhalte Personen unter 18 Jahren offen zugänglich macht, kann belangt werden. Wer sicher gehen will, ist also verpflichtet, solche Inhalte nur gegen Registrierung und vorherigem Altersnachweis zugänglich zu machen.

Am einfachsten gerät man als Inhaltsanbieter jedoch in Gesetzeskonflikte mit den Paragraphen 185 bis 188 des Strafgesetzbuches, welche sich mit Beleidigung, übler Nachrede und Verleumdung befassen. Personen des öffentlichen Lebens müssen schon mal damit rechnen, empfindlich kritisiert zu werden. Bei anderen Personen, etwa solchen, die sich einfach nur durch die Art ihres Auftretens in einem Diskussionsforum unbeliebt gemacht haben, oder bei unbeliebten Personen wie etwa Abmahnanwälten können ein paar unbedachte Äußerungen schnell einen Strafantrag zur Folge haben.

Die meisten professionellen Websites werden von fähigen Autoren oder Redakteuren betreut, weshalb die selbst erstellten Inhalte weniger Gefährdung bedeuten. Brisant wird es jedoch in dem Augenblick, wo eine Website nicht nur Inhalte der eigenen Autoren oder Redakteure anbietet, sondern auch von allen oder vielen Besuchern. Das ist der Fall bei Board- oder Forensystemen, bei Gästebüchern, bei Kommentarmöglichkeiten zu Artikeln oder bei Wiki-Systemen.

Inhalte, die von Seitenbesuchern ohne redaktionelle Kontrolle ins Web gestellt werden können, muss der Anbieter sofort, nachdem er Kenntnis von einem strafrechtlich relevanten Inhalt erlangt hat, löschen. Ein Anbieter kann also nicht um 5 Uhr früh aus dem Bett gezerrt und verhaftet werden, weil um 3 Uhr früh ein Besucher volksverhetzende Naziparolen in seinem Forum gepostet hat. Ruft er jedoch am Vormittag seine Mails ab und

findet dort eine Mail vor, in der ein anderer Besucher auf das problematische Posting hinweist, ist er verpflichtet, sofort zu reagieren. Die Frage, innerhalb welchen Zeitraums problematische Fremdinhalte von der eigenen Website gelöscht werden müssen, ist letztlich eine Ermessensfrage. Bei einer Website, die von einem 20-köpfigen Redaktionsteam betreut wird, können andere Maßstäbe angelegt werden als bei einem privaten Homepage-Betreiber. Wer eine Site als Alleinredakteur anbietet, sollte jedoch sicherheitshalber Foren und Gästebücher deaktivieren oder die Verantwortung dafür an andere delegieren, bevor er vier Wochen in einen internetfreien Urlaub fährt.

Haftung für Links

Das Thema Linkhaftung kochte in den vergangenen Jahren besonders hoch und sorgte für einige Verwirrung. Grund waren zweifelhafte Abmahnwellen und ebenso zweifelhafte Gerichtsentscheidungen, die in offensichtlicher Unkenntnis von der Natur von Hyperlinks gefällt worden waren.

Zweifelsfrei kann jeder, der Links auf illegale Inhalte anbietet, z. B. auf Ziele mit eindeutig strafrechtlichem Inhalt wie etwa auf Warez-Seiten mit Registriercodes lizenzpflichtiger Software, dafür haftbar gemacht werden. Vor einigen Jahren tendierten die Gerichte jedoch dahin, einen Link als »Zueigenmachen« der verlinkten Inhalte zu betrachten, so als würde das verlinkte Ziel durch den Link zum Teil des eigenen Webangebots. Demzufolge wurde der Linkanbieter zum Nebenbeklagten, sobald eine Klage gegen die verlinkte Site anhängig war.

Eine solche juristische Betrachtungsweise macht jedoch das Setzen externer Links zu einem völlig unkontrollierbaren Risiko und führt zum Aussterben externer Links und damit zum Aussterben des gesamten Webgedankens. Das haben aber mittlerweile auch die deutschen Gerichte eingesehen.

Deeplinks

Eine Zeitlang waren auch Links in die Tiefen eines fremden Webangebots bedroht, weil einige Anbieter meinten, man dürfe nur auf ihre Hochglanzstartseite verlinken statt z. B. direkt auf eine bestimmte Produktseite. Ein Urteil des Bundesgerichtshofs (BGH) aus dem Jahr 2003 schuf hier jedoch Rechtssicherheit. Deeplinks dürfen gesetzt werden.

Weiterführender Link:

Telepolis-Artikel »Linkhaftung: Gesetzgeberische Untätigkeit schafft endlich Klarheit«:
http://www.heise.de/tp/r4/artikel/12/12721/1.html

15.1.6 Datenschutz und Auskunftspflicht

Solange eine Website nur aus öffentlich aufrufbaren Seiten besteht, sammelt sie keine personenbezogenen Daten. Anders sieht die Sache jedoch ab dem Moment aus, wo Sie beispielsweise ein Gästebuch anbieten, in dem Anwender nur etwas eintragen können, wenn sie persönliche Daten wie Namen und E-Mail-Adresse hinterlassen, oder wenn die

Anwender die Möglichkeit haben, einen Newsletter zu abonnieren, der vom Site-Betreiber allen abonnierten Anwendern in regelmäßigen oder unregelmäßigen Abständen zugeschickt wird. Auch bei registrierungspflichtigen Angeboten wie Foren, Boards oder anderen Inhalten werden personenbezogene Daten gespeichert.

Unterschieden werden muss indes zwischen Daten, die eine persönliche Identifizierung erlauben, und solchen Daten, die sich nicht einer identifizierbaren Person zuordnen lassen. E-Mail-Adressen, reale Namen, Anschriften, Telefonnummern usw. erlauben eine persönliche Identifizierung. Benutzt eine Website dagegen beispielsweise Cookies, um das Angebot für einen bestimmten Browser, der das frühere Cookie beim erneuten Besuch wieder sendet, zu individualisieren, dann handelt es sich um persönliche Daten, auf Grund deren jedoch keine konkrete Person identifizierbar ist.

Auskunftspflicht

Ein Anwender kann von einem Site-Betreiber verlangen, ihm offen zu legen, welche personenbezogenen (genauer: personenidentifizierenden) Daten dieser von dem Anwender speichert, zu welchem Zweck er das tut und ob und in welcher Form er diese Daten an Dritte weitergibt. Reagieren Sie als Site-Betreiber nicht auf ein solches Verlangen, hat der Anwender gute Chancen auf eine erfolgreiche Klage.

Mögliche problematische Fälle

Personenbezogene Daten sind nicht nur typische Kontaktdaten wie Namen, Anschriften, Telefonnummern und Mailadressen, sondern auch Angaben zu »persönlichen Verhältnissen«. Ein »persönliches Verhältnis« in diesem Sinne ist z.B. die Zugehörigkeit zu einer Organisation. Problematisch können daher Website-Inhalte wie die folgenden werden:

- Eine Schule veröffentlicht eine Liste ihrer Lehrer.
- Ein Abiturjahrgang veröffentlicht eine Liste der Schulabgänger.
- Eine Firma veröffentlicht eine Liste ihrer Mitarbeiter.
- Eine Community-Site veröffentlicht eine Liste ihrer Mitglieder.

Wenn eine ausdrückliche Genehmigung der betroffenen Personen eingeholt wird, schützt das nur bedingt. Denn erstens kann sich der Wille der Person jederzeit ändern und zweitens ist es denkbar, dass die Person einer öffentlichen Nennung in einem bestimmten Zusammenhang zustimmt, der Nennung in einem anderen Zusammenhang dagegen nicht.

15.1.7 Namens- und Markenschutz

Die bekanntesten Rechtsstreitigkeiten bezüglich Websites sind zweifellos Streitigkeiten um Domain-Namen. Aber auch die Verwendung von geschützten Namen und Marken im HTML-Code kann schnell zu einer teuren Abmahnung führen.

Domain-Namen, fremde Marken und eigene Marke anmelden

Bei den NIC-Vergabestellen für Domain-Namen gilt die Devise »wer zuerst kommt, ...«. Diese Laxheit ermöglicht einerseits ein rasches Abwicklungsverfahren bei Anträgen auf neue Domains. Andererseits gelten Domain-Namen jedoch als namens- und markenrechtlich relevante Bezeichnungen. In der Praxis führt dieser Widerspruch häufig zu Rechtsproblemen. Denn mit der Registrierung eines neuen de-Domain-Namens durch die DENIC ist dieser keineswegs vor namens- und markenrechtlichen Ansprüchen geschützt.

Zwar hat sich die Lage mittlerweile etwas entschärft, da inzwischen fast alle Unternehmen, die auf einen überregional geschützten Namen wert legen, unter diesem bereits im Internet erreichbar sind. Doch nicht selten sind die Namen nur für die de-Topleveldomain reserviert, während sie unter Topleveldomains wie *.net*, *.com* oder *.org* noch frei sind.

Wenn professionelle Websites mitsamt Domain neu eingerichtet werden, empfiehlt es sich in jedem Fall, im Vorfeld einen Anwalt mit einer Namens- und Markenrecherche für den oder die gewünschten Namen zu beauftragen. Die damit verbundenen Kosten sind auf jeden Fall geringer als ein teurer Rechtsstreit, bei dem man am Ende noch seine bereits beim Publikum eingeführte Domain verliert.

Noch etwas teuerer, aber auch sicherer ist es, gleich selber eine Marke auf den gewünschten Domain-Namen anzumelden. Denn jede Person kann eine Marke anmelden – dazu ist kein Unternehmen erforderlich. Marken können bei den zuständigen Marken- und Patentämtern beantragt werden. In Deutschland gilt, dass eine Marke verfällt, wenn sie fünf Jahre nach Anmeldung nicht am Markt benutzt wird. Der Betrieb einer Website unter einem als Marke angemeldeten Namen gilt jedoch als Markenbenutzung am Markt. In jedem Fall muss eine Marke alle zehn Jahre gegen Gebühr verlängert werden.

Die Anmeldung einer Marke bietet zwar erhöhte Sicherheit. Die Durchsetzbarkeit einer Marke in einem Rechtsstreit hängt allerdings stark von der so genannten Kennzeichnungskraft der Marke ab. Der Markenschutz ist also durchaus eine relative Angelegenheit: Je stärker die Bekanntheit und Originalität einer Marke, desto leichter lässt sich der Schutz der Marke durchsetzen.

Für Domain-Namen muss eine Wortmarke angemeldet werden (auch grafische Symbole oder Tonfolgen können als Marken angemeldet werden). Die Marke identifiziert Produkte und Dienstleistungen. Es kann ein Eigenname sein, aber auch ein Slogan oder eine Wortkombination. Ideal sind neuartige und dennoch leicht eingängige Wortkombinationen oder Wortspiele. Wenn das Budget es erlaubt, empfiehlt es sich durchaus, professionelle Unterstützung durch Werbetexter bei der Namenswahl in Anspruch zu nehmen.

Namens- und Markenrecht bei Webseiteninhalten

Bei der Nennung fremder Marken im Fließtext empfiehlt es sich zwar, darauf hinzuweisen, dass es sich bei dem Namen um eine angemeldete Marke handelt. Allerdings lässt sich das kaum bei allen Erwähnungen durchhalten, schon gar nicht, wenn die Site-Inhalte auch von Benutzern stammen, etwa in Foren, Boards oder Wikis.

Generell gilt: Missbrauch mit einer fremden Marke findet dann statt, wenn der Eindruck entsteht, dass die fremde Marke für eigene Zwecke benutzt wird. Eine gewöhnliche Erwähnung in einem Text ist daher in aller Regel unkritisch. Problematisch wird es hingegen schon, wenn fremde Markennamen in den Meta-Tags *description* und *keywords* der eigenen Seiten auftauchen. Dies hat in der Vergangenheit schon Abmahnungen zur Folge gehabt. Dem Webseitenanbieter wurde unterstellt, durch Verwendung der fremden Marke in diesen für Suchmaschinen prominenten HTML-Elementen die eigene Site zu pushen.

Vorsicht geboten ist auch bei Fanseiten. So erhielt schon manch ein Liebhaberseitenanbieter eine Abmahnung, weil er im Web abfotografierte Figuren aus Schokoladeneiern zeigte und eine Tauschbörse dafür betrieb, von der Firma, die eigentlich ein Interesse an solchen Fan-Seiten haben sollte. Wenn Sie also eine Fansite für eine prominente Person, ein kultiges Produkt, eine allseits beliebte Science-Fiction-Serie oder Ähnliches planen, dann empfiehlt es sich, vorher mit den entsprechenden Agenturen oder Firmen Kontakt aufzunehmen und das Vorhaben zu schildern. Allerdings bedeutet eine positiv verlaufene Kontaktaufnahme noch keinen Freifahrtschein.

15.1.8 Softwarepatente

Das Informationszeitalter brachte es mit sich, dass Software immer mehr wirtschaftliche Bedeutung zukam. So konnte 1980 in den USA erstmals ein Patent auf Software angemeldet werden. Im europäischen Raum gab es zwar bislang keine vergleichbare Patentregelung. Software galt als mathematikartig und algorithmischer Natur. Dennoch gibt es bereits etliche existierende europäische und deutsche Patente, die man als Softwarepatente bezeichnen muss.

Nutznießer von Softwarepatenten sind in erster Linie große Softwarehäuser, die genügend Mittel haben, um die Patente zu finanzieren, und die ihre Softwareprodukte davor schützen wollen, dass andere, preiswertere Software oder gar OpenSource-Software die gleiche Technik oder das gleiche Anwender-Handling bietet. Darum verwundert es nicht, dass so viele Entwickler und Computerfachleute Sturm laufen. Zum Redaktionszeitpunkt dieses Buches war die Frage, ob Softwarepatente in Europa künftig explizit zulässig sein sollen oder nicht, noch offen. Aktuelle Informationen bietet der FFII (Förderverein für eine Freie Informationelle Infrastruktur) unter der Adresse *http://swpat.ffii.org/index.de.html*.

Reale Beispiele

Das wohl bekannteste Beispiel ist das 1-Click-Patent, eingereicht vom Online-Buchriesen Amazon, das seit März 1999 beim Europäischen Patentamt als gültiges Patent angemeldet ist. Die Idee dahinter: Der Kunde kann Produkte mit einem Mausklick einkaufen, weil er der Website bereits früher all seine Daten – Lieferanschrift, Zahlmethode usw. – bekannt gemacht hat. Das Beispiel zeigt aber auch die typischen Probleme: Technisch gesehen basiert die 1-Click-Idee auf gängigen Features wie Cookies oder Sessions, die etwa vom HTTP-Protokoll oder von einer verwendeten serverseitigen Programmiersprache unter anderem zu eben solchen Zwecken zur Verfügung gestellt werden. Die Folge des Patents



Abbildung 15.2: Softwarepatente, Informationsportal des FFII

ist jedoch, dass Amazon von jedem Shop-Betreiber im Web, der seinen Kunden ebenfalls auf diese Weise das Einkaufen erleichtern möchte, Lizenzgebühren verlangen kann.

Mindestens ebenso erschreckend ist das Beispiel für Lernsoftware, für die seit 1996 ein Patent in Europa und in den USA existiert. Das Patent beschreibt den eigentlich trivialen und selbstverständlichen Prozess, dass eine Lernprüfung elektronisch erstellt und zum Prüfling gelangen kann, dass dessen Übungsergebnisse in einer Datenbank gespeichert werden und sich in Bezug auf die Prüfungsergebnisse Statistiken realisieren lassen. Damit ist praktisch jede Webanwendung, die Lernplattformen bereitstellt, potenzieller Lizenznehmer des Patentinhabers.

Fazit

Angesichts der zahlreichen bereits existierenden Patente, die als »verdeckte Softwarepatente« gelten können, ist die Frage, ob Softwarepatente offiziell zulässig sein sollen oder nicht, nur ein Aspekt. Bereits jetzt können Website-Anbieter von völlig unerwarteter Seite Abmahnungen wegen nicht lizenzierter Verwendung einer Softwareidee erhalten. Website-Anbieter schwimmen, was Softwarepatente betrifft, in einem trüben Tümpel voller Krokodile.

15.1.9 Abmahnungen

In vielen Fällen beginnen juristische Probleme für Website-Betreiber damit, dass sie eine Abmahnung erhalten. Eine Abmahnung ist eine Aufforderung, eine nach Ansicht des Abmahnenden bestehende Rechtsverletzung zu unterlassen.

Im günstigsten Fall verfasst der Geschädigte die Abmahnung selber, da in diesem Fall keine Kosten entstehen. Teuer kann es jedoch werden, wenn sofort ein Anwalt eingeschaltet wird. Dieser berechnet dem Abgemahnten nämlich ohne weiteres seine entstandenen Kosten, die sich oft an dem von ihm festgesetzten Streitwert des Schadens orientieren. So können problemlos vierstellige Summen entstehen, die zusammen mit der Abmahnung gefordert werden.

Noch teurer kann es allerdings werden, wenn man aufgrund des persönlichen Rechtsempfindens eine Abmahnung ignoriert. In diesem Fall kann der Abmahnende die in der Abmahnung angedrohten gerichtlichen Schritte unternehmen. Diese bestehen in der Regel darin, dass der Abmahnende versucht, bei Gericht eine so genannte einstweilige Verfügung zu erwirken. Das Gericht prüft den Antrag und falls es ihm statt gibt, gilt die vermeintliche Rechtsverletzung als vorläufig angenommen. Der ursprünglich Abgemahnte wird nun gerichtlich gezwungen, die Rechtsverletzung zu unterlassen. Andernfalls ist eine Vollstreckung die Folge.

Einen vermeintlichen Rechtsverletzer abzumahnen, ist keinesfalls Pflicht. Wer sich geschädigt fühlt, kann auch ohne Umweg über die Abmahnung gleich gerichtlich eine einstweilige Verfügung beantragen. Insofern sind Abmahnungen eigentlich ein sinnvolles Instrument der Rechtspraxis, um Rechtsverletzungen im Vorfeld außergerichtlich zu regeln. Dass das Abmahnwesen so in Verruf geraten ist, verdanken wir gewissen Rechtsanwälten, die sich auf Abmahnungen in Serie spezialisiert haben und sich mit den Kostennoten eine goldene Nase verdienen. Dank moderner Mittel wie Internet-Suchmaschinen ist es einfach, rechtsverletzende Verwendungen etwa von Marken zu finden.

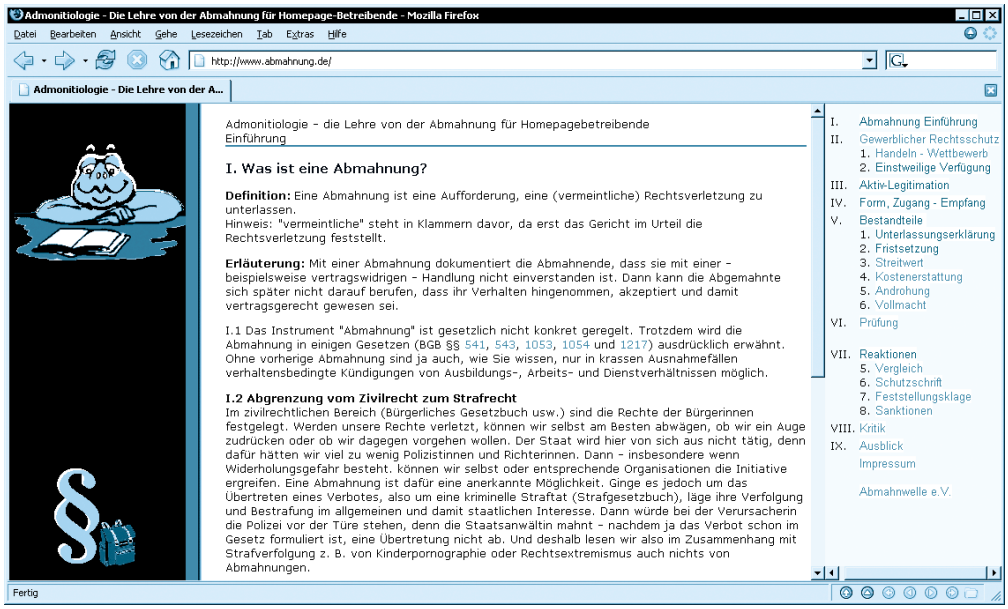


Abbildung 15.3: Alles über Abmahnungen – auf abmahnung.de

Formanforderungen an Abmahnungen

Eine Abmahnung muss Folgendes enthalten:

- Eine präzise und nachvollziehbare Beschreibung der vermeintlichen Rechtsverletzung mitsamt Hinweis auf Gesetze, die anwendbar sind,
- eine explizite Aufforderung, die Rechtsverletzung zu unterlassen,
- eine Aufforderung, zusätzlich eine Unterlassungserklärung gegenüber dem vermeintlich Geschädigten abzugeben,
- eine angemessene Frist, bis zu der die Rechtsverletzung beseitigt und die Unterlassungserklärung abgegeben sein muss,
- eine Angabe zum angenommenen Streitwert,
- die Androhung gerichtlicher Schritte bei Nichtbeachtung der Abmahnung,
- eine Unterschrift.

Wenn die Abmahnung von einem Anwalt kommt, enthält sie zusätzlich meist noch die berüchtigte Kostennote des Anwalts sowie eine Vollmacht des Mandanten.

Wenn eine Abmahnung die genannten Bestandteile nicht enthält, kann sie als formal mangelhaft zurückgewiesen werden. Auf eine anwaltliche Abmahnung so zu reagieren, kann jedoch gefährlich enden. Ein Hinweis noch: Die Abmahnung muss keinesfalls das Wort »Abmahnung« enthalten!

Falls Sie sich selber mal gezwungen sehen, eine Abmahnung zu verfassen (also ohne einen Anwalt einzuschalten), müssen Sie natürlich die genannten Punkte ebenfalls beachten.

Reaktionsmöglichkeiten auf Abmahnungen

Wenn Sie die Rechtsverletzung als gerechtfertigt betrachten, geben Sie in jedem Fall innerhalb der in der Abmahnung genannten Frist die Unterlassungserklärung ab und beseitigen Sie innerhalb der eigenen Website die Ursache der Abmahnung. Seien Sie dabei auch so selbständig und beseitigen Sie das Problem gegebenenfalls an allen vorkommenden Stellen (abgemahnt wird häufig nur eine bestimmte Seite).

Wenn die Abmahnung eine empfindlich hohe anwaltliche Kostennote enthält, ist es in den meisten Fällen sinnvoll, deren Zahlung zu verweigern. In diesem Fall muss der Anwalt reagieren. Er kann wegen der geleisteten Unterlassung nur noch die eigene Kostennote einklagen. In diesem Fall wird die Kostennote selbst zum Streitwert. Die meisten Anwälte scheuen diesen Schritt, weil er nur eine Menge zusätzlicher Arbeit bedeutet. Außerdem könnte sich in vielen Fällen herausstellen, dass die Kostennote viel zu hoch angesetzt war.

In anderen Fällen kann es sinnvoll sein, eine modifizierte (und für Sie günstigere) Unterlassungserklärung abzugeben. Tun Sie dies jedoch besser nicht ohne rechtlichen Beistand.

Wenn Sie die Anschuldigung der Abmahnung für völlig ungerechtfertigt halten, sollten Sie unbedingt rechtlichen Beistand hinzuziehen. Der beauftragte Rechtsanwalt sollte mit den Realitäten im Internet vertraut sein und von daher beurteilen können, wie die

Anschuldigung einzuschätzen ist. Hält auch Ihr Anwalt die Anschuldigung für ungerechtfertigt, können Sie eine gerichtliche Gegenoffensive wagen, nämlich in Form einer so genannten *negativen Feststellungsklage*. Damit begehren Sie eine gerichtliche Feststellung, dass die Anschuldigungen haltlos waren.

Eine negative Feststellungsklage als Racheakt anzustrengen, ist jedoch Unfug. Sinn einer negativen Feststellungsklage ist vor allem, durch eine hoffentlich positive Gerichtsentscheidung zu Ihren Gunsten mit dazu beizutragen, eine bestehende Rechtsunsicherheit zu klären.

15.2 Site-Promoting und Aktualität

Manche Websites beginnen ihre Existenz als »Homepage« oder als kleines persönliches Experiment. Es kommt Feedback und es ergeben sich plötzlich Möglichkeiten, finanziellen Nutzen aus der Site zu ziehen. Eine Site, die zunächst einfach nur als elektronische Visitenkarte gedacht war, wird zu einem Shop-System ausgebaut. Eine private Fansite wird plötzlich zur »offiziellen« Fansite. Das Publikum will sozusagen, dass eine Site in die Profiliga aufsteigt. Als Betreiber müssen Sie sich mit einem solchen Erfolg auseinander setzen und richtig reagieren.

Andere Websites werden gründlich geplant. Am Beginn stehen gewisse Ziele und Zielgruppen. Mit dem genauen Konzept und der Realisierung werden meist Angestellte, Agenturen oder Freelancer beauftragt. Es gibt Kostenrechnungen, Budgets, Lasten- und Pflichtenhefte, Grob- und Feinkonzepte, Aufgabenverteilung, verschiedene Realisierungs- und Teststadien, kurz, das gesamte Programm, das auch bei der Entwicklung von Softwareprojekten üblich ist.

Egal, auf welche Weise eine Site mit professionellem Anspruch entsteht: Damit sie den Erwartungen entspricht, sind einige Aspekte zu beachten. Nachfolgend gehen wir auf die wichtigsten davon ein.

15.2.1 Launches und Relaunches

Bei Websites gibt es zwar – anders als etwa in der Buchwelt – kein echtes »Erscheinungsdatum«, zumal einmal veröffentlichte Sites ja weiterleben und sich in der Regel häufig ändern. Dennoch sind der Online-Gang und die Erstveröffentlichung einer Site, in der viel Planung und Arbeit steckt, etwas Besonderes.

Die Erstveröffentlichung einer Website wird als Launch bezeichnet. Wenn eine Site generalüberholt, also z.B. umstrukturiert wurde oder ein neues Design erhalten hat, spricht man von einem Relaunch.

Ankündigungen

Meistens ist der Domain-Name für ein neues Webprojekt ja schon registriert und auf dem Produktivserver verfügbar, bevor die Projektarbeiten abgeschlossen sind. Der bereits eingerichtete Webserver lässt sich dazu nutzen, um auf einer eigens dafür gestalteten Seite

auf die kommende Site hinzuweisen. Vermeiden Sie dabei die mittlerweile als lächerlich empfundene Baustellen-Metapher. Eine ansprechend gestaltete, informative Ankündigungsseite wird dagegen von niemandem als lästig empfunden, nicht einmal dann, wenn er die Seite über einen Suchmaschinentreffer gefunden hat.

Ankündigungen sollten allerdings Termine nennen und die Termine sollten auch tatsächlich eingehalten werden. Alles andere ist unseriös.

Pressearbeit

Presseorgane sollten für Website-Launches auf jeden Fall eine Pressemitteilung erhalten. Konzentrieren Sie sich dabei auf geeignete Fachorgane und auf Webmagazine. Tageszeitungen berichten über Websites wenn überhaupt eher in ihrem Regionalteil, sofern sich der Site-Launch regional »verkaufen« lässt. Das ist beispielsweise der Fall, wenn sich das Projekt als Aushängeschild für die Modernität der Region darstellen lässt.

Es lohnt sich, Pressemitteilungen vorzuformulieren. Die Presseredakteure können dann selbst entscheiden, ob sie eine eigene Mitteilung verfassen, die vorformulierte übernehmen oder eine abgewandelte Form der vorformulierten Mitteilung bevorzugen.

Eventuell bestehende Beziehungen zur Presse sind Gold wert und sollten genutzt werden. Viele Meldungen, die in der Presse das Licht der Welt erblicken, landen nur wegen persönlicher Beziehungen dort. Im Einzelfall kann es auch sinnvoll sein, solche Beziehungen aufzubauen und beispielsweise Redakteure einzuladen, um ihnen vor Ort das Entstehungsumfeld des Webprojekts zu zeigen.

Der Launch als Event

Es spricht nichts dagegen, aus einem Website-Launch einen »Event« zu machen, also Geschäftspartner, Vertreter der Medien, Entwickler usw. zu einem Buffet oder Ähnlichem einzuladen und die frisch gebackene Site über einen Beamer zu präsentieren. Kreativität ist hierbei gefragt.

15.2.2 Websites und Suchmaschinen

Viele Mythen über Maßnahmen, eine bessere Platzierung in Suchmaschinen zu erzielen, halten sich hartnäckig, sind aber längst nicht mehr wahr. Erstens haben alle Anbieter nennenswerter Suchmaschinen aus den Fehlern der Vergangenheit gelernt und fallen nicht mehr so leicht auf Tricks wie seitenfremde Stichworte oder unsichtbaren Text herein. Zweitens hat die Monopolstellung, die Google mittlerweile unter den Suchmaschinen erkämpft hat, dazu geführt, dass wenn überhaupt nur noch für diese Suchmaschine optimiert wird. Und last but not least: Es bringt Ihnen nichts, wenn Sie in einem Zusammenhang gefunden werden, nach dem gar nicht gesucht wird. Anwender erliegen zwar hin und wieder dem Serendipity-Effekt und lassen sich von einem Link anlocken, der sie von ihrem eigentlich anvisierten Ziel weg bringt. Doch mindestens ebenso viele, wenn nicht mehr Anwender reagieren eher verärgert über Suchmaschinentreffer, die gar nichts oder herzlich wenig mit der gestarteten Suche zu tun haben.

Bedeutung von Suchmaschinen

Zumindest Google kennen und nutzen sehr viele Anwender, die im Web surfen. Dagegen sind ehemals führende Suchdienste wie Altavista oder Fireball tendenziell nur noch Anwendern mit langjähriger Weberfahrung ein Begriff. Ernstzunehmende Google-Alternativen wie AllTheWeb (erreichbar unter *http://www.alltheweb.com/*) sind fast nur Insidern bekannt. Einer Studie von Standard & Poor´s Equity Research Services aus dem Jahr 2004 zufolge wird der Marktanteil von Google auf knapp 50% geschätzt. Dahinter folgt mit 20% der Klassiker Yahoo, der allerdings in erster Linie ein Verzeichnisdienst ist, und dahinter mit 14% bzw. 7% die Suchdienste der kundenspezifischen Online-Riesen MSN und AOL.

Von daher wundert es nicht, dass Google weitgehend diktiert, in welcher Form Suchmaschinenoptimierung betrieben wird. Auf den eigenen Seiten präsentiert sich Google auch sehr auskunftsfreudig. Unter *http://www.google.de/intl/de/webmasters/* können und sollten Site-Betreiber nachlesen, wie Google arbeitet, welche Seiten aufgenommen werden und welche Maßnahmen empfohlen werden, um den Google-Bot glücklich zu machen.

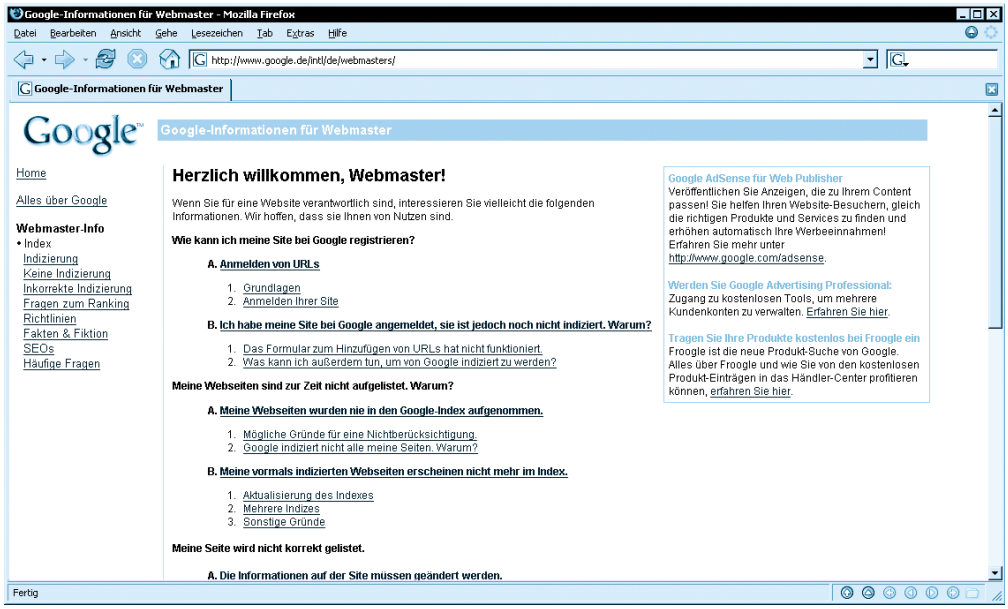


Abbildung 15.4: Google-Dokumentation für Website-Betreiber

Aus Sicht einer Website ist es jedoch auch umgekehrt von Bedeutung, wie hoch überhaupt der Anteil von Anwendern ist, der über eine Suchmaschine zu dem Webangebot findet. Hierbei gilt: Je bekannter eine Site erst mal ist, desto stärker wächst der Anteil an Besuchern, welche die Site über Bookmarks oder Direkteingabe der Adresse in der Adresszeile des Browsers auffinden. Die öffentlich zugängliche Statistik des bekannten Webprojekts SELFHTML, erreichbar unter *http://stats.selfhtml.org/*, zeigt, dass nur ca. 3% der Referrer-Einträge, also der Einträge darüber, woher Anwender zum Webangebot von SELFHTML fanden, auf Google verweisen. Sonstige Suchdienste sind sogar komplett vernachlässigbar.

Viele Sites werden vor allem über bestimmte andere Sites aufgerufen. So hat das österreichische Bildungsministerium beispielsweise eine groß angelegte Initiative für unterrichtsbegleitende Online-Inhalte gestartet. Die entsprechenden Webangebote der Schulbuchverlage werden dabei vorrangig über die zentrale Site des »Bildungsservers« erreicht. Suchmaschinen spielen für den Andrang auf den betreffenden Sites eine völlig untergeordnete Rolle.

Suchmaschinen können für Websites vor allem in nachfolgenden Fällen eine Hilfe sein:

- wenn die Site noch weitgehend unbekannt ist,
- im Falle von Deeplinks zu inhaltstragenden Seiten innerhalb von Websites, die der Anwender zwar kennt, aber an die er bei seiner Suche in dem Augenblick gar nicht gedacht hat.

Suchmaschinen-Ranking

Die meisten Anwender suchen in einer Suchmaschine nur nach einem, maximal zwei Wörtern. Sie sind auch in aller Regel zu bequem oder zu unwissend, um mithilfe von Operatoren, die jede Suchmaschine anbietet, eine Suche zu präzisieren. Die Folge ist, dass die Suche meist zigtausend von Treffern liefert. Nutzungsanalysen ergaben dabei eindeutig, dass nur ganz weit oben platzierte Trefferlinks gute Chancen haben, genutzt zu werden. Schon Links auf Trefferseite zwei werden kaum noch beachtet.

Aus Anbietersicht ist es daher wichtig, dass gefundene Seiten der eigene Site möglichst weit oben in der Trefferliste erscheinen. Die Platzierung einer Site, die zu den Treffern einer Suche gehört, wird als ***Ranking*** bezeichnet. Moderne Suchmaschinen benutzen einen Mix aus diversen Kriterien, anhand derer sie das Ranking festlegen. Dieser Mix wird als ***Ranking-Algorithmus*** bezeichnet. Folgende typische Kriterien gehören zum Ranking-Algorithmus:

- Die Textrelevanz, gemessen am Suchausdruck (wie oft das Wort auf einer Seite vorkommt, ob es auch in Überschriften, in Meta-Tags oder gar im Titel vorkommt, ob es im offensichtlich gemeinten Kontext vorkommt usw.)
- Die Aktualität der Seite, d.h. der Zeitstempel der letzten Änderung
- Der URI (wenn Domain-Name und/oder Verzeichnis- oder Dateinamen etwas mit dem Suchausdruck zu tun haben, wird dies höher bewertet)
- Die Linkpopularität (Links anderer Sites, die auf diese Site oder Seite verweisen)
- Die Popularität der Site aus Sicht der Suchmaschine (wie oft Suchende Treffer-Links zu dieser Site folgen)
- Die Fairness der Site (Sites, in deren Quelltext der Suchmaschinen-Robot Versuche erkennt, die Suchmaschine auszutricksen, werden gar nicht mehr indexiert oder zumindest niedriger eingestuft)

Es gibt noch einige weitere Kriterien. Welche genau und in welchem Gewichtungsverhältnis sie alle zu Buche schlagen, ist das gut gehütete Betriebsgeheimnis jeder Suchmaschine.

Suchmaschinenoptimierung im HTML-Code

Durch folgende Maßnahmen lassen sich Webseiten für Suchmaschinen optimieren:

- Pro Seite möglichst nur ein Thema. Die Site-Struktur sollte also dem Hypertext-Prinzip folgen: Aufteilung in »thematisch atomare« Seiten, die durch nachvollziehbare Verlinkung und Navigation das Lost-in-Hyperspace-Gefühl beim Anwender vermeiden.
- Das zentrale Thema der jeweiligen Seite sollte im Titel, also im title-Element von HTML, sowie in einer h1-Überschrift auftauchen. Titel und Seitenüberschrift sollten möglichst alle wirklich themenrelevanten Wörter enthalten.
- Verwenden Sie für Überschriften in jedem Fall die dafür vorgesehenen HTML-Elemente und vermeiden Sie, einfach nur neutrale div-Boxen so zu formatieren, dass diese aussehen wie Überschriften.
- Die wirklich themenrelevanten Wörter sollten außerdem in den Meta-Tags zu description und keywords auftauchen sowie mehrfach im normalen übrigen Fließtext.
- Wichtige Wörter, Namen, Slogans und dergleichen sollten nicht in Grafiken verpackt werden, sondern als Text gestaltet werden. CSS bietet genügend Möglichkeiten dazu an.
- Für Grafiken und Multimediaelemente auf einer Seite, die dort inhaltstragende Bedeutung haben (z. B. Abbildung einer Landkarte oder Flashmovie mit einer Lernübung), sollten aussagekräftige Alternativtexte verfasst werden. Für Suchmaschinen-Robots sind solche Texte die einzige Möglichkeit, etwas über den Inhalt des Nicht-Text-Elements zu speichern.
- Der HTML-Code jeder Seite sollte valide im Sinne einer konkreten HTML-DTD sein. Verschachtelungsregeln sind zu beachten und geöffnete Tags müssen geschlossen werden.
- Die Seite sollte keine broken Links enthalten, vor allem nicht innerhalb der site-eigenen Navigation.
- Vermeiden Sie Frames, wenn diese dazu führen, dass inhaltstragende Seiten keine eigene Navigation oder keinen Kopfbereich mehr besitzen.

Inhalte vor Suchmaschinen-Robots bewahren mit robots.txt

Wenn Sie eine Textdatei unter dem Namen ***robots.txt*** im Startverzeichnis für das Webangebot einer Domain speichern, wird diese einer gewachsenen De-facto-Konvention zufolge von fast allen Suchmaschinen-Robots ausgelesen. Eine Suchmaschine, die diese Datei ignoriert, gilt selbst nicht mehr als seriös.

Zweck der ***robots.txt*** ist, Suchmaschinen-Robots daran zu hindern, bestimmte Webseiten oder Teile des Webangebots zu indexieren. Das ist sinnvoll für Seiten, deren Inhalt sich laufend ändert (z. B. Titelseite eines Magazins, Newsticker-Seite usw.) oder deren Inhalt zwar öffentlich zugänglich, aber nicht unbedingt in Suchmaschinen auffindbar sein soll (z. B. Test- oder Nonsense-Seiten).

Daneben kann die *robots.txt* auch noch dazu dienen, bestimmte Such-Robots auszuschließen, z. B. wenn Sie aufgrund Ihrer Webstatistik feststellen, dass sich ein bestimmter Such-Robot sehr aggressiv verhält und so aktiv ist, dass er die Site-Performance ausbremst.

Der typische Inhalt einer *robots.txt* sieht so aus:

```
# robots.txt zu http://www.mein-name.de/

User-agent: UniversalRobot/1.0
User-agent: mein-Robot
Disallow: /quellen/dtd/

User-agent: *
Disallow: /unsinn/
Disallow: /temp/
Disallow: /newsticker.php
```

Kommentarzeilen wie in der ersten Zeile werden durch ein Gatterzeichen # eingeleitet.

Ansonsten besteht eine *robots.txt* aus Zeilenblöcken mit einer oder mehreren Zeilen, die mit `User-agent:` beginnen, sowie aus einer oder mehreren Zeilen, die mit `Disallow:` beginnen. Hinter `User-agent:` notieren Sie jeweils den Namen eines bestimmten Such-Robots oder auch einen Platzhalter `*` für »alle Such-Robots«.

Hinter `Disallow:` geben Sie jeweils ein bestimmtes Verzeichnis oder eine bestimmte Datei an. Wildcards wie `*.*` sind hierbei nicht erlaubt.

Linkpopularität

Die Linkpopularität galt eine Zeitlang als die scheinbar objektivste Methode, die Bedeutung und Publikumsbeliebtheit einer Site zu ermitteln und den daraus gewonnenen Faktor beim Ranking zu berücksichtigen. Doch auch dieser Faktor ist wegen Tricksereiversuchen wieder nur zu einem von vielen geworden. So bildeten sich beispielsweise Webringe, bei denen jede Site auf jede andere des Rings verlinkte.

Der Faktor Linkpopularität sollte also nicht mehr überbewertet werden, aber dennoch nicht unberücksichtigt bleiben. Für die Praxis bedeutet dies, dass es sich lohnt, die eigene Site in kleinere, meist themenbezogene Linkverzeichnisse eintragen zu lassen oder Anbieter von anderen, thematisch geeigneten Sites anzumailen und zu fragen, ob dort ein Link auf die Site eingefügt werden könne. Dazu bedarf es allerdings in vielen Fällen persönlicher Überzeugungskraft.

Selbstverständlich besteht auch die Möglichkeit, Links zu kaufen. Klassische Werbe-Banner werden von Suchmaschinen allerdings erkannt und sicher nicht in die Zählung der Linkpopularität mit einbezogen. Seriöse, in den Inhalt sinnvoll eingebettete Textlinks, die gar nicht als Werbung erkennbar sind, können diesen Zweck jedoch sehr gut erfüllen.

Google hat den Faktor der Linkpopularität erweitert, indem man eine Ranking-Skala von 1 bis 10 einführte. Jeder Google bekannten Webadresse wird ein entsprechender Ranking-Wert zugewiesen. Enthält nun eine Seite mit Page-Rank 6 einen Link zu einer fremden

Adresse, so erhält diese automatisch Page-Rank 5, also einen Rangpunkt weniger als die verlinkende Seite. Auch das hat wieder viele Site-Betreiber aufgescheucht und dazu getrieben, sich Links auf fremden Seiten mit hohem Page-Rank zu kaufen. Auch diese Manie hat mittlerweile wieder abgenommen.

Search Engine Optimizers (SEOs)

Schon seit vielen Jahren gibt es Suchmaschinenanmeldedienste, die eine Website in allen nur erdenklichen Suchmaschinen eintragen. Dieser Service ist allerdings kaum noch sinnvoll, da es nur noch wenige bedeutsame Suchmaschinen im Web gibt. Außerdem blocken Suchmaschinen wie Google mittlerweile automatische Eintragungen ab, indem sie beim Eintragen verlangen, eine als Grafik angezeigte Zeichenkombination einzugeben. Dadurch lassen sich sehr wirksam menschliche von automatischen »Benutzern« unterscheiden.

Seit einiger Zeit gibt es darüber hinaus aber auch Webdienste oder PC-Software, deren Aufgabe darin besteht, eine Site für Suchmaschinen zu optimieren bzw. das Site-Ranking in Suchmaschinen zu erhöhen. Die meist aggressiven Versprechungen solcher Dienste und Software sind in der Praxis allerdings kaum haltbar und bewirken möglicherweise sogar das Gegenteil des erhofften Erfolgs. Denn wenn eine Suchmaschine eine von solchen Produkten versuchte Trickserei erkennt, kann das zum Löschen der Site aus der Datenbank führen oder zumindest zu einer Abwertung im Ranking.

15.2.3 Werbung für Websites

Jede Website ist nur eine von Tausenden, egal mit welchem Aufwand sie entwickelt wurde. Das sicherste Mittel, sich langfristig aus der Masse hervorzuheben, sind Inhalte mit hohem Nutzwert oder gute Unterhaltung. Dennoch kann es sich je nach Projekt auch lohnen, ein Werbebudget einzuplanen.

Dabei gibt es wie in der übrigen Werbung mehrere mögliche Strategien. Eine Strategie besteht darin, innerhalb kurzer Zeit massiv zu werben und dabei z.B. auch vergleichsweise teure Werbeformen wie Fernsehwerbung zu nutzen. Das lockt, wenn es gut geht, erst einmal viele neugierige Besucher an. Die Site muss dann allerdings halten, was sie verspricht. Andernfalls verschwinden die Besucher genauso schnell wieder, wie sie kamen.

Eine andere Strategie ist die der langfristigen Etablierung. Dabei wird meist auf preiswertere Werbeformen wie Werbebanner auf anderen, bekannten Sites gesetzt. Dafür ist der Werbezeitraum länger bis dauerhaft.

Welche Strategie erfolgreicher ist, hängt von der Art des Projekts und den gesetzten Zielen ab. Websites, die vor allem von Benutzerinhalten leben, egal ob Partnerbörse, Auktionshaus oder Fan-Community, müssen versuchen, möglichst schnell eine akzeptable Menge an Benutzeraktionen und -einträgen zusammenzubekommen. Denn nichts ist peinlicher als eine groß angelegte, aber gähnend leere Infrastruktur für Benutzeraktivitäten. Eine Site dagegen, die sich vor allem als Informationsportal versteht und mit einem Startangebot

online geht, das jedoch noch erweitert werden soll, wird eher auf eine Werbeform setzen, die ein langsames, aber stetiges Besucherwachstum verspricht.

Webspezifische Werbeformen

Bleiben wir bei der Werbung innerhalb des Web. Zahlreiche Sites, die selber eine gewisse Bekanntheit genießen, bieten die Möglichkeit an, Produkte zu bewerben. Das können natürlich auch andere, noch nicht so bekannte Sites sein.

Die klassischste Form ist der anklickbare Werbe-*Banner*, eine Pixelgrafik in der standardisierten Größe von 468x60 Pixel, die in einen Link zur beworbenen Site eingebettet wird. Anbieter-Sites, die solche Banner schalten, verpflichten sich, den Banner zentral im Kopfbereich der Seite anzuzeigen. Genau diese Aspekte – die einheitliche Größe und die immer gleiche Position – haben jedoch dazu geführt, dass solche Banner kaum noch so viel Aufmerksamkeit auf sich ziehen, dass darauf geklickt wird. Statistiken zufolge klicken unter 0,5% der Besucher einer Seite auf einen darin platzierten Werbebanner an der typischen Stelle. Allenfalls die bekannten Effekte unterbewusster Werbewirkung bleiben übrig. Viele Browser oder zusätzlich installierbare Tools bieten sogar die Möglichkeit an, solche leicht erkennbaren Banner bei der Seitenanzeige auszublenden. Es lohnt sich also, über Alternativen nachzudenken. Alternativen sind:

- *Sponsored Textlinks:* Auf einer bekannten Site wird ein Link auf Ihre Site als ganz normaler Textlink gesetzt, idealerweise in einem lesenswerten Textzusammenhang, einer »redaktionellen Empfehlung« oder einer Rubrik wie »Site der Woche«. Diese Werbeform ist besonders unaufdringlich und hat eine hohe Anwenderakzeptanz.
- *Popup- und Popunder-Fenster:* Beim Aufruf einer bekannten Site öffnet sich automatisch ein kleines Zweitfenster, frei für Werbung, z. B. für Ihre Site. Das Fenster erscheint entweder gleich im Vordergrund (Popup) oder hinter dem Haupt-Browser-Fenster, zunächst unsichtbar, bleibt stehen und überrascht den Anwender, wenn dieser das Haupt-Browser-Fenster schließt (Popunder). Popunders haben für die Werbeanbieter den Vorteil, dass der Anwender, wenn er das Werbefenster entdeckt, oft gar nicht mehr weiß, welche Seite das Fenster geöffnet hat. Beide Formen haben jedoch eine sehr niedrige Anwenderakzeptanz. Viele Browser unterdrücken bereits optional oder per Voreinstellung automatisch geöffnete Zweitfenster. Und last but not least widersprechen Popups und Popunders dem Prinzip des tabbed Browsing, das von immer mehr Browsern verwendet wird.
- *Popup-Divs:* Eine Reaktion verzweifelter Werbeanbieter auf die Feindseligkeit des Publikums und der Browser-Hersteller gegenüber Popups und Popunders. Anstelle von Fenstern erscheinen einfach mit CSS gestaltete, absolut positionierte div-Bereiche über dem übrigen Seiteninhalt und enthalten Werbung. Die Popup-Divs verschwinden automatisch nach ein paar Sekunden wieder oder sie simulieren ein Fenster mit Schließen-Schaltfläche. Diese Form wird von Anwendern jedoch als besonders aggressiv und ärgerlich empfunden.

- *Content- bzw. News-Breaks:* Ebenfalls eine Form der Bannerwerbung. Jedoch ist die Werbegrafik in diesem Fall nur 148x40 Pixel groß und erscheint nicht oben zentriert, sondern links oder rechts in Spalten an verschiedenen Stellen.
- *Dynamische Banner:* Eine andere Alternative setzt zwar auf den Standardbanner an der gewohnten Stelle, doch dafür nicht in Form einer langweiligen statischen oder (schlimmer) zappelnden animierten GIF-Grafik, sondern in Form eines Flashmovie oder Java-Applet. Flashmovies setzen sich dabei immer mehr durch, da sie schneller geladen sind im Browser.
- *Grafische Anzeigen in Newslettern:* Viele größere Sites betreiben einen Newsletter-Service, den auch viele tausend Besucher in Anspruch nehmen. Darin werden sie von Zeit zu Zeit oder in regelmäßigen Abständen über Neuigkeiten des Anbieters oder seiner Site informiert. Die Newsletter werden meistens als HTML-Mails verschickt und können daher auch Grafiken enthalten. Diese Möglichkeit wird vielfach genutzt, um Werbung von Drittanbietern zu schalten. Die Werbung erscheint somit nicht auf der Website, sondern in den Newsletter-Mails.

Es gibt noch diverse weitere Ansätze. Der BVDW (Bundesverband Digitale Wirtschaft), erreichbar unter *http://www.bvdw.org/*, ist ein Interessensverband, der sich unter anderem auch für die Etablierung von elektronischen Werbeformen engagiert. Für seine Mitglieder bietet er Beratungsfunktion an.

15.2.4 Newsticker, Newsletter und RSS-Newsfeed

Von jeder professionellen Website wird erwartet, dass sie aktuell ist. Dazu gehört zwar vor allem die Pflege von Inhalten und Links, doch diese Arbeit ist an der Oberfläche kaum sichtbar. Um nicht nur aktuell zu sein, sondern dies auch zu demonstrieren, bieten fast alle größeren Sites irgendeine Art von prominent platziertem News-System an.

Newsticker

Manche Sites beschränken sich auf einen Newsticker. Dieser kann in folgenden typischen technischen Varianten angeboten werden:

- in einem Zweitfenster (Popup-Fenster),
- in einem eingebetteten Frame-Fenster (iframe),
- als automatisch scrollender und/oder navigierbarer Bereich (mit JavaScript, Java oder Flash realisiert).

Nachfolgendes Listing zeigt, wie sich mit nur wenig JavaScript-Code ein automatisch scrollender Newsticker integrieren lässt:

Listing 15.1: HTML-Dokument mit javascript-basiertem, automatisch scrollendem Ticker

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

```
<head>
<title>Blindtexticker</title>
<meta http-equiv="content-type" content="text/html;
   charset=ISO-8859-1">
<style type="text/css">
html,body { margin-top:0px; margin-left:0px; height:100% }
h1 { font-family:Arial,sans-serif; font-size:23px;
   margin-top:0px; margin-bottom:23px; }
p { font-family:Arial,sans-serif; font-size:13px;
   margin-top:9px; }
#ticker { position:absolute; top:10px; left:20px;
   width:140px; color:#000000; z-index:1; }
#above { position:absolute; top:0px; left:20px; width:140px;
   border:none; height:10px; z-index:2;
   background-color:#FFFFFF; }
#beyond { position:absolute; top:150px; left:20px;
   width:140px; border:none; height:100%; z-index:2;
   background-color:#FFFFFF; border-top:solid 2px blue; }
#page { position:absolute; top:10px; left:200px; right:20px;
   border-left:2px solid red; padding-left:30px;
   height:100%; }
.tickertext { font-family:Arial,sans-serif;
   font-size:11px; margin-top:9px; }
</style>
<script type="text/javascript">
<!--
var pixel = 1;
var scrolled_pixel = 0;
var max_height = 0;
var displayed_height = 140;
function init() {
   if(document.getElementById) {
      max_height =
         document.getElementById("ticker").offsetHeight +
         displayed_height;
      show_ticker();
   }
}
function show_ticker() {
  if(document.getElementById)
   document.getElementById("ticker").style.top =
   (displayed_height - scrolled_pixel) + "px";
  scrolled_pixel += pixel;
  if(scrolled_pixel > max_height)
   scrolled_pixel = 0;
  window.setTimeout("show_ticker()",50);
}
//-->
</script>
</head>
<body onLoad="init()">
<div id="above"></div>
<div id="ticker">
```

```
<p class="tickertext"><b>Schon gewußt?</b> Blindtexte können
philosophische Abgründe bergen.</p>
<p class="tickertext"><b>Blindtext selbst gemacht:</b> Lesen Sie
auf der Seite <a href="selbermachen.htm">Selber blind texten</a>,
was Sie beim Verfassen eigener Blindtexte beachten müssen.</p>
<p class="tickertext"><b>Blindtext-Award!</b> Wir freuen uns,
Ihnen unseren <a href="award.htm">Blindtext-Award</a> vorstellen
zu dürfen, den wir monatlich für die besten Blindtexte im Web
vergeben.</p>
<p class="tickertext"><b>Autoren zum Thema:</b> Was renommierte
Schriftsteller zum Thema Blindtext sagen. Lesen Sie Interviews
mit vier namhaften Autoren der Gegenwartsliteratur - exklusiv
bei uns!</p>
</div>
<div id="beyond">
<p>Diese Seiten wurden ganz sicher
blind drauf los geschrieben!</p>
</div>
<div id="page">
<h1>Blindtextticker links</h1>
<p>Und das, was auf dem Rest der Seite steht.</p>
</div>
</body>
</html>
```



Abbildung 15.5: Javascript-basierter, automatisch scrollender Ticker

Der Ticker-Inhalt ist bei diesem Beispiel in den HTML-Code der Seite integriert und erscheint dort im div-Bereich mit dem id-Namen `ticker`. In der Abbildung ist dies der Bereich oben links mit der kleineren Schrift. Der Inhalt scrollt automatisch in akzeptabler Lesegeschwindigkeit. Ist der Inhalt komplett durchgescrollt, beginnt der nächste Durchlauf.

In der Praxis könnte der Inhalt dieses Bereichs beispielsweise mittels PHP oder SSI in den HTML-Code integriert werden, wobei der Inhalt aus einer separat gepflegten News-Datei

oder einer News-Datenbanktabelle geholt wird. Ein interne Webseite könnte das formulargestützte Editieren der News-Texte ermöglichen.

Newsticker gelten als anwenderfreundlich und können auf umfangreicheren Sites z.B. die Aufgabe übernehmen, auf neue Inhalte hinzuweisen.

E-Mail-Newsletter

Bei Newslettern, die über E-Mail verschickt werden, ist der technische Aufwand schwieriger absehbar. Newsletter, die nicht mehr als maximal ein paar hundert Abonnenten haben und nur einmal pro Woche oder Monat verschickt werden, lassen sich problemlos mit guten E-Mail-Clients verwalten und versenden.

Etwas komfortabler ist der Einsatz einer Mailingliste. Das klassische Produkt dieser Sparte, Majordomo, hat mittlerweile durch ein ebenfalls OpenSource-basiertes Produkt namens Mailman (*http://www.gnu.org/software/mailman/*) starke Konkurrenz erhalten. Mailman bietet unter anderem auch eine webbasierte Verwaltungsoberfläche an. Es kann auf allen Unix/Linux-basierten Systemen installiert werden, erfordert allerdings eine vorangehende Installation des Python Scriptinterpreters (*http://www.python.org/*).

Wer alle nur erdenklichen Features wie personalisierte Mails oder Feedback-Verwaltung braucht und ein extrem hohes Versandaufkommen hat, benötigt dagegen eine in aller Regel kommerzielle Softwarekomplettlösung, wie sie z.B. das Postcast-Paket anbietet (*http://www.postcast.com/*). Das Paket verfügt über ein komplett vom eigenen PC aus steuerbares Mail-Broadcasting mit integriertem SMTP-Server.

RSS-Newsfeed

Stark in Mode gekommen ist eine XML-basierte News-Lösung namens RSS. Die Abkürzung steht seit Version 2.0 des RSS-Formats für ***Really Simple Syndication***. Das Konzept besteht darin, die News-Daten in einer XML-Datei mit bestimmtem, standardisiertem Aufbau zu speichern. Die Datenhaltung ist damit zunächst komplett von jeder Nutzung abgekoppelt, was den Vorteil hat, dass die Nutzungsmöglichkeiten flexibel sind. In ***Abschnitt 14.4*** haben wir das RSS-Datenformat behandelt und in ***Abschnitt 14.8.5*** haben wir an einem Praxisbeispiel gezeigt, wie Sie mit PHP ein News-Script mit RSS-Fees-Funktion erstellen können.

Konzeptionell müssen Sie entscheiden, ob Sie im RSS-Format nur News-Schlagzeilen, Anlesertexte oder die vollständigen News-Texte anbieten möchten. Ein Feed, der nur Schlagzeilen anbietet, ist allenfalls für Nutzer interessant, die RSS beispielsweise in Form dynamischer Bookmarks nutzen, wie es vom Browser Firefox angeboten wird. Wer einen richtigen RSS-Reader benutzt, wird einen solche Feed dagegen für dumm und überflüssig halten. Die meisten Anbieter wählen den Weg, im Feed Schlagzeilen und Anlesertexte anzubieten. Der Anwender erhält damit eine optisch gute Übersicht über aktuelle News und kann Details auf der Website selbst nachlesen. Verfechter der reinen Feed-Lehre kritisieren jedoch auch diese Praxis, da der Feed dabei nur als »Appetizer« genutzt wird, nicht jedoch als wirklich site-unabhängiges Medienorgan. Andererseites haben viele Feed-Anbieter Bedenken, dass bei einem Feed, der gar nicht mehr auf die Website für weitere

Inhalte verweist, die Bindung regelmäßiger Seitenbesucher an die Seite verloren geht. Am Ende wird wohl das Benutzerverhalten darüber entscheiden.

15.2.5 Benutzerbasierte Inhalte

Nicht wenige der erfolgreichsten und umfangreichsten Webprojekte basieren vorwiegend auf Inhalten, die von Benutzern beigesteuert werden. Beispiele sind das Auktionshaus eBay oder die freie Enzyklopädie Wikipedia.

Wer jedoch glaubt, dass es genügt, einfach eine Infrastruktur bereitzustellen und zu warten, bis sich alles mit Inhalt füllt, der irrt. Im Web gibt es zahllose Beispiele vor allem von groß angelegten Bulletin-Board-Systemen, in denen gähnende Leere herrscht. Um ein Projekt dieser Art erfolgreich zu etablieren, muss man entweder zum richtigen Zeitpunkt die richtige neue Idee haben (dies trifft wohl auf eBay und Wikipedia zu) oder man muss die Site intensiv bewerben (diesen Weg ist beispielsweise die mittlerweile sehr erfolgreiche Partnerbörse *neu.de* gegangen).

Ein weiterer Irrtum besteht darin, dass Sites mit vorwiegend benutzerbasierten Inhalten weniger redaktionelle Betreuung brauchen. Im Gegenteil: Bei Wikipedia überwacht eine ganze Schar freiwilliger Helfer das Editiergeschehen und greift binnen kürzester Zeit ein, wenn beispielsweise Naziideologien statt sachliche Inhalte beigetragen werden. Bei eBay überwacht ebenfalls ein stattliches Team von Redakteuren laufend alle neu geposteten Auktionen, um schwarze Schafe, offensichtlichen Unsinn und illegale Verkaufsversuche zu entfernen. Große Foren- oder Board-Systeme funktionieren nur durch spürbare, persönliche Moderation. Meist gibt es Moderatoren für bestimmte Bereiche. Sites mit benutzerbasierten Inhalten erfordern also nicht weniger, sondern eher noch mehr Manpower als Sites mit vorwiegend redaktionell beigesteuerten Inhalten.

Erfolgversprechend sind auch Mischformen. Besucher einer erfolgreichen, frequentierten Site mit einem guten Redaktionsangebot nehmen es dankbar an, wenn sie auf der gleichen Plattform auch eine Möglichkeit finden, über die behandelten Themen zu diskutieren oder Fragen zu stellen.

Foren- und Boardsysteme

Meistens werden heutzutage fertige Bulletin-Scripts eingesetzt, um so etwas wie eine Benutzerinteraktion zu realisieren. Doch dabei wird häufig übersehen, dass diese Art Software vorwiegend für Anwendungen der Art gedacht ist, dass Anwender ein neues Thema mit einer Fragestellung beginnen und alle übrigen Beiträge zum Thema sich antwortend mit dem Ursprungsposting auseinander setzen. Geeignet ist dieser Typus, der von Scripts wie phpBB (*http://www.phpbb.de/*) repräsentiert wird, vor allem für Supportforen oder für Umfrageforen.

Für Fachdiskussionen ist dieser Typus wegen seiner flachen Struktur weniger geeignet. Hierfür eignen sich eher thread-orientierte Diskussionsforen, bei denen Diskussionsteilnehmer besser mitverfolgen können, wer auf wen antwortet, wo Nebendiskussionen (so genannte Thread-Drifts) entstehen usw. In der Regel gibt es auch nur ein Forum oder wenige Foren und nicht wie bei Board-Systemen eine weit verzweigte thematische Struktur.

Ein schönes Beispiel für ein thread-basiertes Forum ist das Freeware-Produkt mit dem bescheidenen Titel »Mein kleines Forum« (*http://www.mylittlehomepage.net/de/*). Das Produkt bietet auch das Umschalten auf Board-Ansicht oder gemischte Ansicht zwischen Thread-Struktur und Board-Struktur an.

Das Wunschziel jeder Installation eines Boards oder Forums sollte Anwenderbindung sein. Denn während gewöhnliche redaktionelle Inhalte meist nur innerhalb weniger Minuten konsumiert werden, sorgen interaktive Webinhalte dafür, dass sich Besucher mitunter stundenlang auf einer Website tummeln. Vereinzelt lässt sich sogar suchtartiges Besucherverhalten feststellen, ähnlich wie in Chats. Die persönliche Bindung an die Site wird dadurch natürlich enorm gesteigert.

Allerdings muss jede vielbesuchte Installation auch auf negative Begleiterscheinungen wie Spammer und mehr oder weniger intellektuelle Störenfriede (so genannte Trolle) reagieren. Die Gegenmaßnahmen reichen von verbalen Verwarnungen über Löschungen von Postings bis hin zu Strafanzeigen wegen Hausfriedensbruch, Verleumdung oder Beleidigung. Eine engagierte Moderation ist dazu erforderlich.

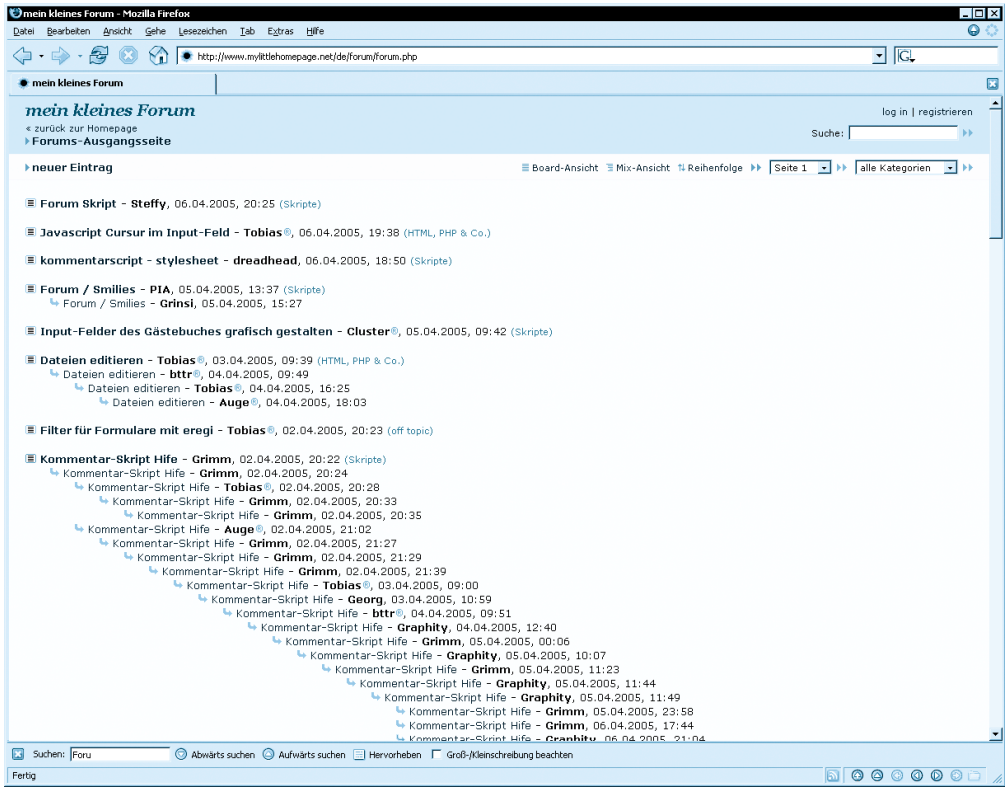


Abbildung 15.6: Auf PHP mit MySQL basierendes, thread-basiertes Forum

Wiki-Systeme

Die Welt war reif für eine kostenlos zugängliche, von beliebigen Benutzern editierbare Enzyklopädie des Menschheitswissens. Das World Wide Web selber verstand sich bereits als Versuch, etwas Derartiges zu realisieren. Doch die Strukturen des Web waren letztlich zu allgemein. Ein anderes Prinzip – das der so genannten Wikis – erwies sich dagegen als idealer Nährboden für das gemeinsame Zusammentragen von Wissen. Das Projekt ***Wikipedia*** (*http://www.wikipedia.org/*, deutsch: *http://de.wikipedia.org/*) ist das bekannteste und erfolgreichste Wiki-System.

Doch das Wiki-Prinzip funktioniert auch in vielen anderen Bereichen, vor allem in Zusammenhang mit Fachwissen, egal ob es sich um Programmiersprachen, Kindererziehung oder Finanzwesen handelt.

Es gibt durchaus sinnvolle Gründe, eine Site, deren Aufgabe ein ständig wachsendes Informationsnetz ist, als Wiki-System zu konzipieren:

- Kein Einzelautor, nicht einmal ein festes Redaktionsteam kann so viele Details zusammentragen wie eine hinreichend große Menge an Besuchern, die selber über fundierte Kenntnisse verfügen.
- Die Redaktion kann sich auf Maßnahmen zur Qualitätssicherung der Inhalte konzentrieren und muss nicht alles selbst verfassen und recherchieren.
- Wikis unterstützen in fast idealer Weise das Hypertextprinzip des Web.

In einem Wiki-System können die Benutzer jederzeit neue Artikel einfügen oder bestehende Artikel ändern. Das System lässt sich so konfigurieren, dass wirklich jeder Benutzer, sogar wenn er sich nicht registriert, freien Zugang zur Bearbeitung der Inhalte hat, oder auch so, dass nur registrierte Benutzer diese Möglichkeit haben, während die übrigen Benutzer die Artikel nur lesen können. Darüber hinaus sind Wiki-Systeme in der Regel so konzipiert, dass es spezielle Zugänge für eine Kernredaktion gibt, die z.B. auch Artikel löschen darf oder im Fall von »Verschlimmbesserungen« eine ältere Artikelfassung wiederherstellen kann. Auch die Versionenkontrolle, also das Speichern von Artikelzuständen, gehört zu einem guten Wiki-System.

Das Editieren von Artikeln erfolgt über normale Webformulare. Die Artikel werden in aller Regel in Datenbanken gespeichert. Um einfache Textstrukturierung und Textformatierung zu ermöglichen, steht meist eine einfache Syntax zur Verfügung:

Der abgebildete Textabsatz lautet im Quelltext (sieht also im Bearbeitungsformular so aus):

```
'''Wikis''', auch '''WikiWikis''' und '''WikiWebs''' genannt, sind im [[World
Wide Web]] verfügbare Seitensammlungen, die von den Benutzern nicht nur gelesen,
sondern auch [[online]] geändert werden.  Sie ähneln damit [[Content Management
System]]en. Der Name stammt von ''wikiwiki'', dem [[Hawaiianische
Sprache|hawaiianischen]] Wort für "schnell". Wie bei [[Hypertext]]en üblich,
sind die einzelnen Seiten und Artikel eines Wikis durch Querverweise
([[Hyperlink|Link]]s) miteinander verbunden. Die Seiten lassen sich jedoch
```

```
sofort am [[Bildschirm]] ändern. Dazu gibt es in der Regel eine
Bearbeitungsfunktion, die ein Eingabefenster öffnet, in dem der Text des
Artikels bearbeitet werden kann.
```

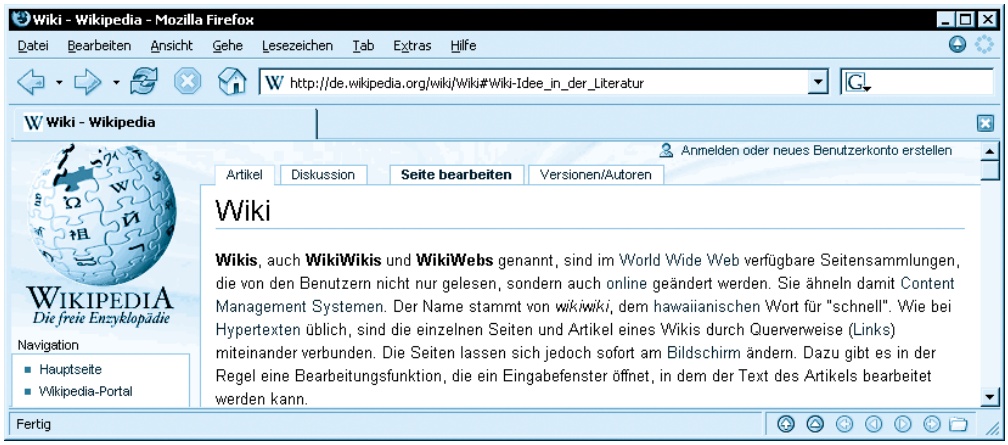


Abbildung 15.7: Anfang des Wikipedia-Artikels über Wikis

Gut erkennbar ist, wie fette Textstellen und interne Links erstellt werden. Interne Wiki-Links – und das ist eine Besonderheit von Wikis – führen stets zu anderen Begriffen, egal, ob es zu den betreffenden Begriffen bereits einen Artikel gibt oder nicht. Wenn ja, dann führt der Link zum entsprechenden Artikel, wenn nein, dann führt er zu einer Seite, auf der darauf hingewiesen wird, dass der Artikel noch nicht existiert. Diese Seite ermuntert gleichzeitig dazu, den Artikel doch gleich selbst zu erstellen.

Wikis stellen eine Provokation für althergebrachte Wissensvermittlung dar: Sie sind praktizierter Wissenskommunismus, leben vom Vertrauen in die Benutzer und muten den Benutzern erfolgreich zu, jenseits von schicken WYSIWYG-Editoren ihre Artikel beizutragen. Damit das Wiki-Prinzip funktioniert, muss allerdings erst einmal eine kritische Masse an Information und aktiven Bearbeitern vorhanden sein. Eine völlig leere Wiki-Infrastruktur wird von Benutzern nicht akzeptiert. Redaktionelle Vorarbeit ist also erforderlich und redaktionell aktive Betreuung ebenfalls.

Links zu spezielleren Wiki-Sites (beispielhaft):

Bonsai-Wiki: *http://www.bonsaiwiki.de/*
Anime- und Manga-Wiki: *http://www.aniki.info/*
Indiander-Wiki: *http://www.indianer-wiki.org/*
Juristen-Wiki: *http://www.jurawiki.de/*
Pferde-Wiki: *http://www.pferdewiki.de/*

Um ein Wiki-System selbst anzubieten, empfiehlt es sich, auf eine der vorhandenen und oft OpenSource-basierten Script- und Softwarelösungen zurückzugreifen, da die Neuentwicklung eines Wiki-Systems mit allen Aspekten extrem aufwändig ist. Wegen des über-

ragenden Erfolgs von Wikipedia setzen die meisten kleineren Wiki-Anbieter mittlerweile auf die gleiche Software wie Wikipedia selbst, nämlich auf *MediaWiki*, eine in PHP programmierte Wiki-Engine mit MySQL-Datenbankanbindung. Dennoch lohnt es sich, verschiedene verfügbare Lösungen zu vergleichen, um für die eigenen Vorstellungen die geeignetste zu finden.

Links zu Wiki-Software:

MediaWiki (PHP/MySQL, OpenSource):
http://sourceforge.net/projects/wikipedia
Confluence (Java, kommerziell): *http://www.atlassian.com/software/confluence/*
coWiki (PHP/MySQL, OpenSource):
http://www.indianer-wiki.org/
DokuWiki (PHP, OpenSource):
http://wiki.splitbrain.org/wiki:dokuwiki
PmWiki (PHP, OpenSource):
http://www.pmwiki.org/

Alle Wiki-Engines arbeiten mit Templates, die sich an eigene Design- und Layoutvorstellungen anpassen lassen.

15.2.6 Favicons – die Site-Abrundung

Eines der unauffälligen, aber den Stil mitbestimmenden Elemente, die mittlerweile bei keiner größeren Site mehr wegzudenken sind, sind die kleinen Icon-Grafiken, die der Site in einigen Browsern wie Mozilla/Firefox in der Adresszeile oder die einem Bookmark-/Favoriteneintrag vorangestellt werden.



Abbildung 15.8: Icon in Adresszeile, Tab und als Logo wiederholt

Ein Favicon ist eine Pixelgrafik mit den Ausmaßen 16x16 Pixeln und einer Palette von 256 RGB-Farben. Auf diese Minifläche sollte ein Symbol gezaubert werden, das gut erkennbar, individuell und passend zum übrigen Design ist. Die Grafik muss im Windows-Icon-Format abgespeichert werden, und zwar unter dem Dateinamen *favicon.ico*. Zum Abspeichern in diesem Format benötigen Sie ein Grafikprogramm, welches im ico-Format

speichern kann (unter MS Windows ist beispielsweise das hervorragende Freeware-Programm IrfanView geeignet, beziehbar unter der Adresse *http://www.irfanview.com/*).

Legen Sie die ***favicon.ico*** auf dem Server-Rechner im Startverzeichnis der Webseiten der Domain ab. Für den MS Internet Explorer genügt das bereits. Er fordert vom Webserver automatisch eine Datei ***favicon.ico*** im Startverzeichnis an. Um die Funktionalität jedoch auch für andere Browser sicherzustellen, sollten Sie die Icon-Datei im HTML-Quelltext mithilfe eines dafür gedachten link-Elements einbinden. Notieren Sie dazu zwischen `<head>` und `</head>` folgende Angabe:

```
<link rel="shortcut icon" href="/favicon.ico">
```

Im Webserver muss dazu allerdings auch noch ein Mime-Type-Eintrag vorgenommen werden, damit die ***.ico***-Datei korrekt an den Browser ausgeliefert wird. In der Konfiguration des Apache-Webservers können Sie dazu folgende Direktive eintragen:

```
AddType image/x-icon .ico
```

Der Eintrag kann wahlweise in der zentralen ***httpd.conf*** oder in einer ***.htaccess***-Datei im Verzeichnis der Datei ***favicon.ico*** notiert werden.

Teil 8 – Referenz

16 Referenz zu HTML

Die HTML-Referenz behandelt alle im W3C-Standard definierten HTML-Elemente, Attribute sowie alle in HTML 4.01 definierten benannten Entities. Es wird jedoch nur die empfohlene strict-Variante von HTML berücksichtigt.

16.1 HTML-Elemente

Die Referenz behandelt alle HTML-Elemente, die zum HTML-Standard 4.01 strict gehören.

16.1.1 Hinweise zu den Tabellen

Die Referenzinformationen zu den HTML-Elementen sind in Tabellen dieser Form enthalten:

Bedeutung:	Kurzinformation zum Einsatzzweck des Elements
Notation:	Informationen, ob End-Tag erforderlich usw.
Eltern-elemente:	Aufzählung aller Elemente, innerhalb deren das Element als direktes Kindelement vorkommen kann
Kindelemente:	Aufzählung aller Elemente, die das Element selbst als direkte Kindelemente enthalten kann
Attribute:	Aufzählung aller in diesem Element erlaubten Attribute

Tabelle 16.1: Schema der Referenztabellen für HTML-Elemente

16.1.2 HTML-Elemente in alphabetischer Reihenfolge

a

Bedeutung:	Verweis und/oder einen Anker für Verweise
Notation:	<a [Attribute]>
Eltern-elemente:	abbr, acronym, address, b, bdo, big, blockquote, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, img, ins, kbd, label, li, object, p, pre, q, samp, select, small, span, strong, sub, sup, td, th, tt, ul, var

Tabelle 16.2: Referenz zum a-Element

Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, dfn, del, em, i, img, input, ins, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	accesskey, charset, class, coords, dir, href, hreflang, id, lang, name, onblur, onclick, ondblclick, onfocus, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, rel, rev, shape, style, tabindex, title, type

Tabelle 16.2: Referenz zum a-Element (Fortsetzung)

abbr

Bedeutung:	Abkürzung wie »HTML«
Notation:	<abbr [Attribute]></abbr>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, li, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, del, dfn, em, i, img, input, ins, kbd, label, map, object, q, samp, select, script, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.3: Referenz zum abbr-Element

acronym

Bedeutung:	Aussprechbare Abkürzung wie »UNESCO«
Notation:	<acronym [Attribute]></acronym>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, li, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, del, dfn, em, i, img, input, ins, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.4: Referenz zum acronym-Element

address

Bedeutung:	Block-Element für Adress- Anschriften- oder Namens-/Mailangaben
Notation:	<address [Attribute]></address>
Eltern-elemente:	blockquote, body, button, dd, del, div, dt, fieldset, form, ins, li, noscript, object, td, th

Tabelle 16.5: Referenz zum address-Element

Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, dfn, em, i, img, input, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.5: Referenz zum address-Element (Fortsetzung)

area

Bedeutung:	Bereich in einer verweis-sensitiven Grafik (Image-Map)
Notation:	<area [Attribute]> (XHML: <area [Attribute] />)
Eltern-elemente:	Map
Kindelemente:	–
Attribute:	Accesskey, charset, class, coords, dir, href, hreflang, id, lang, name, nohref, onblur, onclick, ondblclick, onfocus, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, rel, rev, shape, style, tabindex, title, type

Tabelle 16.6: Referenz zum area-Element

b

Bedeutung:	Fettschrift (Inline)
Notation:	<b [Attribute]>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, li, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, del, dfn, em, i, img, input, ins, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.7: Referenz zum b-Element

base

Bedeutung:	Adressbasis des Dokuments
Notation:	<base [Attribute]> (XHML: <base [Attribute] />)
Eltern-elemente:	head

Tabelle 16.8: Referenz zum base-Element

Kindelemente:	–
Attribute:	href

Tabelle 16.8: Referenz zum base-Element (Fortsetzung)

bdo

Bedeutung:	Umkehrung der Textrichtung
Notation:	<bdo [Attribute]></bdo>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, li, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, delm dfn, em, i, img, input, ins, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, style, title

Tabelle 16.9: Referenz zum bdo-Element

big

Bedeutung:	Größere Schrift als normal (Inline)
Notation:	<big [Attribute]></big>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, li, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, del, dfn, em, i, img, input, ins, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.10: Referenz zum big-Element

blockquote

Bedeutung:	Blockzitat, darf nur Blockelemente als Inhalt enthalten
Notation:	<blockquote [Attribute]></blockquote>
Eltern-elemente:	blockquote, body, button, dd, del, div, dt, fieldset, form, ins, li, map, noscript, object, script, td, th
Kindelemente:	address, blockquote, dl, div, fieldset, form, h1-6, hr, noscript, ol, p, pre, table, ul
Attribute:	cite, class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.11: Referenz zum blockquote-Element

body

Bedeutung:	Dokumentkörper, sichtbarer Teil
Notation:	<body [Attribute]></body>
Eltern-elemente:	html
Kindelemente:	address, blockquote, del, dl, div, form, h1-6, hr, ins, noscript, ol, p, pre, script, table, ul
Attribute:	class, dir, id, lang, onclick, ondblclick, onload, onmousedown, onmouseup, onmouse-over, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, onunload, style, title

Tabelle 16.12: Referenz zum body-Element

br

Bedeutung:	Erzwungener Zeilenumbruch
Notation:	<br [Attribute]> (XHML: <br [Attribute] />)
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, button, cite, code, dd, del, dfn, div, dt, em, field-set, form, h1-h6, i, ins, kbd, label, li, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, var
Kindelemente:	–
Attribute:	id, class, style, title

Tabelle 16.13: Referenz zum br-Element

button

Bedeutung:	Schaltfläche in einem Formular
Notation:	<button [Attribute]></button>
Eltern-elemente:	abbr, acronym, address, b, bdo, big, blockquote, caption, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, legend, li, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, var
Kindelemente:	abbr, acronym, address, b, bdo, big, blockquote, br, cite, code, dfn, dl, div, em, h1-6, hr, i, img, kbd, map, noscript, object, ol, p, pre, q, samp, script, small, span, strong, sub, sup, table, tt, ul, var
Attribute:	accesskey, class, datafld, datasrc, dataformatas, disabled, dir, id, lang, name, onblur, onclick, ondblclick, onfocus, onmousedown, onmouseup, onmouseover, onmouse-move, onmouseout, onkeypress, onkeydown, onkeyup, style, tabindex, title, type, value

Tabelle 16.14: Referenz zum button-Element

caption

Bedeutung:	Tabellenüberschrift
Notation:	<caption [Attribute]></caption>
Eltern-elemente:	table
Kindelemente:	a, abbr, acronym, b, bdo, big, br, cite, code, del, dfn, em, i, img, ins, kbd, q, samp, script, small, span, strong, sub, sup, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.15: Referenz zum caption-Element

cite

Bedeutung:	Zitat bzw. Referenz für ein Zitat (Inline)
Notation:	<cite [Attribute]></cite>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, li, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, del, dfn, em, i, img, input, ins, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.16: Referenz zum cite-Element

code

Bedeutung:	Quelltext/Listing (Inline)
Notation:	<code [Attribute]></code>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, li, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, del, dfn, em, i, img, input, ins, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.17: Referenz zum code-Element

col

Bedeutung:	Vordefinition einer Tabellenspalte
Notation:	<col [Attribute]> (XHML: <col [Attribute] />)
Eltern-elemente:	colgroup, table
Kindelemente:	–
Attribute:	align, char, charof, class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, span, style, title, valign, width

Tabelle 16.18: Referenz zum col-Element

colgroup

Bedeutung:	Vordefinition einer Gruppe von Tabellenspalten
Notation:	<colgroup [Attribute]></colgroup>
Eltern-elemente:	table
Kindelemente:	col
Attribute:	align, char, charof, class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, span, style, title, valign, width

Tabelle 16.19: Referenz zum colgroup-Element

dd

Bedeutung:	Definition eines Terms innerhalb einer Definitionsliste
Notation:	<colgroup [Attribute]></colgroup>
Eltern-elemente:	dl
Kindelemente:	a, abbr, acronym, address, b, bdo, big, blockquote, br, cite, code, del, dfn, div, dl, em, form, h1-h6, hr, i, img, ins, kbd, noscript, object, ol, p, pre, q, samp, script, small, span, strong, sub, sup, table, tt, ul, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.20: Referenz zum dd-Element

del

Bedeutung:	Änderungsmarkierung für »gelöschter/zu löschender Text«
Notation:	<del [Attribute]></del>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, body, button, caption, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, legend, li, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, var
Kindelemente:	a, abbr, acronym, address, b, bdo, big, blockquote, br, button, cite, code, del, dfn, div, dl, em, fieldset, form, h1-h6, i, img, input, ins, kbd, label, noscript, object, ol, p, pre, q, samp, select, small, span, strong, sub, sup, textarea, tt, ul, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.21: Referenz zum del-Element

dfn

Bedeutung:	Begriff, der an dieser Stelle eingeführt/definiert wird
Notation:	<dfn [Attribute]></dfn>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, li, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, del, dfn, em, i, img, input, ins, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.22: Referenz zum dfn-Element

div

Bedeutung:	Allgemeines Blockelement / Bereich
Notation:	<div [Attribute]>
Eltern-elemente:	blockquote, body, button, dd, del, div, fieldset, form, ins, li, noscript, object, td, th
Kindelemente:	a, abbr, acronym, address, b, bdo, big, blockquote, br, cite, code, del, dfn, div, dl, em, form, h1-h6, hr, i, img, ins, kbd, noscript, object, ol, p, pre, q, samp, script, small, span, strong, sub, sup, table, tt, ul, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.23: Referenz zum div-Element

dl

Bedeutung:	Definitionsliste, bestehend aus zu definierenden Ausdrücken und Definitionen
Notation:	<dl [Attribute]></dl>
Eltern-elemente:	blockquote, body, button, dd, del, div, fieldset, form, ins, li, noscript, object, td, th
Kindelemente:	dd, dt
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.24: Referenz zum dl-Element

dt

Bedeutung:	Zu definierender Ausdruck in einer Definitionsliste
Notation:	<dt [Attribute]></dt>
Eltern-elemente:	dl
Kindelemente:	a, abbr, acronym, b, bdo, big, br, cite, code, del, dfn, em, i, img, input, ins, kbd, label, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.25: Referenz zum dt-Element

em

Bedeutung:	Hervorhebung/emphatisch (Inline)
Notation:	<em [Attribute]>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, li, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, del, dfn, em, i, img, input, ins, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.26: Referenz zum em-Element

fieldset

Bedeutung:	Elementgruppe innerhalb eines Formulars
Notation:	<fieldset [Attribute]></fieldset>
Eltern-elemente:	blockquote, button, dd, del, div, fieldset, form, ins, li, noscript, object, td, th
Kindelemente:	a, abbr, acronym, address, b, bdo, big, blockquote, br, cite, code, del, dfn, div, dl, em, form, h1-h6, hr, i, img, ins, kbd, legend, noscript, object, ol, p, pre, q, samp, script, small, span, strong, sub, sup, table, tt, ul, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.27: Referenz zum fieldset-Element

form

Bedeutung:	Formulardefinition
Notation:	<form [Attribute]></form>
Eltern-elemente:	blockquote, dd, del, div, fieldset, ins, li, noscript, object, td, th
Kindelemente:	address, blockquote, del, dl, div, fieldset, form, h1-6, hr, ins, noscript, ol, p, pre, script, table, ul
Attribute:	action, accept, accept-charset, class, dir, enctype, id, lang, method, name, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, onreset, onsubmit, style, title

Tabelle 16.28: Referenz zum font-Element

h1, h2, h3, h4, h5, h6

Bedeutung:	Überschrift 1. bis 6. Ordnung
Notation:	<h[1-6] [Attribute]></h[1-6]>
Eltern-elemente:	blockquote, body, button, dd, del, div, dt, fieldset, form, ins, li, noscript, object, td, th
Kindelemente:	a, abbr, acronym, b, bdo, big, br, cite, code, del, dfn, em, i, img, input, ins, kbd, label, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.29: Referenz zu den Elementen h1, h2, h3, h4, h5 und h6

head

Bedeutung:	Kopfdaten eines HTML-Dokuments
Notation:	<head [Attribute]></head>
Eltern-elemente:	html
Kindelemente:	base, link, meta, object, script, style
Attribute:	dir, lang, profile

Tabelle 16.30: Referenz zum head-Element

hr

Bedeutung:	Horizontale Linie
Notation:	<hr [Attribute]> (XHML: <hr [Attribute] />)
Eltern-elemente:	blockquote, body, button, dd, del, div, fieldset, form, ins, li, noscript, object, td, th
Kindelemente:	–
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.31: Referenz zum hr-Element

html

Bedeutung:	Dokument-Element von HTML
Notation:	<html [Attribute]></html>
Eltern-elemente:	–
Kindelemente:	head, body
Attribute:	dir, lang, version

Tabelle 16.32: Referenz zum html-Element

i

Bedeutung:	Kursivschrift (Inline)
Notation:	<i [Attribute]></i>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, li, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var

Tabelle 16.33: Referenz zum i-Element

Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, del, dfn, em, i, img, input, ins, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.33: Referenz zum i-Element (Fortsetzung)

img

Bedeutung:	Grafikreferenz
Notation:	<img [Attribute]> (XHML:)
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, li, object, p, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	–
Attribute:	alt, class, dir, height, id, ismap, lang, longdesc, name, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, src, style, title, usemap, width

Tabelle 16.34: Referenz zum img-Element

input

Bedeutung:	Formularfeld für Eingaben, Radiobuttons, Checkboxen, Schaltflächen usw.
Notation:	<input [Attribute]> (XHML: <input [Attribute] />)
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, cite, code, dd, del, dfn, div, dt, em, fieldset, h1-h6, i, ins, kbd, label, li, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	–
Attribute:	accept, accesskey, alt, class, checked, datafld, datasrc, dataformatas, disabled, dir, id, ismap, lang, maxlength, name, onblur, onchange, onclick, ondblclick, onfocus, onmouse-down, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkey-down, onkeyup, onselect, readonly, size, src, style, tabindex, title, type, usemap, value

Tabelle 16.35: Referenz zum input-Element

ins

Bedeutung:	Änderungsmarkierung für »eingefügt/einzufügender Text«
Notation:	<ins [Attribute]></ins>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, body, button, caption, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, legend, li, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, var

Tabelle 16.36: Referenz zum ins-Element

Kindelemente:	a, abbr, acronym, address, b, bdo, big, blockquote, br, button, cite, code, del, dfn, div, dl, em, fieldset, form, h1-h6, i, img, input, ins, kbd, label, noscript, object, ol, p, pre, q, samp, select, small, span, strong, sub, sup, textarea, tt, ul, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.36: Referenz zum ins-Element (Fortsetzung)

kbd

Bedeutung:	Vom Anwender einzugebender Text (Inline)
Notation:	<kbd [Attribute]></kbd>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, li, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, del, dfn, em, i, img, input, ins, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.37: Referenz zum kbd-Element

label

Bedeutung:	Beschriftung für ein Formularfeld
Notation:	<label [Attribute]></label>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, h1-h6, i, ins, kbd, label, li, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	a, abbr, acronym, b, bdo, big, br, cite, code, del, dfn, em, i, img, input, ins, kbd, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	accesskey, class, dir, for, id, lang, onblur, onclick, ondblclick, onfocus, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.38: Referenz zum label-Element

legend

Bedeutung:	Überschrift einer fieldset-Formularelementegruppe
Notation:	<legend [Attribute]></legend>
Eltern-elemente:	fieldset

Tabelle 16.39: Referenz zum legend-Element

Kindelemente:	a, abbr, acronym, b, bdo, big, br, cite, code, del, dfn, em, i, img, input, ins, kbd, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	accesskey, class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.39: Referenz zum legend-Element (Fortsetzung)

li

Bedeutung:	Listeneintrag in einer Aufzählungs- oder nummerierten Liste
Notation:	<li [Attribute]>
Eltern-elemente:	ol, ul
Kindelemente:	a, abbr, acronym, address, b, bdo, big, blockquote, br, cite, code, del, dfn, div, dl, em, form, h1-h6, hr, i, img, ins, kbd, noscript, object, ol, p, pre, q, samp, script, small, span, strong, sub, sup, table, tt, ul, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.40: Referenz zum li-Element

link

Bedeutung:	Logische Verknüpfungen des Dokuments
Notation:	<link [Attribute]> (XHML: <link [Attribute] />)
Eltern-elemente:	head
Kindelemente:	–
Attribute:	charset, class, dir, href, hreflang, id, lang, media, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, rel, rev, style, title, type

Tabelle 16.41: Referenz zum link-Element

map

Bedeutung:	Definitionsbereich für verweis-sensitive Grafiken (Image-Maps)
Notation:	<map [Attribute]></map>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, h1-h6, i, ins, kbd, label, li, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var

Tabelle 16.42: Referenz zum map-Element

Kindelemente:	address, blockquote, dl, div, fieldset, form, h1-6, hr, noscript, ol, p, pre, table, ul
Attribute:	class, dir, id, lang, name, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.42: Referenz zum map-Element (Fortsetzung)

meta

Bedeutung:	Metadaten zum HTML-Dokument
Notation:	<meta [Attribute]> (XHML: <meta [Attribute] />)
Eltern-elemente:	head
Kindelemente:	–
Attribute:	content, dir, http-equiv, lang, name, scheme

Tabelle 16.43: Referenz zum meta-Element

noscript

Bedeutung:	Alternativbereich für den Fall, dass ein Script-Bereich nicht ausführbar ist
Notation:	<noscript [Attribute]></noscript>
Eltern-elemente:	blockquote, body, button, dd, del, div, fieldset, form, ins, li, object, td, th
Kindelemente:	address, blockquote, dl, div, fieldset, form, h1-6, hr, ol, p, pre, table, ul
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.44: Referenz zum noscript-Element

object

Bedeutung:	Objekt für Fremddaten und Multimedia-Referenzen
Notation:	<object [Attribute]></object>
Eltern-elemente:	blockquote, body, button, dd, del, div, fieldset, form, ins, li, noscript, object, td, th
Kindelemente:	a, abbr, acronym, address, b, bdo, big, blockquote, br, cite, code, del, dfn, div, dl, em, form, h1-h6, hr, i, img, ins, kbd, noscript, object, ol, p, param, pre, q, samp, script, small, span, strong, sub, sup, table, tt, ul, var
Attribute:	archive, class, classid, codebase, codetype, data, datafld, datasrc, dataformatas, declare, height, dir, id, lang, name, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, standby, style, tabindex, title, type, usemap, width

Tabelle 16.45: Referenz zum object-Element

ol

Bedeutung:	Nummerierte Liste
Notation:	<ol [Attribute]>
Eltern-elemente:	blockquote, body, button, dd, del, div, fieldset, form, ins, li, noscript, object, td, th
Kindelemente:	li
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.46: Referenz zum ol-Element

optgroup

Bedeutung:	Gruppe von Auswahloptionen in einer Auswahlliste
Notation:	<optgroup [Attribute]></optgroup>
Eltern-elemente:	select
Kindelemente:	option
Attribute:	class, dir, disabled, id, label, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.47: Referenz zum optgroup-Element

option

Bedeutung:	Eintrag in einer Auswahlliste
Notation:	<option [Attribute]></option>
Eltern-elemente:	select, optgroup
Kindelemente:	–
Attribute:	class, dir, disabled, id, label, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, selected, style, title, value

Tabelle 16.48: Referenz zum option-Element

p

Bedeutung:	Textabsatz
Notation:	<p [Attribute]></p>
Eltern-elemente:	blockquote, body, button, dd, del, div, dt, fieldset, form, ins, li, noscript, object, td, th
Kindelemente:	a, abbr, acronym, b, bdo, big, br, cite, code, del, dfn, em, i, img, input, ins, kbd, label, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.49: Referenz zum p-Element

param

Bedeutung:	Parameter für Objekte
Notation:	<param [Attribute]> (XHML: <param [Attribute] />)
Eltern-elemente:	object
Kindelemente:	–
Attribute:	id, name, value, valuetype, type

Tabelle 16.50: Referenz zum param-Element

pre

Bedeutung:	Präformatierter Text mit Umbrüchen wie eingegeben
Notation:	<pre [Attribute]></pre>
Eltern-elemente:	blockquote, body, button, dd, del, div, dt, fieldset, form, ins, li, noscript, object, td, th
Kindelemente:	a, abbr, acronym, b, bdo, br, cite, code, del, dfn, em, i, input, ins, kbd, label, q, samp, script, select, span, strong, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.51: Referenz zum pre-Element

q

Bedeutung:	Zitat (Inline)
Notation:	<q [Attribute]></q>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, li, noscript, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, del, dfn, em, i, img, input, ins, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	cite, class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.52: Referenz zum q-Element

samp

Bedeutung:	Beispieltext (Inline)
Notation:	<samp [Attribute]></samp>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, li, noscript, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, del, dfn, em, i, img, input, ins, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.53: Referenz zum samp-Element

script

Bedeutung:	Scriptbereich, z. B. für JavaScript
Notation:	<script [Attribute]></script>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, body, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, head, i, ins, kbd, label, li, object, p, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	–
Attribute:	charset, defer, event, for, src, type

Tabelle 16.54: Referenz zum script-Element

select

Bedeutung:	Auswahlliste
Notation:	<select [Attribute]></select>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, cite, code, dd, del, dfn, div, dt, em, fieldset, h1-h6, i, ins, kbd, label, li, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	optgroup, option
Attribute:	class, dir, disabled, id, lang, name, multiple, onblur, onchange, onclick, ondblclick, onfocus, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, size, style, tabindex, title

Tabelle 16.55: Referenz zum select-Element

small

Bedeutung:	Kleinere Schrift als normal (Inline)
Notation:	<small [Attribute]></small>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, li, noscript, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, del, dfn, em, i, img, input, ins, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.56: Referenz zum small-Element

span

Bedeutung:	Allgemeines Inline-Element ohne Eigenbedeutung
Notation:	<span [Attribute]>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, li, noscript, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, del, dfn, em, i, img, input, ins, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.57: Referenz zum span-Element

strong

Bedeutung:	Hervorhebung, betont (Inline)
Notation:	<strong [Attribute]>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, li, noscript, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, del, dfn, em, i, img, input, ins, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.58: Referenz zum strong-Element

style

Bedeutung:	Stylebereich, z. B. für CSS
Notation:	<style [Attribute]></style>
Eltern-elemente:	head
Kindelemente:	–
Attribute:	dir, lang, media, title, type

Tabelle 16.59: Referenz zum style-Element

sub

Bedeutung:	Tiefgestellter Text
Notation:	
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, li, noscript, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, del, dfn, em, i, img, input, ins, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.60: Referenz zum sub-Element

sup

Bedeutung:	Hochgestellter Text
Notation:	
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, li, noscript, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, del, dfn, em, i, img, input, ins, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.61: Referenz zum sup-Element

table

Bedeutung:	Tabellendefinition
Notation:	<table [Attribute]></table>
Eltern-elemente:	blockquote, body, button, dd, del, div, fieldset, form, ins, li, noscript, object, td, th
Kindelemente:	caption, col, colgroup, tbody, tfoot, thead, tr
Attribute:	border, cellpadding, cellspacing, class, dir, id, frame, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, rules, style, summary, title, width

Tabelle 16.62: Referenz zum table-Element

tbody

Bedeutung:	Tabellenkörper
Notation:	<tbody [Attribute]></tbody>
Eltern-elemente:	table
Kindelemente:	tr
Attribute:	align, char, charoff, class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title, valign

Tabelle 16.63: Referenz zum tbody-Element

td

Bedeutung:	Datenzelle in einer Tabelle
Notation:	<td [Attribute]></td>
Eltern-elemente:	tr
Kindelemente:	a, abbr, acronym, address, b, bdo, big, blockquote, br, cite, code, del, dfn, div, dl, em, form, h1-h6, hr, i, img, ins, kbd, noscript, object, ol, p, pre, q, samp, script, small, span, strong, sub, sup, table, tt, ul, var
Attribute:	abbr, align, axis, char, charoff, class, colspan, dir, headers, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, rowspan, scope, style, title, valign

Tabelle 16.64: Referenz zum td-Element

textarea

Bedeutung:	Mehrzeiliger Eingabebereich in einem Formular
Notation:	<textarea [Attribute]></textarea>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, cite, code, dd, del, dfn, div, dt, em, fieldset, h1-h6, i, ins, kbd, label, li, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	–
Attribute:	accesskey, class, cols, dir, disabled, id, lang, name, onblur, onchange, onclick, ondblclick, onfocus, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, onselect, readonly, rows, style, tabindex, title

Tabelle 16.65: Referenz zum textarea-Element

tfoot

Bedeutung:	Tabellenfuß
Notation:	<tfoot [Attribute]></tfoot>
Eltern-elemente:	table
Kindelemente:	tr
Attribute:	align, char, charoff, class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title, valign

Tabelle 16.66: Referenz zum tfoot-Element

th

Bedeutung:	Kopfzelle in einer Tabelle
Notation:	<th [Attribute]></th>
Eltern-elemente:	tr
Kindelemente:	a, abbr, acronym, address, b, bdo, big, blockquote, br, cite, code, del, dfn, div, dl, em, form, h1-h6, hr, i, img, ins, kbd, noscript, object, ol, p, pre, q, samp, script, small, span, strong, sub, sup, table, tt, ul, var
Attribute:	abbr, align, axis, char, charoff, class, colspan, dir, headers, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, rowspan, scope, style, title, valign

Tabelle 16.67: Referenz zum th-Element

thead

Bedeutung:	Tabellenkopf
Notation:	<thead [Attribute]></thead>
Eltern-elemente:	table
Kindelemente:	tr
Attribute:	align, char, charoff, class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title, valign

Tabelle 16.68: Referenz zum thead-Element

title

Bedeutung:	Titel eines HTML-Dokuments
Notation:	<title [Attribute]></title>
Eltern-elemente:	head
Kindelemente:	–
Attribute:	dir, lang

Tabelle 16.69: Referenz zum title-Element

tr

Bedeutung:	Tabellenzeile
Notation:	<tr [Attribute]></tr>
Eltern-elemente:	table, tbody, tfoot, thead
Kindelemente:	td, th
Attribute:	align, char, charoff, class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title, valign

Tabelle 16.70: Referenz zum tr-Element

tt

Bedeutung:	Dicktengleiche Schrift/Teletyper (Inline)
Notation:	<tt [Attribute]></tt>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, li, noscript, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, del, dfn, em, i, img, input, ins, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.71: Referenz zum tt-Element

ul

Bedeutung:	Aufzählungsliste
Notation:	<ul [Attribute]>
Eltern-elemente:	blockquote, body, button, dd, del, div, fieldset, form, ins, li, noscript, object, td, th
Kindelemente:	li
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.72: Referenz zum ul-Element

var

Bedeutung:	Variable, Name, Bezeichnung (Inline)
Notation:	<var [Attribute]></var>
Eltern-elemente:	a, abbr, acronym, address, b, bdo, big, blockquote, button, cite, code, dd, del, dfn, div, dt, em, fieldset, form, h1-h6, i, ins, kbd, label, li, noscript, object, p, pre, q, samp, small, span, strong, sub, sup, td, th, tt, ul, var
Kindelemente:	a, abbr, acronym, b, bdo, big, br, button, cite, code, del, dfn, em, i, img, input, ins, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var
Attribute:	class, dir, id, lang, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, style, title

Tabelle 16.73: Referenz zum var-Element

16.2 HTML-Attribute

Die Referenz behandelt alle HTML-Attribute, die zum HTML-Standard 4.01 strict gehören.

16.2.1 Hinweise zu den Tabellen

Die Referenzinformationen sind in Tabellen dieser Form enthalten:

Bedeutung:	Kurzinformationen, welche Bedeutung das Attribut hat
Elemente:	Aufzählung aller Elemente, denen das Attribut zugewiesen werden kann
Werte:	Werte, die dem Attribut zugewiesen werden können

Tabelle 16.74: Schema der Referenztabellen für HTML-Attribute

16.2.2 Attribute in alphabetischer Reihenfolge

abbr

Bedeutung:	Abkürzungstext für Tabellenkopfzelleninhalt
Elemente:	td, th
Werte:	Text (CDATA)

Tabelle 16.75: Referenz zum abbr-Attribut

accept-charset

Bedeutung:	Liste unterstützter Zeichensätze/Zeichenkodierungen
Elemente:	form
Werte:	Leerzeichengetrennte Liste mit Zeichensatzangaben

Tabelle 16.76: Referenz zum accept-charset-Attribut

accept

Bedeutung:	Liste erlaubter Mime-Typen für Datei-Upload
Elemente:	form, input
Werte:	Kommagetrennte Liste von Mime-Typen

Tabelle 16.77: Referenz zum accept-Attribut

accesskey

Bedeutung:	Hotkey für Direktzugriff über Tastatur
Elemente:	a, area, button, input, label, legend, textarea
Werte:	Einzelnes Zeichen (gewünschter Buchstabe, Ziffer)

Tabelle 16.78: Referenz zum accesskey-Attribut

action

Bedeutung:	Script, das bei Absenden eines Formulars aufgerufen wird
Elemente:	form
Werte:	URI-Angabe

Tabelle 16.79: Referenz zum action-Attribut

align

Bedeutung:	horizontale Ausrichtung
Elemente:	col, colgroup, tbody, td, tfoot, th, thead, tr
Werte:	Einer der Werte: left | center | right | justify | char

Tabelle 16.80: Referenz zum align-Attribut

alt

Bedeutung:	Alternativtext, falls Element nicht angezeigt werden kann
Elemente:	area, img, input
Werte:	Text (CDATA)

Tabelle 16.81: Referenz zum alt-Attribut

archive

Bedeutung:	Liste von URIs von Archivdateien
Elemente:	object
Werte:	Leerzeichengetrennte Liste mit URIs

Tabelle 16.82: Referenz zum archive-Attribut

axis

Bedeutung:	Logische Kategorie, zu denen eine Tabellenzelle gehört
Elemente:	td, th
Werte:	Name, Bezeichnung der Kategorie (CDATA)

Tabelle 16.83: Referenz zum axis-Attribut

border

Bedeutung:	Rahmen bzw. Rahmendicke
Elemente:	table
Werte:	Ganzzahl (Anzahl Pixel für Rahmendicke)

Tabelle 16.84: Referenz zum border-Attribut

cellpadding

Bedeutung:	Tabellenweiter Innenabstand zwischen Tabellenzellenrand und Zelleninhalt
Elemente:	table
Werte:	Ganzzahl (Anzahl Pixel) oder Prozentwert

Tabelle 16.85: Referenz zum cellpadding-Attribut

cellspacing

Bedeutung:	Tabellenweiter Abstand zwischen Tabellenzellen
Elemente:	table
Werte:	Ganzzahl (Anzahl Pixel) oder Prozentwert

Tabelle 16.86: Referenz zum cellspacing-Attribut

char

Bedeutung:	Zeichen, an dem der Inhalt betroffener Zellen ausgerichtet werden soll
Elemente:	col, colgroup, tbody, td, tfoot, th, thead, tr
Werte:	Einzelnes Zeichen, z. B. ein Punkt oder ein Komma

Tabelle 16.87: Referenz zum chat-Attribut

charoff

Bedeutung:	Offset-Position, an der ein Ausrichtungszeichen frühestens vorkommen kann
Elemente:	col, colgroup, tbody, td, tfoot, th, thead, tr
Werte:	Ganzzahl

Tabelle 16.88: Referenz zum charoff-Attribut

charset

Bedeutung:	Zeichensatz der verlinkten Quelle
Elemente:	a, link, script
Werte:	Zeichensatzangabe nach RFC 2045

Tabelle 16.89: Referenz zum charset-Attribut

checked

Bedeutung:	Radiobutton oder Checkbox per Default ausgewählt anzeigen
Elemente:	input
Werte:	Bei HTML keine Wertzuweisung, bei XHTML: checked=»checked«

Tabelle 16.90: Referenz zum checked-Attribut

cite

Bedeutung:	URI einer Zitatquelle oder Angabe eines Grunds
Elemente:	blockquote, del, ins, q
Werte:	Bei blockquote und q ein URI, bei del und ins Text (CDATA)

Tabelle 16.91: Referenz zum cite-Attribut

class

Bedeutung:	Klassennamen, z.B. für CSS-Formate
Elemente:	a, abbr, acronym, address, area, b, bdo, big, blockquote, body, br, button, caption, cite, code, col, colgroup, dd, del, dfn, div, dl, dt, em, fieldset, form, h1-h6, hr, i, img, input, ins, kbd, label, legend, li, link, map, noscript, object, ol, optgroup, option, p, pre, q, samp, select, small, span, strong, sub, sup, table, tbody, td, textarea, tfoot, th, thead, tr, tt, ul, var
Werte:	Ein oder mehrere Klassennamen (mehrere Klassennamen leerzeichengetrennt)

Tabelle 16.92: Referenz zum class-Attribut

classid

Bedeutung:	Implementierung einer Ressource (z.B. für ActiveX-Controls)
Elemente:	object
Werte:	Angabe entsprechend der jeweiligen Implementierungsvorschrift

Tabelle 16.93: Referenz zum abbr-Attribut

codebase

Bedeutung:	Adressbasis für Datei oder Ressource eines Objekts
Elemente:	object
Werte:	URI

Tabelle 16.94: Referenz zum codebase-Attribut

codetype

Bedeutung:	Mime-Type von Datei oder Ressource eines Objekts
Elemente:	object
Werte:	Mime-Type-Angabe

Tabelle 16.95: Referenz zum codetype-Attribut

cols

Bedeutung:	Anzahl Zeichen pro Zeile in einem mehrzeiligen Eingabefeld
Elemente:	textarea
Werte:	Ganzzahl

Tabelle 16.96: Referenz zum cols-Attribut

colspan

Bedeutung:	Anzahl Spalten, über die sich eine Tabellenzelle erstrecken soll
Elemente:	td, th
Werte:	Ganzzahl

Tabelle 16.97: Referenz zum cols-Attribut

content

Bedeutung:	Wertzuweitung bei einer Meta-Angabe
Elemente:	meta
Werte:	Der Wert hängt von der Art der Meta-Angabe ab

Tabelle 16.98: Referenz zum content-Attribut

coords

Bedeutung:	Liste von Längen oder Positionspunkten eines verweis-sensitiven Bereichs
Elemente:	area
Werte:	Kommagetrennte Liste von Zahlenangaben

Tabelle 16.99: Referenz zum coords-Attribut

data

Bedeutung:	Datenquelle eines Objekts
Elemente:	object
Werte:	URI

Tabelle 16.100: Referenz zum data-Attribut

datetime

Bedeutung:	Zeitstempel einer Änderung
Elemente:	del, ins
Werte:	Zeitstempel nach ISO-Norm, z. B. 2005-04-14T16:50:00+01:00

Tabelle 16.101: Referenz zum datetime-Attribut

declare

Bedeutung:	Objekt nur deklarieren, aber keine Instanz erzeugen
Elemente:	object
Werte:	Bei HTML keine Wertzuweisung, bei XHTML: declare=»declare«

Tabelle 16.102: Referenz zum declare-Attribut

defer

Bedeutung:	Verhindert Ausgaben eines Scripts
Elemente:	script
Werte:	Bei HTML keine Wertzuweisung, bei XHTML: defer=»defer«

Tabelle 16.103: Referenz zum defer-Attribut

dir

Bedeutung:	Textrichtung
Elemente:	a, abbr, acronym, address, area, b, bdo, big, blockquote, body, button, caption, cite, code, col, colgroup, dd, del, dfn, div, dl, dt, em, fieldset, form, h1-h6, head, hr, html, i, img, input, ins, kbd, label, legend, li, link, map, meta, noscript, object, ol, optgroup, option, p, pre, q, samp, select, small, span, strong, style, sub, sup, table, tbody, td, textarea, tfoot, th, thead, title, tr, tt, ul, var
Werte:	Einer der Werte: ltr (links nach rechts) | rtl (rechts nach links)

Tabelle 16.104: Referenz zum dir-Attribut

disabled

Bedeutung:	Formularelement ausgrauen (für den Anwender sichtbar, doch nicht verfügbar)
Elemente:	button, input, optgroup, option, select, textarea
Werte:	Bei HTML keine Wertzuweisung, bei XHTML: disabled=»disabled«

Tabelle 16.105: Referenz zum disabled-Attribut

enctype

Bedeutung:	Mime-Type der in einem Formular übertragenen Daten
Elemente:	form
Werte:	Mime-Type-Angabe entsprechend RFC 2045

Tabelle 16.106: Referenz zum enctype-Attribut

for

Bedeutung:	Beschriftung für ein Formularfeld
Elemente:	label
Werte:	id-Name des zugehörigen Formularfelds (id=)

Tabelle 16.107: Referenz zum for-Attribut

frame

Bedeutung:	Rahmengestaltung einer Tabelle
Elemente:	table
Werte:	Einer der Werte: void (kein Rahmen) | above (nur oben) | below (nur unten) | hsides (nur links und rechts) | lhs (nur links) | rhs (nur rechts) | vsides (nur oben und unten) | box (rundum) | border (rundum)

Tabelle 16.108: Referenz zum frame-Attribut

headers

Bedeutung:	Herstellung einer Zuordnung zu Tabellenkopfzellen
Elemente:	td, th
Werte:	Ein oder mehrere id-Namen von th-Elementen (mehrere Namen durch Leerzeichen trennen)

Tabelle 16.109: Referenz zum headers-Attribut

height

Bedeutung:	Höhe einer Grafik oder eines Objekts
Elemente:	img, object
Werte:	Ganzzahl (Anzahl Pixel) oder Prozentwert

Tabelle 16.110: Referenz zum height-Attribut

href

Bedeutung:	URI einer verlinkten Quelle oder Adressbasis für Referenzen
Elemente:	a, area, base, link
Werte:	URI

Tabelle 16.111: Referenz zum href-Attribut

hreflang

Bedeutung:	Sprache einer verlinkten Quelle
Elemente:	a, link
Werte:	Sprachenkürzel entsprechend RFC 1766

Tabelle 16.112: Referenz zum hreflang-Attribut

http-equiv

Bedeutung:	Formulierung eines HTTP-Headers für die Server-Antwort
Elemente:	meta
Werte:	Gültiger HTTP-Header-Feldname

Tabelle 16.113: Referenz zum http-equiv-Attribut

id

Bedeutung:	Dokumentweit eindeutiger Name, z. B. geeignet für CSS oder JavaScript/DOM
Elemente:	a, abbr, acronym, address, area, b, bdo, big, blockquote, body, button, caption, cite, code, col, colgroup, dd, del, dfn, div, dl, dt, em, fieldset, form, h1-h6, hr, i, img, input, ins, kbd, label, legend, li, link, map, noscript, object, ol, optgroup, option, p, pre, q, samp, select, small, span, strong, sub, sup, table, tbody, td, textarea, tfoot, th, thead, tr, tt, ul, var
Werte:	Name, der mit einem Buchstaben a–z oder A–Z beginnt und nachfolgend nur aus weiteren solchen Buchstaben, Ziffern 0–9, Unterstrich und einigen weiteren Zeichen bestehen darf. Es ist zu empfehlen, nur Buchstaben, Ziffern und Unterstrich zu verwenden.

Tabelle 16.114: Referenz zum id-Attribut

ismap

Bedeutung:	Zeichnet eine Grafik als verweis-sensitiv mit serverseitiger Verarbeitung aus
Elemente:	img, input
Werte:	Bei HTML keine Wertzuweisung, bei XHTML: ismap=»ismap«

Tabelle 16.115: Referenz zum ismap-Attribut

label

Bedeutung:	Kurzbezeichnung/Titel einer Auswahlmöglichkeit
Elemente:	optgroup, option
Werte:	Text (CDATA)

Tabelle 16.116: Referenz zum label-Attribut

lang

Bedeutung:	Sprache des Elementinhalts
Elemente:	a, abbr, acronym, address, area, b, bdo, big, blockquote, body, button, caption, cite, code, col, colgroup, dd, del, dfn, div, dl, dt, em, fieldset, form, h1-h6, head, hr, html, i, img, input, ins, kbd, label, legend, li, link, map, meta, noscript, object, ol, optgroup, option, p, pre, q, samp, select, small, span, strong, style, sub, sup, table, tbody, td, textarea, tfoot, th, thead, title, tr, tt, ul, var
Werte:	Sprachenkürzel entsprechend RFC 1766

Tabelle 16.117: Referenz zum lang-Attribut

longdesc

Bedeutung:	Quelle einer Langbeschreibung zu einer Grafik
Elemente:	img
Werte:	URI

Tabelle 16.118: Referenz zum longdesc-Attribut

maxlength

Bedeutung:	Maximale Länge erlaubter Bytes im Formularfeld
Elemente:	input
Werte:	Ganzzahl

Tabelle 16.119: Referenz zum maxlength-Attribut

media

Bedeutung:	Nennung geeigneter Medientypen
Elemente:	link, style
Werte:	Einer oder mehrere der Werte: all (alle) | aural (akustische Medien) | braille (Braille-Blindenzeilen) | embossed (Braille-Drucker) | handheld (Kleinstcomputer) | print (Drucker) | projection (Beamer) | screen (Bildschirm) | tty (Textmonitore) | tv (Fernseher). Mehrere Werte durch Leerzeichen trennen.

Tabelle 16.120: Referenz zum media-Attribut

method

Bedeutung:	Übertragungsmethode für Formulardaten
Elemente:	form
Werte:	Einer der Werte: get | post

Tabelle 16.121: Referenz zum method-Attribut

multiple

Bedeutung:	Mehrfachauswahl in einer Auswahlliste
Elemente:	select
Werte:	Bei HTML keine Wertzuweisung, bei XHTML: multiple=»multiple«

Tabelle 16.122: Referenz zum multiple-Attribut

name

Bedeutung:	Name eines Elements, darf z. B. bei Radiobuttons auch mehrfach vorkommen
Elemente:	a, button, form, img, input, map, meta, object, param, select, textarea
Werte:	Name, der in Hinblick auf Verwendung in Programmiersprachen mit einem Buchstaben a–z oder A–Z beginnt und nachfolgend nur aus weiteren solchen Buchstaben, Ziffern 0–9, Unterstrich und einigen weiteren Zeichen bestehen sollte

Tabelle 16.123: Referenz zum name-Attribut

nohref

Bedeutung:	Verweisloser Bereich in einer verweis-sensitiven Grafik
Elemente:	area
Werte:	Bei HTML keine Wertzuweisung, bei XHTML: nohref=»nohref«

Tabelle 16.124: Referenz zum nohref-Attribut

onblur

Bedeutung:	Event-Handler »wenn Element den Fokus verliert«
Elemente:	a, area, button, input, label, select, textarea
Werte:	Script-Code (aus HTML-Sicht: CDATA)

Tabelle 16.125: Referenz zum onblur-Attribut

onchange

Bedeutung:	Event-Handler »nach Wertänderung des Elements«
Elemente:	input, select, textarea
Werte:	Script-Code (aus HTML-Sicht: CDATA)

Tabelle 16.126: Referenz zum onchange-Attribut

onclick

Bedeutung:	Event-Handler »bei Anklicken des Elementbereichs«
Elemente:	a, abbr, acronym, address, area, b, big, blockquote, body, button, caption, cite, code, col, colgroup, dd, del, dfn, div, dl, dt, em, fieldset, form, h1-h6, hr, i, img, input, ins, kbd, label, legend, li, link, map, noscript, object, ol, optgroup, option, p, pre, q, samp, select, small, span, strong, sub, sup, table, tbody, td, textarea, tfoot, th, thead, tr, tt, ul, var
Werte:	Script-Code (aus HTML-Sicht: CDATA)

Tabelle 16.127: Referenz zum onclick-Attribut

ondblclick

Bedeutung:	Event-Handler »bei Doppelklick in den Elementbereich«
Elemente:	a, abbr, acronym, address, area, b, big, blockquote, body, button, caption, cite, code, col, colgroup, dd, del, dfn, div, dl, dt, em, fieldset, form, h1-h6, hr, i, img, input, ins, kbd, label, legend, li, link, map, noscript, object, ol, optgroup, option, p, pre, q, samp, select, small, span, strong, sub, sup, table, tbody, td, textarea, tfoot, th, thead, tr, tt, ul, var
Werte:	Script-Code (aus HTML-Sicht: CDATA)

Tabelle 16.128: Referenz zum ondblclick-Attribut

onfocus

Bedeutung:	Event-Handler »wenn Element den Fokus erhält«
Elemente:	a, area, button, input, label, select, textarea
Werte:	Script-Code (aus HTML-Sicht: CDATA)

Tabelle 16.129: Referenz zum onfocus-Attribut

onkeydown

Bedeutung:	Event-Handler »bei Herunterdrücken einer Taste im Elementbereich«
Elemente:	a, abbr, acronym, address, area, b, big, blockquote, body, button, caption, cite, code, col, colgroup, dd, del, dfn, div, dl, dt, em, fieldset, form, h1-h6, hr, i, img, input, ins, kbd, label, legend, li, link, map, noscript, object, ol, optgroup, option, p, pre, q, samp, select, small, span, strong, sub, sup, table, tbody, td, textarea, tfoot, th, thead, tr, tt, ul, var
Werte:	Script-Code (aus HTML-Sicht: CDATA)

Tabelle 16.130: Referenz zum onkeydown-Attribut

onkeypress

Bedeutung:	Event-Handler »nach heruntergedrückter und wieder losgelassener Taste im Elementbereich«
Elemente:	a, abbr, acronym, address, area, b, big, blockquote, body, button, caption, cite, code, col, colgroup, dd, del, dfn, div, dl, dt, em, fieldset, form, h1-h6, hr, i, img, input, ins, kbd, label, legend, li, link, map, noscript, object, ol, optgroup, option, p, pre, q, samp, select, small, span, strong, sub, sup, table, tbody, td, textarea, tfoot, th, thead, tr, tt, ul, var
Werte:	Script-Code (aus HTML-Sicht: CDATA)

Tabelle 16.131: Referenz zum onkeypress-Attribut

onkeyup

Bedeutung:	Event-Handler »nach wieder losgelassener Taste im Elementbereich«
Elemente:	a, abbr, acronym, address, area, b, big, blockquote, body, button, caption, cite, code, col, colgroup, dd, del, dfn, div, dl, dt, em, fieldset, form, h1-h6, hr, i, img, input, ins, kbd, label, legend, li, link, map, noscript, object, ol, optgroup, option, p, pre, q, samp, select, small, span, strong, sub, sup, table, tbody, td, textarea, tfoot, th, thead, tr, tt, ul, var
Werte:	Script-Code (aus HTML-Sicht: CDATA)

Tabelle 16.132: Referenz zum onkeyup-Attribut

onload

Bedeutung:	Event-Handler »nachdem vollständig geladen«
Elemente:	body
Werte:	Script-Code (aus HTML-Sicht: CDATA)

Tabelle 16.133: Referenz zum onload-Attribut

onmousedown

Bedeutung:	Event-Handler »nach heruntergedrückter Maustaste im Elementbereich«
Elemente:	a, abbr, acronym, address, area, b, big, blockquote, body, button, caption, cite, code, col, colgroup, dd, del, dfn, div, dl, dt, em, fieldset, form, h1-h6, hr, i, img, input, ins, kbd, label, legend, li, link, map, noscript, object, ol, optgroup, option, p, pre, q, samp, select, small, span, strong, sub, sup, table, tbody, td, textarea, tfoot, th, thead, tr, tt, ul, var
Werte:	Script-Code (aus HTML-Sicht: CDATA)

Tabelle 16.134: Referenz zum onmousedown-Attribut

onmousemove

Bedeutung:	Event-Handler »beim Bewegen im Elementbereich mit der Maus«
Elemente:	a, abbr, acronym, address, area, b, big, blockquote, body, button, caption, cite, code, col, colgroup, dd, del, dfn, div, dl, dt, em, fieldset, form, h1-h6, hr, i, img, input, ins, kbd, label, legend, li, link, map, noscript, object, ol, optgroup, option, p, pre, q, samp, select, small, span, strong, sub, sup, table, tbody, td, textarea, tfoot, th, thead, tr, tt, ul, var
Werte:	Script-Code (aus HTML-Sicht: CDATA)

Tabelle 16.135: Referenz zum onmouseover-Attribut

onmouseout

Bedeutung:	Event-Handler »nach Verlassen des Elementbereichs mit der Maus«
Elemente:	a, abbr, acronym, address, area, b, big, blockquote, body, button, caption, cite, code, col, colgroup, dd, del, dfn, div, dl, dt, em, fieldset, form, h1-h6, hr, i, img, input, ins, kbd, label, legend, li, link, map, noscript, object, ol, optgroup, option, p, pre, q, samp, select, small, span, strong, sub, sup, table, tbody, td, textarea, tfoot, th, thead, tr, tt, ul, var
Werte:	Script-Code (aus HTML-Sicht: CDATA)

Tabelle 16.136: Referenz zum onmouseout-Attribut

onmouseover

Bedeutung:	Event-Handler »beim Erreichen des Elementbereichs mit der Maus«
Elemente:	a, abbr, acronym, address, area, b, big, blockquote, body, button, caption, cite, code, col, colgroup, dd, del, dfn, div, dl, dt, em, fieldset, form, h1-h6, hr, i, img, input, ins, kbd, label, legend, li, link, map, noscript, object, ol, optgroup, option, p, pre, q, samp, select, small, span, strong, sub, sup, table, tbody, td, textarea, tfoot, th, thead, tr, tt, ul, var
Werte:	Script-Code (aus HTML-Sicht: CDATA)

Tabelle 16.137: Referenz zum onmouseout-Attribut

onmouseup

Bedeutung:	Event-Handler »nach wieder losgelassener Maustaste im Elementbereich«
Elemente:	a, abbr, acronym, address, area, b, big, blockquote, body, button, caption, cite, code, col, colgroup, dd, del, dfn, div, dl, dt, em, fieldset, form, h1-h6, hr, i, img, input, ins, kbd, label, legend, li, link, map, noscript, object, ol, optgroup, option, p, pre, q, samp, select, small, span, strong, sub, sup, table, tbody, td, textarea, tfoot, th, thead, tr, tt, ul, var
Werte:	Script-Code (aus HTML-Sicht: CDATA)

Tabelle 16.138: Referenz zum onmouseup-Attribut

onreset

Bedeutung:	Event-Handler »beim Zurücksetzen eines Formulars«
Elemente:	form
Werte:	Script-Code (aus HTML-Sicht: CDATA)

Tabelle 16.139: Referenz zum onreset-Attribut

onselect

Bedeutung:	Event-Handler »bei selektiertem Text«
Elemente:	input, textarea
Werte:	Script-Code (aus HTML-Sicht: CDATA)

Tabelle 16.140: Referenz zum onselect-Attribut

onsubmit

Bedeutung:	Event-Handler »beim Absenden eines Formulars«
Elemente:	form
Werte:	Script-Code (aus HTML-Sicht: CDATA)

Tabelle 16.141: Referenz zum onsubmit-Attribut

onunload

Bedeutung:	Event-Handler »beim Verlassen der angezeigten Seite«
Elemente:	body
Werte:	Script-Code (aus HTML-Sicht: CDATA)

Tabelle 16.142: Referenz zum onunload-Attribut

profile

Bedeutung:	Datenquelle mit Profilangaben für Meta-Daten
Elemente:	head
Werte:	URI

Tabelle 16.143: Referenz zum profile-Attribut

readonly

Bedeutung:	Formularelement kann nur betrachtet, aber nicht geändert werden
Elemente:	input, textarea
Werte:	Bei HTML keine Wertzuweisung, bei XHTML: readonly=»readonly«

Tabelle 16.144: Referenz zum readonly-Attribut

rel

Bedeutung:	Link eines der unten genannten Typen, unidirektionaler Link
Elemente:	a, link
Werte:	Einer der Werte: contents (Inhaltsverzeichnis) | chapter (Kapitel) | section (Abschnitt) | subsection (Unterabschnitt) | index (Stichwortverzeichnis) | glossary (Glossar) | appendix (Anhang) | search (Suche) | author (Autor) | copyright (Rechtshinweise) | next (nächste Seite) | prev (vorherige Seite) | first (erste Seite) | last (letzte Seite) | up (übergeordnete Seite) | top (Startseite) | help (Hilfe) | bookmark (verwandte Seite) | stylesheet (Stylesheet-Datei) | alternate (andere Fassung der Seite) | alternate stylesheet (alternative Stylesheet-Datei) | shortcut icon (Symbol)

Tabelle 16.145: Referenz zum rel-Attribut

rev

Bedeutung:	Link eines der unten genannten Typen, bidirektionaler Link
Elemente:	a, link
Werte:	Einer der Werte: contents (Inhaltsverzeichnis) | chapter (Kapitel) | section (Abschnitt) | subsection (Unterabschnitt) | index (Stichwortverzeichnis) | glossary (Glossar) | appendix (Anhang) | search (Suche) | author (Autor) | copyright (Rechtshinweise) | next (nächste Seite) | prev (vorherige Seite) | first (erste Seite) | last (letzte Seite) | up (übergeordnete Seite) | top (Startseite) | help (Hilfe) | bookmark (verwandte Seite) | stylesheet (Stylesheet-Datei) | alternate (andere Fassung der Seite) | alternate stylesheet (alternative Stylesheet-Datei) | shortcut icon (Symbol)

Tabelle 16.146: Referenz zum rev-Attribut

rows

Bedeutung:	Anzahl Zeilen in einem mehrzeiligen Eingabefeld
Elemente:	textarea
Werte:	Ganzzahl

Tabelle 16.147: Referenz zum rows-Attribut

rowspan

Bedeutung:	Anzahl Zeilen, über die sich eine Tabellenzelle erstrecken soll
Elemente:	td, th
Werte:	Ganzzahl

Tabelle 16.148: Referenz zum rowspan-Attribut

rules

Bedeutung:	Regel zur Gitternetzdarstellung bei einer Tabelle
Elemente:	table
Werte:	Einer der Werte: none (keine Gitternetzlinien) | groups (nur zwischen Kopf, Körper und Fuß der Tabelle) | rows (nur zwischen Zeilen) | cols (nur zwischen Spalten) | all (vollständiges Gitternetz)

Tabelle 16.149: Referenz zum rules-Attribut

scheme

Bedeutung:	Format der Meta-Angabe
Elemente:	meta
Werte:	Text (CDATA)

Tabelle 16.150: Referenz zum scheme-Attribut

scope

Bedeutung:	Erstreckungsbereich der (zugehörigen) Kopfzelle
Elemente:	td, th
Werte:	Einer der Werte: row (Zeile) | col (Spalte) | rowgroup (Zeilengruppe) | colgroup (Spaltengruppe)

Tabelle 16.151: Referenz zum scope-Attribut

selected

Bedeutung:	Vorausgewählter Eintrag einer Auswahlliste
Elemente:	option
Werte:	Bei HTML keine Wertzuweisung, bei XHTML: selected=»selected«

Tabelle 16.152: Referenz zum selected-Attribut

shape

Bedeutung:	Art eines verweis-sensitiven Bereichs
Elemente:	a, area
Werte:	Einer der Werte: rect (Rechteck) | circle (Kreis) | poly (Mehreck, Polygon)

Tabelle 16.153: Referenz zum shape-Attribut

size

Bedeutung:	Bei input-Feldern die Anzahl anzuzeigender Zeichen, bei Auswahllisten die Anzahl anzuzeigender Einträge
Elemente:	input, select
Werte:	Ganzzahl

Tabelle 16.154: Referenz zum size-Attribut

span

Bedeutung:	Bei col Anzahl nachfolgend von Attributen betroffener Spalten, bei colgroup Anzahl Spalten in der Spaltengruppe
Elemente:	col, colgroup
Werte:	Ganzzahl

Tabelle 16.155: Referenz zum span-Attribut

src

Bedeutung:	Datenquelle eines Scripts oder einer Grafik
Elemente:	img, input, script
Werte:	URI

Tabelle 16.156: Referenz zum src-Attribut

standby

Bedeutung:	Angezeigte Meldung, während ein Objekt geladen wird
Elemente:	object
Werte:	Meldungstext (CDATA)

Tabelle 16.157: Referenz zum standby-Attribut

style

Bedeutung:	Style-Formatdefinitionen eines Elements
Elemente:	a, abbr, acronym, address, area, b, bdo, big, blockquote, body, br, button, caption, cite, code, col, colgroup, dd, del, dfn, div, dl, dt, em, fieldset, form, h1-h6, hr, i, img, input, ins, kbd, label, legend, li, link, map, noscript, object, ol, optgroup, option, p, pre, q, samp, select, small, span, strong, sub, sup, table, tbody, td, textarea, tfoot, th, thead, tr, tt, ul, var
Werte:	Formatdefinitionen entsprechend der Style-Sprache (Default: CSS)

Tabelle 16.158: Referenz zum style-Attribut

summary

Bedeutung:	Zusammenfassende Beschreibung eines Tabelleninhalts
Elemente:	table
Werte:	Text (CDATA)

Tabelle 16.159: Referenz zum summary-Attribut

tabindex

Bedeutung:	Positionsnummer beim Durchsteppen mit der Tab-Taste
Elemente:	a, area, button, input, object, select, textarea
Werte:	Ganzzahl

Tabelle 16.160: Referenz zum tabindex-Attribut

title

Bedeutung:	Kurzinfo zu einem Element (kann als Tooltipp angezeigt werden)
Elemente:	a, abbr, acronym, address, area, b, bdo, big, blockquote, body, br, button, caption, cite, code, col, colgroup, dd, del, dfn, div, dl, dt, em, fieldset, form, h1-h6, hr, i, img, input, ins, kbd, label, legend, li, link, map, noscript, object, ol, optgroup, option, p, pre, q, samp, select, small, span, strong, style, sub, sup, table, tbody, td, textarea, tfoot, th, thead, tr, tt, ul, var
Werte:	x, y, z

Tabelle 16.161: Referenz zum title-Attribut

type

Bedeutung:	Mime-Type des Inhalts bzw. der Quelle oder (bei input) Typ des Eingabefelds
Elemente:	a, input, link, object, param, script, style
Werte:	Bei a, link, object, param, script und style eine Mime-Type-Angabe entsprechend RFC 2045, bei input einer der Werte: button (Button für Scriptanbindung) | checkbox (Checkbox) | file (Feld für Datei-Upload) | hidden (verstecktes Feld) | password (Textfeld mit verdeckter Eingabe) | radio (Radiobutton) | reset (Zurücksetzen-Button) | submit (Absendebutton)

Tabelle 16.162: Referenz zum type-Attribut

usemap

Bedeutung:	Link zu einem map-Element für verweis-sensitive Grafiken
Elemente:	img, input, object
Werte:	URI

Tabelle 16.163: Referenz zum usemap-Attribut

valign

Bedeutung:	Vertikale Ausrichtung von betroffenen Tabellenzellen
Elemente:	col, colgroup, tbody, td, tfoot, th, thead, tr
Werte:	Einer der Werte: top (obenbündig) | middle (mittig) | bottom (untenbündig) | baseline (zur Textbasislinie)

Tabelle 16.164: Referenz zum valign-Attribut

value

Bedeutung:	Wert oder Beschriftung eines Parameters oder Formularelements
Elemente:	button, input, option, param
Werte:	Text (CDTATA)

Tabelle 16.165: Referenz zum value-Attribut

valuetype

Bedeutung:	Bei Parametern ein Hinweis, wie der Wert zu interpretieren ist
Elemente:	param
Werte:	Einer der Werte: data (Daten) | ref (Referenz auf Daten) | object (Objekt)

Tabelle 16.166: Referenz zum valuetype-Attribut

version

Bedeutung:	Benutzte HTML-Version
Elemente:	html
Werte:	HTML- oder XHTML-Version (z.B. 4.01 oder 1.0)

Tabelle 16.167: Referenz zum version-Attribut

width

Bedeutung:	Anzeigebreite einer Grafik, eines Objekts, einer Tabelle oder einer Spalte oder Spaltengruppe
Elemente:	col, colgroup, img, object, table
Werte:	Ganzzahl (Anzahl Pixel) oder Prozentwert

Tabelle 16.168: Referenz zum width-Attribut

16.3 Benannte HTML-Entities

Die Referenz behandelt alle HTML-Entities von HTML 4.01.

16.3.1 Hinweise zu den Tabellen

Die Tabellen haben folgenden Aufbau:

Zeichen	Bedeutung	HTML-Entity
Darzustellendes Zeichen	Kurzbeschreibung des Zeichens	Zeichenkette, die in HTML zur Darstellung des Zeichens notiert werden muss

Tabelle 16.169: Schema der Referenztabellen für HTML-Entities

16.3.2 Benannte Entities

Die Tabellen sind nach Art/Funktion der Zeichen gegliedert.

Benannte Entities für HTML-eigene Zeichen

Zeichen	Bedeutung	HTML-Entity
"	Anführungszeichen oben	"
&	Ampersand-Zeichen, kaufmännisches Und	&
<	Öffnende spitze Klammer	<
>	Schließende spitze Klammer	>

Tabelle 16.170: Benannte Entities für HTML-eigene Zeichen

Benannte Entities für Zeichen aus Zeichensatz ISO-8859-1

Zeichen	Bedeutung	HTML-Entity
	erzwungenes Leerzeichen	
¡	umgekehrtes Ausrufezeichen	¡
¢	Cent-Zeichen	¢
£	Pfund-Zeichen	£
¤	Währungszeichen	¤
¥	Yen-Zeichen	¥
¦	durchbrochener Strich	¦
§	Paragraph-Zeichen	§
¨	Pünktchen oben	¨
©	Copyright-Zeichen	©
ª	Ordinal-Zeichen weiblich	ª
«	angewinkelte Anführungszeichen links	«
¬	Verneinungs-Zeichen	¬
	bedingter Trennstrich	
®	Registriermarke-Zeichen	®
¯	Überstrich	¯
°	Grad-Zeichen	°
±	Plusminus-Zeichen	±
²	Hoch-2-Zeichen	²
³	Hoch-3-Zeichen	³
´	Acute-Zeichen	´
µ	Mikro-Zeichen	µ
¶	Absatz-Zeichen	¶
·	Mittelpunkt	·
¸	Häkchen unten	¸
¹	Hoch-1-Zeichen	¹
º	Ordinal-Zeichen männlich	º
»	angewinkelte Anführungszeichen rechts	»
¼	ein Viertel	¼
½	ein Halb	½
¾	drei Viertel	¾
¿	umgekehrtes Fragezeichen	¿
À	A mit Accent grave	À
Á	A mit Accent acute	Á
Â	A mit Circumflex	Â

Tabelle 16.171: Benannte Entities für Zeichen aus Zeichensatz ISO-8859-1

Zeichen	Bedeutung	HTML-Entity
Ã	A mit Tilde	Ã
Ä	A Umlaut	Ä
Å	A mit Ring	Å
Æ	A mit legiertem E	Æ
Ç	C mit Häkchen	Ç
È	E mit Accent grave	È
É	E mit Accent acute	É
Ê	E mit Circumflex	Ê
Ë	E Umlaut	Ë
Ì	I mit Accent grave	Ì
Í	I mit Accent acute	Í
Î	I mit Circumflex	Î
Ï	I Umlaut	Ï
Ð	Eth (isländisch)	Ð
Ñ	N mit Tilde	Ñ
Ò	O mit Accent grave	Ò
Ó	O mit Accent acute	Ó
Ô	O mit Circumflex	Ô
Õ	O mit Tilde	Õ
Ö	O Umlaut	Ö
×	Mal-Zeichen	×
Ø	O mit Schrägstrich	Ø
Ù	U mit Accent grave	Ù
Ú	U mit Accent acute	Ú
Û	U mit Circumflex	Û
Ü	U Umlaut	Ü
Ý	Y mit Accent acute	Ý
Þ	THORN (isländisch)	Þ
ß	scharfes S	ß
à	a mit Accent grave	à
á	a mit Accent acute	á
â	a mit Circumflex	â
ã	a mit Tilde	ã
ä	a Umlaut	ä
å	a mit Ring	å
æ	a mit legiertem e	æ
ç	c mit Häkchen	ç

Tabelle 16.171: Benannte Entities für Zeichen aus Zeichensatz ISO-8859-1 (Fortsetzung)

Zeichen	Bedeutung	HTML-Entity
è	e mit Accent grave	è
é	e mit Accent acute	é
ê	e mit Circumflex	ê
ë	e Umlaut	ë
ì	i mit Accent grave	ì
í	i mit Accent acute	í
î	i mit Circumflex	î
ï	i Umlaut	ï
ð	eth (isländisch)	ð
ñ	n mit Tilde	ñ
ò	o mit Accent grave	ò
ó	o mit Accent acute	ó
ô	o mit Circumflex	ô
õ	o mit Tilde	õ
ö	o Umlaut	ö
÷	Divisions-Zeichen	÷
ø	o mit Schrägstrich	ø
ù	u mit Accent grave	ù
ú	u mit Accent acute	ú
û	u mit Circumflex	û
ü	u Umlaut	ü
ý	y mit Accent acute	ý
þ	thorn (isländisch)	þ

Tabelle 16.171: Benannte Entities für Zeichen aus Zeichensatz ISO-8859-1 (Fortsetzung)

Benannte Entities für griechische Buchstaben

Zeichen	Bedeutung	HTML-Entity
Α	Alpha groß	Α
α	alpha klein	α
Β	Beta groß	Β
β	beta klein	β
Γ	Gamma groß	Γ
γ	gamma klein	γ
Δ	Delta groß	Δ
δ	delta klein	δ
Ε	Epsilon groß	Ε

Tabelle 16.172: Benannte Entities für griechische Buchstaben

Zeichen	Bedeutung	HTML-Entity
ε	epsilon klein	ε
Z	Zeta groß	Ζ
ζ	zeta klein	ζ
H	Eta groß	Η
η	eta klein	η
Θ	Theta groß	Θ
θ	theta klein	θ
I	Iota groß	Ι
ι	iota klein	ι
K	Kappa groß	Κ
κ	kappa klein	κ
Λ	Lambda groß	Λ
λ	lambda klein	λ
M	My groß	Μ
μ	my klein	μ
N	Ny groß	Ν
ν	ny klein	ν
Ξ	Xi groß	Ξ
ξ	xi klein	ξ
O	Omikron groß	Ο
ο	omikron klein	ο
Π	Pi groß	Π
π	pi klein	π
P	Rho groß	Ρ
ρ	rho klein	ρ
Σ	Sigma groß	Σ
ς	Schluss-Sigma	ς
σ	sigma klein	σ
T	Tau groß	Τ
τ	tau klein	τ
Y	Ypsilon groß	Υ
υ	ypsilon klein	υ
Φ	Phi groß	Φ
ϕ	phi klein	φ
X	Chi groß	Χ
χ	chi klein	χ
Ψ	Psi groß	Ψ

Tabelle 16.172: Benannte Entities für griechische Buchstaben (Fortsetzung)

Zeichen	Bedeutung	HTML-Entity
ψ	psi klein	ψ
Ω	Omega groß	Ω
ω	omega klein	ω
ϑ	theta Symbol	ϑ
ϒ	ypsilon mit Haken	ϒ
ϖ	pi-Symbol	ϖ

Tabelle 16.172: Benannte Entities für griechische Buchstaben (Fortsetzung)

Benannte Entities für mathematische Symbole

Zeichen	Bedeutung	HTML-Entity
∀	für alle	∀
∂	partiell	∂
∃	existiert	∃
∅	leere Menge	∅
∇	Nabla	∇
∈	Element von	∈
∉	kein Element von	∉
∋	Element von (gespiegelt)	∋
∏	Produkt	∏
∑	Summe	∑
−	minus	−
*	Asterisk	∗
√	Wurzel	√
∝	proportional zu	∝
∞	unendlich	∞
∠	Winkel	∠
&	und	∧
?	oder	∨
∩	Schnittmenge	∩
∪	Vereinigungsmenge	∪
∫	Integral	∫
∴	deshalb	∴
~	ähnlich wie	∼
≅	kongruent	≅
≈	beinahe gleich	≈
≠	ungleich	≠

Tabelle 16.173: Benannte Entities für mathematische Symbole

Zeichen	Bedeutung	HTML-Entity
≡	identisch mit	≡
≤	kleiner gleich	≤
≥	größer gleich	≥
⊃	Untermenge von	⊂
⊇	Obermenge von	⊃
⊄	keine Untermenge von	⊄
⊆	Untermenge von oder gleich mit	⊆
⊇	Obermenge von oder gleich mit	⊇
⊕	Restklassenaddition	⊕
⊗	Kronecker-Produkt	⊗
⊥	senkrecht zu	⊥
·	Punkt-Operator	⋅
◊	Raute	◊

Tabelle 16.173: Benannte Entities für mathematische Symbole (Fortsetzung)

Benannte Entities für technische Symbole

Zeichen	Bedeutung	HTML-Entity
⌈	links oben	⌈
⌉	rechts oben	⌉
⌊	links unten	⌊
⌋	rechts unten	⌋
<	spitze Klammer links	⟨
>	spitze Klammer rechts	⟩

Tabelle 16.174: Benannte Entities für technische Symbole

Benannte Entities für Pfeilsymbole

Zeichen	Bedeutung	HTML-Entity
←	Pfeil links	←
↑	Pfeil oben	↑
→	Pfeil rechts	→
↓	Pfeil unten	↓
↔	Pfeil links/rechts	↔
↵	Pfeil unten-Knick-links	↵
⇐	Doppelpfeil links	⇐

Tabelle 16.175: Benannte Entities für Pfeilsymbole

Zeichen	Bedeutung	HTML-Entity
⇑	Doppelpfeil oben	⇑
⇒	Doppelpfeil rechts	⇒
⇓	Doppelpfeil unten	⇓
⇔	Doppelpfeil links/rechts	⇔

Tabelle 16.175: Benannte Entities für Pfeilsymbole (Fortsetzung)

Benannte Entities für diverse Symbole

Zeichen	Bedeutung	HTML-Entity
•	Bullet-Zeichen	•
′	Minutenzeichen	′
‾	Überstrich	‾
⁄	Bruchstrich	⁄
℘	Weierstraß p	℘
ℑ	Zeichen für »imaginär«	ℑ
ℜ	Zeichen für »real«	ℜ
™	Trademark-Zeichen	™
€	Euro-Zeichen	€
ℵ	Alef-Symbol	ℵ
♠	Pik-Zeichen	♠
♣	Kreuz-Zeichen	♣
♥	Herz-Zeichen	♥
♦	Karo-Zeichen	♦

Tabelle 16.176: Benannte Entities für diverse Symbole

Benannte Entities für Interpunktionszeichen

Zeichen	Bedeutung	HTML-Entity
	Leerzeichen Breite n	
	Leerzeichen Breite m	
	schmales Leerzeichen	
	null breiter Nichtverbinder	‌
	null breiter Verbinder	‍
	links-nach-rechts-Zeichen	‎
	rechts-nach-links-Zeichen	‏
–	Gedankenstrich Breite n	–
—	Gedankenstrich Breite m	—

Tabelle 16.177: Benannte Entities für Interpunktionszeichen

Zeichen	Bedeutung	HTML-Entity
‘	einfaches Anführungszeichen links	‘
’	einfaches Anführungszeichen rechts	’
‚	einfaches low-9-Zeichen	‚
«	doppeltes Anführungszeichen links	“
»	doppeltes Anführungszeichen rechts	”
„	doppeltes low-9-Zeichen rechts	„
†	Kreuz	†
‡	Doppelkreuz	‡
…	horizontale Ellipse (Auslassungszeichen)	…
‰	Promille	‰
‹	angewinkeltes einzelnes Anf.zeichen links	‹
›	angewinkeltes einzelnes Anf.zeichen rechts	›

Tabelle 16.177: Benannte Entities für Interpunktionszeichen (Fortsetzung)

17 CSS-Referenz

Die Referenz behandelt die CSS-Eigenschaften des Standards 2.1.

17.1 CSS-Eigenschaften

Die Referenz behandelt die CSS-Eigenschaften des Standards 2.1 in alphabetischer Reihenfolge.

17.1.1 Hinweise zu den Tabellen

Die Referenzinformationen sind in Tabellen dieser Form enthalten:

Bedeutung:	Kurzinformationen, welche Bedeutung die Eigenschaft hat.
Typ:	Medientyp (visuell, aural oder printorientiert).
Vererbung:	Ja, wenn ein Wert automatisch von dem eines Elternelements geerbt wird, nein, wenn nicht.
Elemente:	Angabe, welchen HTML-Elementen die Eigenschaft zugeordnet werden kann.
Werte:	Werte, die der Eigenschaft zugewiesen werden können; **fett** dargestellte Werte sind die voreingestellten Default-Werte

Tabelle 17.1: Schema der Referenztabellen für CSS-Eigenschaften

17.1.2 CSS-Eigenschaften in alphabetischer Reihenfolge

azimuth

Bedeutung:	Raum-Effekt links/rechts/vorne/hinten
Typ:	Aural
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Zahl 0-360, gefolgt von der Maßeinheit deg, grad, oder rad, oder einer der folgenden Werte: left-side (von linksseitig) | far-left (von weit links) | left (von links) | center-left (von Mitte links) | **center** (aus der Mitte) | center-right (von Mitte rechts) | right (von rechts) | far-right (von weit rechts) | right-side (von rechtsseitig) | behind (von hinten) | leftwards (von links) | rightwards (von rechts) | inherit (wie Elternelement)

Tabelle 17.2: Referenz zur CSS-Eigenschaft »azimuth«

background-attachment

Bedeutung:	Wasserzeichen-Effekt
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	einer der folgenden Werte: **scroll** (Hintergrundbild scrollt mit) | fixed (Hintergrundbild bleibt stehen) | inherit (wie Elternelement)

Tabelle 17.3: Referenz zur CSS-Eigenschaft »background-attachment«

background-color

Bedeutung:	Hintergrundfarbe
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Farbangabe oder einer der folgenden Werte: **transparent** (farblos/durchsichtig) | inherit (wie Elternelement)

Tabelle 17.4: Referenz zur CSS-Eigenschaft »background-color«

background-image

Bedeutung:	Hintergrundgrafik
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	url (URI) der Grafik oder einer der folgenden Werte: **none** (kein Hintergrundbild) | inherit (wie Elternelement)

Tabelle 17.5: Referenz zur CSS-Eigenschaft »background-image«

background-position

Bedeutung:	Position des Hintergrundbilds
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist

Tabelle 17.6: Referenz zur CSS-Eigenschaft »background-position«

Werte:	**top left** oder left top (links oben), top center, center top oder nur center (Mitte oben), right top oder top right (rechts oben), left center, center left oder nur left (links mittig), center center oder nur center (Mitte mittig), right center, center right oder nur right (rechts mittig), bottom left oder left bottom (links unten), bottom center, center bottom oder nur bottom (Mitte unten), right bottom oder bottom right (rechts unten), oder zwei durch Leerzeichen getrennte numerische Angaben mit Maßeinheit (auch Prozentwerte erlaubt) für »Abstand von der linken oberen Ecke« (der erste Wert bedeutet den Linksabstand, der zweite den Obenabstand)

Tabelle 17.6: Referenz zur CSS-Eigenschaft »background-position« (Fortsetzung)

background-repeat

Bedeutung:	Wiederholung des Hintergrundbilds (Wallpaper-Effekt)
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	einer der folgenden Werte: **repeat** (Wallpapper über die gesamte Fläche) | repeat-x (Wallpaper nur »erste Zeile«) | repeat-y (Wallpaper nur »erste Spalte«) | no-repeat (kein Wallpaper-Effekt, nur einmalig anzeigen) | inherit (wie Elternelement)

Tabelle 17.7: Referenz zur CSS-Eigenschaft »background-repeat«

background

Bedeutung:	Hintergrund (zusammenfassend)
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Leerzeichengetrennte, summarische Angabe zu den Eigenschaften background-color, background-image, background-repeat, background-attachment und background-position, oder der folgende Wert: inherit

Tabelle 17.8: Referenz zur CSS-Eigenschaft »background«

border

Bedeutung:	Rahmen (zusammenfassend)
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Leerzeichengetrennte, summarische Angabe zu einem, zweien oder allen dreien der Eigenschaftentypen border-color, border-style und border-width

Tabelle 17.9: Referenz zur CSS-Eigenschaft »border«

border-bottom

Bedeutung:	Rahmen unten (zusammenfassend)
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Leerzeichengetrennte, summarische Angabe zu einem, zweien oder allen dreien der Eigenschaftentypen border-color, border-style und border-width

Tabelle 17.10: Referenz zur CSS-Eigenschaft »border-bottom«

border-bottom-color

Bedeutung:	Farbe Rahmen unten
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Farbangabe oder einer der folgenden Werte: transparent (farblos/durchsichtig) | inherit (wie Elternelement)

Tabelle 17.11: Referenz zur CSS-Eigenschaft »border-bottom-color«

border-bottom-style

Bedeutung:	Typ Rahmen unten
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	a (), b (), c(), numerischer Wert mit Maßeinheit (auch Prozentualwert).

Tabelle 17.12: Referenz zur CSS-Eigenschaft »border-bottom-style«

border-bottom-width

Bedeutung:	Stärke Rahmen unten
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Numerischer Wert mit Maßeinheit oder einer der folgenden Werte: thin (dünn) | **medium** (mittelstark) | thick (dick) | inherit (wie Elternelement)

Tabelle 17.13: Referenz zur CSS-Eigenschaft »border-bottom-width«

border-collapse

Bedeutung:	Rahmenzusammenfall bei Tabellenzellen
Typ:	Visuell
Vererbung:	Ja
Elemente:	table
Werte:	Einer der folgenden Werte: collapse (zusammenfallende Rahmen) | **separate** (auseinander liegende Rahmen) | inherit (wie Elternelement)

Tabelle 17.14: Referenz zur CSS-Eigenschaft »border-collapse«

border-color

Bedeutung:	Rahmenfarbe (zusammenfassend für alle vier Seiten)
Typ:	Visuell
Vererbung:	Nein
Elemente:	x,y,z
Werte:	Farbangabe oder einer der folgenden Werte: transparent (farblos/durchsichtig) | inherit (wie Elternelement)

Tabelle 17.15: Referenz zur CSS-Eigenschaft »border-color«

border-left

Bedeutung:	Rahmen links (zusammenfassend)
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Leerzeichengetrennte, summarische Angabe zu einem, zweien oder allen dreien der Eigenschaftentypen border-color, border-style und border-width

Tabelle 17.16: Referenz zur CSS-Eigenschaft »border-left«

border-left-color

Bedeutung:	Farbe Rahmen links
Typ:	Visuell
Vererbung:	Nein

Tabelle 17.17: Referenz zur CSS-Eigenschaft »border-left-color«

Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Farbangabe oder einer der folgenden Werte: transparent (farblos/durchsichtig) | inherit (wie Elternelement)

Tabelle 17.17: Referenz zur CSS-Eigenschaft »border-left-color« (Fortsetzung)

border-left-style

Bedeutung:	Typ Rahmen links
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Einer der folgenden Werte: **none** oder hidden (kein Rahmen) | dotted (gepünktelter Rahmen) | dashed (gestrichelter Rahmen) | solid (durchgezogener Rahmen) | double (doppelter, durchgezogener Rahmen) | einer der 3D-Effekte groove, ridge, inset oder outset | inherit (wie Elternelement)

Tabelle 17.18: Referenz zur CSS-Eigenschaft »border-left-style«

border-left-width

Bedeutung:	Stärke Rahmen links
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Numerischer Wert mit Maßeinheit oder einer der folgenden Werte: thin (dünn) | **medium** (mittelstark) | thick (dick) | inherit (wie Elternelement)

Tabelle 17.19: Referenz zur CSS-Eigenschaft »border-left-width«

border-right

Bedeutung:	Rahmen rechts (zusammenfassend)
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Leerzeichengetrennte, summarische Angabe zu einem, zweien oder allen dreien der Eigenschaftentypen border-color, border-style und border-width

Tabelle 17.20: Referenz zur CSS-Eigenschaft »border- right«

border- right-color

Bedeutung:	Farbe Rahmen rechts
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Farbangabe oder einer der folgenden Werte: transparent (farblos/durchsichtig) | inherit (wie Elternelement)

Tabelle 17.21: Referenz zur CSS-Eigenschaft »border-right-color«

border-right-style

Bedeutung:	Typ Rahmen rechts
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Einer der folgenden Werte: **none** oder hidden (kein Rahmen) | dotted (gepünktelter Rahmen) | dashed (gestrichelter Rahmen) | solid (durchgezogener Rahmen) | double (doppelter, durchgezogener Rahmen) | einer der 3D-Effekte groove, ridge, inset oder outset | inherit (wie Elternelement)

Tabelle 17.22: Referenz zur CSS-Eigenschaft »border-bottom-style«

border-right-width

Bedeutung:	Stärke Rahmen rechts
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Numerischer Wert mit Maßeinheit oder einer der folgenden Werte: thin (dünn) | **medium** (mittelstark) | thick (dick) | inherit (wie Elternelement)

Tabelle 17.23: Referenz zur CSS-Eigenschaft »border-right-width«

border-spacing

Bedeutung:	Rahmenabstand im Tabellengitternetz
Typ:	Visuell
Vererbung:	Ja

Tabelle 17.24: Referenz zur CSS-Eigenschaft »border-spacing«

Elemente:	table
Werte:	Numerischer Wert mit Maßeinheit oder der folgende Wert: inherit (wie Elternelement)

Tabelle 17.24: Referenz zur CSS-Eigenschaft »border-spacing« (Fortsetzung)

border-style

Bedeutung:	Rahmentyp (zusammenfassend für alle vier Seiten)
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Einer der folgenden Werte: **none** oder hidden (kein Rahmen) | dotted (gepünktelter Rahmen) | dashed (gestrichelter Rahmen) | solid (durchgezogener Rahmen) | double (doppelter, durchgezogener Rahmen) | einer der 3D-Effekte groove, ridge, inset oder outset | inherit (wie Elternelement)

Tabelle 17.25: Referenz zur CSS-Eigenschaft »border-style«

border-top

Bedeutung:	Rahmen oben (zusammenfassend)
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Leerzeichengetrennte, summarische Angabe zu einem, zweien oder allen dreien der Eigenschaftentypen border-color, border-style und border-width

Tabelle 17.26: Referenz zur CSS-Eigenschaft »border-top«

border-top-color

Bedeutung:	Farbe Rahmen oben
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Farbangabe oder einer der folgenden Werte: transparent (farblos/durchsichtig) | inherit (wie Elternelement)

Tabelle 17.27: Referenz zur CSS-Eigenschaft »border-top-color«

border-top-style

Bedeutung:	Typ Rahmen oben
Typ:	Visuell
Vererbung:	Nein
Elemente:	x,y,z
Werte:	Einer der folgenden Werte: **none** oder hidden (kein Rahmen) | dotted (gepünktelter Rahmen) | dashed (gestrichelter Rahmen) | solid (durchgezogener Rahmen) | double (doppelter, durchgezogener Rahmen) | einer der 3D-Effekte groove, ridge, inset oder outset | inherit (wie Elternelement)

Tabelle 17.28: Referenz zur CSS-Eigenschaft »border-top-style«

border-top-width

Bedeutung:	Stärke Rahmen oben
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Numerischer Wert mit Maßeinheit oder einer der folgenden Werte: thin (dünn) | **medium** (mittelstark) | thick (dick) | inherit (wie Elternelement)

Tabelle 17.29: Referenz zur CSS-Eigenschaft »border-top-width«

border-width

Bedeutung:	Rahmenstärke (zusammenfassend für alle vier Seiten)
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Numerischer Wert mit Maßeinheit oder einer der folgenden Werte: thin (dünn) | **medium** (mittelstark) | thick (dick) | inherit (wie Elternelement)

Tabelle 17.30: Referenz zur CSS-Eigenschaft »border-width«

bottom

Bedeutung:	Startposition von unten
Typ:	Visuell
Vererbung:	Nein

Tabelle 17.31: Referenz zur CSS-Eigenschaft »bottom«

Elemente:	Alle, die eine Angabe zur Eigenschaft position enthalten
Werte:	Numerischer Wert mit Maßeinheit (auch Prozentualwert) oder einer der folgenden Werte: auto (automatisch berechnen) | inherit (wie Elternelement)

Tabelle 17.31: Referenz zur CSS-Eigenschaft »bottom« (Fortsetzung)

caption-side

Bedeutung:	Position der Tabellenbeschriftung
Typ:	Visuell
Vererbung:	Ja
Elemente:	caption
Werte:	Einer der folgenden Werte: **top** (oben, Tabellenüberschrift) | bottom (unten, Tabellenunterschrift) | inherit (wie Elternelement)

Tabelle 17.32: Referenz zur CSS-Eigenschaft »caption-side«

clear

Bedeutung:	Fortsetzung bei Textumfluss
Typ:	Visuell
Vererbung:	Nein
Elemente:	Blockelemente
Werte:	Einer der folgenden Werte: **none** (keine erzwungene Fortsetzung unterhalb) | left (erzwingt bei aktivem float:left Fortsetzung unterhalb) | right (erzwingt bei aktivem float:right Fortsetzung unterhalb) | both (erzwingt bei aktivem float:left und aktivem float:right die Fortsetzung unterhalb) | inherit (wie Elternelement)

Tabelle 17.33: Referenz zur CSS-Eigenschaft »clear«

clip

Bedeutung:	Sichtbarer Ausschnitt
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, die position:absolute enthalten
Werte:	rect(<oben>, <rechts>, <unten>, <links>), wobei <oben>, <rechts>, <unten> und <links> je ein numerischer Wert mit Maßeinheit ist oder einer der folgenden Werte: **auto** (automatisch berechnen) | inherit (wie Elternelement)

Tabelle 17.34: Referenz zur CSS-Eigenschaft »clip«

color

Bedeutung:	Textfarbe
Typ:	Visuell
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Farbangabe oder der folgende Wert: inherit (wie Elternelement)

Tabelle 17.35: Referenz zur CSS-Eigenschaft »color«

content

Bedeutung:	Automatisch hinzugefügter Inhalt
Typ:	Alle
Vererbung:	Nein
Elemente:	Pseudo-Elemente mit:before oder:after
Werte:	Statischer Text | url(URI) | counter() oder einer der folgenden Werte: normal (kein automatisch hinzugefügter Inhalt vor dem Element) | inherit (wie Elternelement)

Tabelle 17.36: Referenz zur CSS-Eigenschaft »content«

counter-increment

Bedeutung:	Nummerierungszähler erhöhen
Typ:	Alle
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Name einer Zählervariablen, die mit der Funktion counter() definiert wurde

Tabelle 17.37: Referenz zur CSS-Eigenschaft »counter-increment«

counter-reset

Bedeutung:	Nummerierungszähler zurücksetzen
Typ:	Alle
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Name einer Zählervariablen, die mit der Funktion counter() definiert wurde

Tabelle 17.38: Referenz zur CSS-Eigenschaft »counter-reset«

cue

Bedeutung:	Erkennungssignal vor/nach einem Element
Typ:	Aural
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	url(URI) oder einer der folgenden Werte: **none** (kein Erkennungssignal) | inherit (wie Elternelement)

Tabelle 17.39: Referenz zur CSS-Eigenschaft »cue«

cue-after

Bedeutung:	Erkennungssignal nach einem Element
Typ:	url(URI) oder einer der folgenden Werte: **none** (kein Erkennungssignal) | inherit (wie Elternelement)
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	a (), b (), c(), numerischer Wert mit Maßeinheit (auch Prozentualwert).

Tabelle 17.40: Referenz zur CSS-Eigenschaft »cue-after«

cue-before

Bedeutung:	Erkennungssignal vor einem Element
Typ:	Aural
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	url(URI) für Sound-Datei oder einer der folgenden Werte: **none** (kein Erkennungssignal) | inherit (wie Elternelement)

Tabelle 17.41: Referenz zur CSS-Eigenschaft »cue-before«

cursor

Bedeutung:	Mauszeiger
Typ:	Visuell, interaktiv
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist

Tabelle 17.42: Referenz zur CSS-Eigenschaft »cursor«

Werte:	url(URI) für Grafikdatei oder einer der folgenden Werte: **auto** (Mauszeiger wie für das Element sonst üblich) | crosshair (Fadenkreuz) | default (Standard-Mauszeiger) | pointer (Zeiger) | move (Kreuz, das Bewegungsfähigkeit des Elements signalisiert) | e-resize (Pfeil nach rechts) | ne-resize (Pfeil nach rechts oben) | nw-resize (Pfeil nach links oben) | n-resize (Pfeil nach oben) | se-resize (Pfeil nach rechts unten) | sw-resize (Pfeil nach links unten) | s-resize (Pfeil nach unten) | w-resize (Pfeil nach links) | text (Mauszeiger wie über normalem Text) | wait (Mauszeiger wie bei »bitte warten«) | help (Mauszeiger für Hilfesymbol) | progress (Mauszeiger für Fortschrittssignalisierung) | inherit (wie Elternelement)

Tabelle 17.42: Referenz zur CSS-Eigenschaft »cursor« (Fortsetzung)

direction

Bedeutung:	Schriftrichtung
Typ:	Visuell
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Einer der folgenden Werte: **ltr** (links nach rechts) | rtl (rechts nach links) | inherit (wie Elternelement)

Tabelle 17.43: Referenz zur CSS-Eigenschaft »direction«

display

Bedeutung:	Art der Darstellung
Typ:	Alle
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Einer der folgenden Werte: **inline** (wie ein Inline-Element) | block (wie ein Blockelement) | list-item (wie ein li-Element) | marker (in Verbindung mit der Eigenschaft content) | table (Element entspricht einem table-Element) | inline-table (Element entspricht einem table-Element, das mit display:inline definiert ist) | table-row (Element entspricht einem tr-Element) | table-row-group (Element entspricht einem tbody-Element) | table-header-group (Element entspricht einem thead-Element) | table-footer-group (Element entspricht einem tfoot-Element) | table-column (Element entspricht einem col-Element) | table-column-group (Element entspricht einem colgroup-Element) | table-cell (Element entspricht einem td- oder th-Element) | none (Element wird nicht angezeigt) | inherit (wie Elternelement)

Tabelle 17.44: Referenz zur CSS-Eigenschaft »display«

elevation

Bedeutung:	Raum-Effekt oben/unten
Typ:	Aural
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Zahl -90 bis 90, gefolgt von der Maßeinheit deg, oder einer der folgenden Werte: below (von unten) | **level** (von der Mitte) | above (von oben) | higher (von weiter oben) | lower (von weiter unten) | inherit (wie Elternelement)

Tabelle 17.45: Referenz zur CSS-Eigenschaft »elevation«

empty-cells

Bedeutung:	Anzeige/Nichtanzeige leerer Zellen
Typ:	Visuell
Vererbung:	Ja
Elemente:	Table
Werte:	Einer der folgenden Werte: **show** (leere Zellen als Zellen ohne Inhalt darstellen) | hide (leere Zellen als nicht vorhandene Zellen darstellen) | inherit (wie Elternelement)

Tabelle 17.46: Referenz zur CSS-Eigenschaft »empty-cells«

float

Bedeutung:	Textumfluss
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Einer der folgenden Werte: left (nachfolgende Elemente fließen rechts um das Element) | right (nachfolgende Elemente fließen links um das rechts ausgerichtete Element) | none (normaler Textfluss) | inherit (wie Elternelement)

Tabelle 17.47: Referenz zur CSS-Eigenschaft »float«

font

Bedeutung:	Schrift (zusammenfassend)
Typ:	Visuell
Vererbung:	Ja

Tabelle 17.48: Referenz zur CSS-Eigenschaft »font«

Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Summarische Angabe aus Werten zu font-family, font-size, font-style, font-variant, font-weight und line-height

Tabelle 17.48: Referenz zur CSS-Eigenschaft »font« (Fortsetzung)

font-family

Bedeutung:	Schriftart
Typ:	Visuell
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Ein oder mehrere (kommagetrennte) Namen von Schriftarten, oder eine der folgenden generischen Angaben: serif (Serifenschrift) | sans-serif (serifenlose Schrift) | cursive (Schreibschrift) | fantasy (ungewöhnliche Schrift) | monospace (dicktengleiche Schrift) | inherit (wie Elternelement)

Tabelle 17.49: Referenz zur CSS-Eigenschaft »font-family«

font-size

Bedeutung:	Schriftgröße
Typ:	Visuell
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Numerische Angabe mit Maßeinheit (auch Prozentualwert) oder der folgende Wert: inherit (wie Elternelement)

Tabelle 17.50: Referenz zur CSS-Eigenschaft »font-size«

font-style

Bedeutung:	Schriftstil
Typ:	Visuell
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Einer der folgenden Werte: **normal** (Normalschrift) | italic (kursiv) | oblique (kursiv) | inherit (wie Elternelement)

Tabelle 17.51: Referenz zur CSS-Eigenschaft »font-style«

font-variant

Bedeutung:	Schriftvariante
Typ:	Visuell
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Einer der folgenden Werte: **normal** (Normalschrift) | small-caps (Kapitälchen) | inherit (wie Elternelement)

Tabelle 17.52: Referenz zur CSS-Eigenschaft »font-variant«

font-weight

Bedeutung:	Schriftgewicht
Typ:	Visuell
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Einer der folgenden Werte: **normal** (Normalschrift) | bold (fett) | bolder (extra fett) | lighter (dünner) | inherit (wie Elternelement) oder einer der Werte 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 für extra dünn (100) bis extra dick (900)

Tabelle 17.53: Referenz zur CSS-Eigenschaft »font-weight«

height

Bedeutung:	Elementhöhe
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle Blockelemente, img und object
Werte:	Numerischer Wert mit Maßeinheit (auch Prozentualwert) oder einer der folgenden Werte: auto (automatisch berechnen) | inherit (wie Elternelement)

Tabelle 17.54: Referenz zur CSS-Eigenschaft »height«

left

Bedeutung:	Startposition von links
Typ:	Visuell
Vererbung:	Nein

Tabelle 17.55: Referenz zur CSS-Eigenschaft »left«

Elemente:	Alle, die eine Angabe zur Eigenschaft position enthalten
Werte:	Numerischer Wert mit Maßeinheit (auch Prozentualwert), oder einer der folgenden Werte: auto (automatisch berechnen) | inherit (wie Elternelement)

Tabelle 17.55: Referenz zur CSS-Eigenschaft »left« (Fortsetzung)

letter-spacing

Bedeutung:	Zeichenabstand
Typ:	Visuell
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Numerischer Wert mit Maßeinheit oder einer der folgenden Werte: **normal** (normaler Zeichenabstand) | inherit (wie Elternelement)

Tabelle 17.56: Referenz zur CSS-Eigenschaft »letter-spacing«

line-height

Bedeutung:	Zeilenhöhe
Typ:	Visuell
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Numerischer Wert mit Maßeinheit (auch Prozentualwert) oder einer der folgenden Werte: **normal** (normale Zeilenhöhe) | inherit (wie Elternelement)

Tabelle 17.57: Referenz zur CSS-Eigenschaft »line-height«

list-style

Bedeutung:	Darstellungstyp der Liste
Typ:	Visuell
Vererbung:	Ja
Elemente:	ol, ul, abhängig von der Wertzuweisung
Werte:	Summarische Angabe aus den Eigenschaften list-style-image, list-style-position und list-style-type

Tabelle 17.58: Referenz zur CSS-Eigenschaft »list-style«

list-style-image

Bedeutung:	Bullet-Symbolgrafik
Typ:	Visuell
Vererbung:	Ja
Elemente:	ul
Werte:	url(URI) oder einer der folgenden Werte: none (kein Bulletzeichen) | inherit (wie Elternelement)

Tabelle 17.59: Referenz zur CSS-Eigenschaft »list-style-image«

list-style-position

Bedeutung:	Position von Bullet-Symbolen oder Nummerierungszahlen
Typ:	Visuell
Vererbung:	Ja
Elemente:	ol, ul
Werte:	Einer der folgenden Werte: inside (Bullet- oder Nummerierungszeichen in der Liste) | **outside** (Bullet- oder Nummerierungszeichen nach außen gestellt) inherit (wie Elternelement)

Tabelle 17.60: Referenz zur CSS-Eigenschaft »list-style-position«

list-style-type

Bedeutung:	Gibt ... an.
Typ:	Visuell
Vererbung:	Ja
Elemente:	ol, ul, abhängig von der Wertzuweisung
Werte:	Für ol-Listen einer der folgenden Werte: decimal (Nummerierung mit arabischen Ziffern) | decimal-leading-zero (wie decimal, nur mit führenden Nullen) | lower-roman (Nummerierung mit kleinen römischen Ziffern) | upper-roman (Nummerierung mit großen römischen Ziffern) | georgian (Nummerierung nach traditionell georgischer Art) | armenian (Nummerierung nach traditionell armenischer Art) | lower-alpha (Nummerierung mit Kleinbuchstaben) | upper-alpha (Nummerierung mit Großbuchstaben) | lower-greek (Nummerierung mit griechischen Kleinbuchstaben) | upper-greek (Nummerierung mit griechischen Großbuchstaben) | none (keine Nummerierung) | inherit (wie Elternelement). Für ul-Listen einer der folgenden Werte: disc (Punkt), circle (Kreis), square (Quadrat) | none (kein Aufzählungszeichen) | inherit (wie Elternelement)

Tabelle 17.61: Referenz zur CSS-Eigenschaft »list-style-type«

margin

Bedeutung:	Abstand zu Nachbar- bzw. Elternelementen (zusammenfassend)
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle außer Elementen innerhalb von table
Werte:	Ein bis vier numerische Werte mit Maßeinheit (auch Prozentualwert) oder folgende Werte: auto (automatisch berechnen) | inherit (wie Elternelement); ein Wert bedeutet alle Abstände, zwei Werte bedeuten Abstände oben und unten, drei Werte bedeuten erstens oben, zweitens links/rechts und drittens unten und vier Werte bedeuten erstens oben, zweitens rechts, drittens unten, viertens links

Tabelle 17.62: Referenz zur CSS-Eigenschaft »margin«

margin-bottom

Bedeutung:	Abstand zu Nachbar- bzw. Elternelementen unten
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle außer Elementen innerhalb von table
Werte:	Numerischer Wert mit Maßeinheit (auch Prozentualwert), oder einer der folgenden Werte: auto (automatisch berechnen) | inherit (wie Elternelement)

Tabelle 17.63: Referenz zur CSS-Eigenschaft »margin-right«

margin-left

Bedeutung:	Abstand zu Nachbar- bzw. Elternelementen links
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle außer Elementen innerhalb von table
Werte:	Numerischer Wert mit Maßeinheit (auch Prozentualwert) oder einer der folgenden Werte: auto (automatisch berechnen) | inherit (wie Elternelement)

Tabelle 17.64: Referenz zur CSS-Eigenschaft »margin-right«

margin-right

Bedeutung:	Abstand zu Nachbar- bzw. Elternelementen rechts
Typ:	Visuell

Tabelle 17.65: Referenz zur CSS-Eigenschaft »margin-right«

Vererbung:	Nein
Elemente:	Alle außer Elementen innerhalb von table
Werte:	Numerischer Wert mit Maßeinheit (auch Prozentualwert) oder einer der folgenden Werte: auto (automatisch berechnen) | inherit (wie Elternelement)

Tabelle 17.65: Referenz zur CSS-Eigenschaft »margin-right« (Fortsetzung)

margin-top

Bedeutung:	Abstand zu Nachbar- bzw. Elternelementen oben
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle außer Elementen innerhalb von table
Werte:	Numerischer Wert mit Maßeinheit (auch Prozentualwert) oder einer der folgenden Werte: auto (automatisch berechnen) | inherit (wie Elternelement)

Tabelle 17.66: Referenz zur CSS-Eigenschaft »margin-top«

max-height

Bedeutung:	Maximalhöhe eines Elements
Typ:	Visuell
Vererbung:	Nein
Elemente:	Blockelemente, Tabellenelemente, img, object
Werte:	Nnumerischer Wert mit Maßeinheit (auch Prozentualwert) oder einer der folgenden Werte: **none** (keine Höhenbegrenzung) | inherit (wie Elternelement)

Tabelle 17.67: Referenz zur CSS-Eigenschaft »max-height«

max-width

Bedeutung:	Maximalbreite eines Elements
Typ:	Visuell
Vererbung:	Nein
Elemente:	Blockelemente, Tabellenelemente, img, object
Werte:	Numerischer Wert mit Maßeinheit (auch Prozentualwert) oder einer der folgenden Werte: **none** (keine Breitenbegrenzung) | inherit (wie Elternelement)

Tabelle 17.68: Referenz zur CSS-Eigenschaft »max-width«

min-height

Bedeutung:	Mindesthöhe eines Elements
Typ:	Visuell
Vererbung:	Nein
Elemente:	Blockelemente, Tabellenelemente, img, object
Werte:	Numerischer Wert mit Maßeinheit (auch Prozentualwert) oder folgender Wert: inherit (wie Elternelement)

Tabelle 17.69: Referenz zur CSS-Eigenschaft »min-height«

min-width

Bedeutung:	Mindestbreite eines Elements
Typ:	Visuell
Vererbung:	Nein
Elemente:	Blockelemente, Tabellenelemente, img, object
Werte:	Numerischer Wert mit Maßeinheit (auch Prozentualwert) oder folgender Wert: inherit (wie Elternelement)

Tabelle 17.70: Referenz zur CSS-Eigenschaft »min-width«

orphans

Bedeutung:	Alleinstehende Zeilen am Seitenende
Typ:	Visuell, seitenorientiert
Vererbung:	Ja
Elemente:	Alle Blockelemente
Werte:	Numerischer Wert (Anzahl zu einem Block gehörender Zeilen, die am Seitenende vor einem Seitenumbruch stehen müssen) | der folgende Wert: inherit (wie Elternelement)

Tabelle 17.71: Referenz zur CSS-Eigenschaft »orphans«

outline

Bedeutung:	Fokus-signalisierender Elementrahmen
Typ:	Visuell, interaktiv
Vererbung:	Nein

Tabelle 17.72: Referenz zur CSS-Eigenschaft »outline«

Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Leerzeichengetrennte, summarische Angabe zu einem, zweien oder allen dreien der Eigenschaftentypen outline-color, outline-style und outline-width

Tabelle 17.72: Referenz zur CSS-Eigenschaft »outline« (Fortsetzung)

outline-color

Bedeutung:	Farbe für fokus-signalisierenden Elementrahmen
Typ:	Visuell, interaktiv
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Farbangabe oder einer der folgenden Werte: **invert** (invertierte Darstellung) | inherit (wie Elternelement)

Tabelle 17.73: Referenz zur CSS-Eigenschaft »outline-color«

outline-style

Bedeutung:	Typ des fokus-signalisierenden Elementrahmens
Typ:	Visuell, interaktiv
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Einer der folgenden Werte: **none** oder hidden (kein Rahmen) | dotted (gepünktelter Rahmen) | dashed (gestrichelter Rahmen) | solid (durchgezogener Rahmen) | double (doppelter, durchgezogener Rahmen) | einer der 3D-Effekte groove, ridge, inset oder outset | inherit (wie Elternelement)

Tabelle 17.74: Referenz zur CSS-Eigenschaft »outline-style«

outline-width

Bedeutung:	Dicke des fokus-signalisierenden Elementrahmens
Typ:	Visuell, interaktiv
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Numerischer Wert mit Maßeinheit oder einer der folgenden Werte: thin (dünn) | **medium** (mittelstark) | thick (dick) | inherit (wie Elternelement)

Tabelle 17.75: Referenz zur CSS-Eigenschaft »outline-width«

overflow

Bedeutung:	Elementbereich mit übergroßem Inhalt
Typ:	Visuell
Vererbung:	Nein
Elemente:	Blockelemente, img, object, Tabellenzellen
Werte:	Einer der folgenden Werte: visible (Element ausdehnen, so dass der gesamte Inhalt sichtbar ist) | hidden (angegebene Elementgröße gelten lassen und übergroße Inhalte abschneiden) | scroll (Scrollleisten an den Elementgrenzen anbieten, um im übergroßen Inhalt scrollen zu können) | auto (Browser die Entscheidung überlassen) | inherit (wie Elternelement)

Tabelle 17.76: Referenz zur CSS-Eigenschaft »overflow«

padding

Bedeutung:	Innenabstand von Elementinhalt zu Elementgrenze (zusammenfassend)
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Ein bis vier numerische Werte mit Maßeinheit (auch Prozentualwert) oder folgender Wert: inherit (wie Elternelement); ein Wert bedeutet alle Innenabstände, zwei Werte bedeuten Innenabstände oben und unten, drei Werte bedeuten erstens oben, zweitens links/rechts und drittens unten und vier Werte bedeuten erstens oben, zweitens rechts, drittens unten, viertens links

Tabelle 17.77: Referenz zur CSS-Eigenschaft »padding«

padding-bottom

Bedeutung:	Innenabstand von Elementinhalt zu Elementgrenze unten
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Numerischer Wert mit Maßeinheit (auch Prozentualwert) oder folgender Wert: inherit (wie Elternelement)

Tabelle 17.78: Referenz zur CSS-Eigenschaft »padding-bottom«

padding-left

Bedeutung:	Innenabstand von Elementinhalt zu Elementgrenze links
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Numerischer Wert mit Maßeinheit (auch Prozentualwert) oder folgender Wert: inherit (wie Elternelement)

Tabelle 17.79: Referenz zur CSS-Eigenschaft »padding-left«

padding-right

Bedeutung:	Innenabstand von Elementinhalt zu Elementgrenze rechts
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Numerischer Wert mit Maßeinheit (auch Prozentualwert) oder folgender Wert: inherit (wie Elternelement)

Tabelle 17.80: Referenz zur CSS-Eigenschaft »padding-right«

padding-top

Bedeutung:	Innenabstand von Elementinhalt zu Elementgrenze oben
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Numerischer Wert mit Maßeinheit (auch Prozentualwert) oder folgender Wert: inherit (wie Elternelement)

Tabelle 17.81: Referenz zur CSS-Eigenschaft »padding-top«

page-break-after

Bedeutung:	Seitenumbruch nach einem Element
Typ:	Visuell, seitenorientiert
Vererbung:	Nein
Elemente:	Alle Blockelemente

Tabelle 17.82: Referenz zur CSS-Eigenschaft »page-break-after«

Werte:	Einer der folgenden Werte: auto (Seitenumbruch wenn erforderlich) | always (auf jeden Fall Seitenumbruch nach Element einfügen) | avoid (auf keinen Fall Seitenumbruch nach Element) | left (ein oder zwei Seitenumbrüche nach Element einfügen, so dass nachfolgende Inhalte auf einer linken Seite beginnen) | right (ein oder zwei Seitenumbrüche nach Element einfügen, so dass nachfolgende Inhalte auf einer rechten Seite beginnen) | inherit (wie Elternelement)

Tabelle 17.82: Referenz zur CSS-Eigenschaft »page-break-after« (Fortsetzung)

page-break-before

Bedeutung:	Seitenumbruch vor einem Element
Typ:	Visuell, seitenorientiert
Vererbung:	Nein
Elemente:	Alle Blockelemente
Werte:	Einer der folgenden Werte: auto (Seitenumbruch wenn erforderlich) | always (auf jeden Fall Seitenumbruch vor Element einfügen) | avoid (auf keinen Fall Seitenumbruch vor Element) | left (ein oder zwei Seitenumbrüche vor Element einfügen, so dass Element auf einer linken Seite beginnt) | right (ein oder zwei Seitenumbrüche vor Element einfügen, so dass Element auf einer rechten Seite beginnt) | inherit (wie Elternelement)

Tabelle 17.83: Referenz zur CSS-Eigenschaft »page-break-before«

page-break-inside

Bedeutung:	Seitenumbruch innerhalb eines Elements
Typ:	Visuell, seitenorientiert
Vererbung:	Ja
Elemente:	Alle Blockelemente
Werte:	Einer der folgenden Werte: auto (Seitenumbruch wenn erforderlich) | avoid (auf keinen Fall Seitenumbruch innerhalb Element einfügen) | inherit (wie Elternelement)

Tabelle 17.84: Referenz zur CSS-Eigenschaft »page-break-inside«

pause

Bedeutung:	Sprechpause vor/nach einem Element
Typ:	Aural
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Numerischer Wert mit Maßeinheit s (Sekunden) oder ms (Millisekunden), auch Prozentualwert, oder der folgende Wert: inherit (wie Elternelement)

Tabelle 17.85: Referenz zur CSS-Eigenschaft »pause«

pause-after

Bedeutung:	Sprachpause nach einem Element
Typ:	Aural
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Numerischer Wert mit Maßeinheit s (Sekunden) oder ms (Millisekunden), auch Prozentualwert, oder der folgende Wert: inherit (wie Elternelement)

Tabelle 17.86: Referenz zur CSS-Eigenschaft »pasue-after«

pause-before

Bedeutung:	Sprechpause vor einem Element
Typ:	Aural
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Numerischer Wert mit Maßeinheit s (Sekunden) oder ms (Millisekunden), auch Prozentualwert, oder der folgende Wert: inherit (wie Elternelement)

Tabelle 17.87: Referenz zur CSS-Eigenschaft »pause-before«

pitch-range

Bedeutung:	Sprechweise
Typ:	Aural
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Wert zwischen 0 (monoton) und 100 (moduliert) oder der folgende Wert: inherit (wie Elternelement)

Tabelle 17.88: Referenz zur CSS-Eigenschaft »pitch-range«

pitch

Bedeutung:	Sprechhöhe
Typ:	Aural
Vererbung:	Ja

Tabelle 17.89: Referenz zur CSS-Eigenschaft »pitch«

Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Numerischer Wert mit Maßeinheit Hz (Hertz) oder kHz (Kilohertz) oder einer der folgenden Werte: low (tief) | x-low (sehr tief) | **medium** (mittel) | high (hoch) | x-high (sehr hoch) | inherit (wie Elternelement)

Tabelle 17.89: Referenz zur CSS-Eigenschaft »pitch« (Fortsetzung)

play-during

Bedeutung:	Hintergrundsound während Sprachausgabe
Typ:	Aural
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	url(URI) für Sound-Datei, auf Wunsch kombinierbar mit einem der folgenden Werte: mix (vermischt mit dem Hintergrundsound des Elternelements) | repeat (Sounddatei wird wiederholt, solange die Ausgabe des Elementtextes dauert) oder einer der folgenden Werte: **auto** (falls das Elternelement einen Hintergrundsound hat, wird dieser weiter abgespielt) | none (falls das Elternelement einen Hintergrundsound hat, wird dieser unterbrochen) | inherit (wie Elternelement)

Tabelle 17.90: Referenz zur CSS-Eigenschaft »play during«

position

Bedeutung:	Positionierung eines Elements
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Einer der folgenden Werte: static (normale Positionierung) | relative (relativ zur normalen Positionierung) | absolute (absolute Positionierung, gemessen am Elternelement) | fixed (fixe Positionierung) | inherit (wie Elternelement)

Tabelle 17.91: Referenz zur CSS-Eigenschaft »position«

quotes

Bedeutung:	Anführungszeichen um ein Element
Typ:	Visuell
Vererbung:	Ja

Tabelle 17.92: Referenz zur CSS-Eigenschaft »quotes«

Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Zwei leerzeichengetrennte Zeichen für die gewünschten Anführungszeichen oder einer der folgenden Werte: none (keine Anführungszeichen) | inherit (wie Elternelement)

Tabelle 17.92: Referenz zur CSS-Eigenschaft »quotes« (Fortsetzung)

richness

Bedeutung:	Stimmgewalt
Typ:	Aural
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Wert zwischen 0 (schwach) und 100 (gewaltig) oder folgender Wert: inherit (wie Elternelement)

Tabelle 17.93: Referenz zur CSS-Eigenschaft »richness«

right

Bedeutung:	Startposition von rechts
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, die eine Angabe zur Eigenschaft position enthalten
Werte:	Numerischer Wert mit Maßeinheit (auch Prozentualwert) oder einer der folgenden Werte: auto (automatisch berechnen) | inherit (wie Elternelement)

Tabelle 17.94: Referenz zur CSS-Eigenschaft »right«

speak-header

Bedeutung:	Kopfzellenwiederholung bei Sprachausgabe
Typ:	Visuell, aural, printorientiert
Vererbung:	Ja
Elemente:	Th
Werte:	einer der folgenden Werte: once (nur einmal wiederholen) | always (Kopfzelle vor jeder zugeordneten Datenzelle wiederholen) | inherit (wie Elternelement)

Tabelle 17.95: Referenz zur CSS-Eigenschaft »speak-header«

speak-numeral

Bedeutung:	Konvention beim Aussprechen von Zahlen
Typ:	Aural
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Einer der folgenden Werte: digits (Zahl wird Ziffer für Ziffer ausgesprochen) | **continuous** (Zahl wird normal ausgesprochen) | inherit (wie Elternelement)

Tabelle 17.96: Referenz zur CSS-Eigenschaft »speak-numeral«

speak-punctuation

Bedeutung:	Konvention beim Aussprechen von Satzzeichen
Typ:	Aural
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Einer der folgenden Werte: code (Satzzeichen werden explizit genannt) | **none** (Satzzeichen werden nicht explizit genannt) | inherit (wie Elternelement)

Tabelle 17.97: Referenz zur CSS-Eigenschaft »speak-puctuation«

speak

Bedeutung:	Art der Aussprache eines Elements
Typ:	Aural
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Einer der folgenden Werte: **normal** (normale Aussprache) | none (Element wird nicht ausgesprochen) | spell-out (buchstabierende Aussprache)

Tabelle 17.98: Referenz zur CSS-Eigenschaft »speak«

speech-rate

Bedeutung:	Sprechgeschwindigkeit
Typ:	Aural
Vererbung:	Ja

Tabelle 17.99: Referenz zur CSS-Eigenschaft »speech-rate«

Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Numerischer Wert ohne Maßeinheit (Mittel ist 180-200) oder einer der folgenden Werte: x-slow (sehr langsam) | slow (langsam) slower (langsamer als normal) | **medium** (mittel) | faster (schneller als normal) | fast (schnell) | x-fast (sehr schnell) | inherit (wie Elternelement)

Tabelle 17.99: Referenz zur CSS-Eigenschaft »speech-rate« (Fortsetzung)

stress

Bedeutung:	Sprechhast
Typ:	Aural
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Numerischer Wert zwischen 0 (behäbig) und 100 (extrem hastig), der folgende Wert: inherit (wie Elternelement)

Tabelle 17.100: Referenz zur CSS-Eigenschaft »stress«

table-layout

Bedeutung:	Fixe/variable Spaltenbreiten
Typ:	Visuell
Vererbung:	Nein
Elemente:	table
Werte:	Einer der folgenden Werte: **auto** (variable Breiten) | fixed (fixe Breiten) | inherit (wie Elternelement)

Tabelle 17.101: Referenz zur CSS-Eigenschaft »table-layout«

text-align

Bedeutung:	Textausrichtung
Typ:	Visuell
Vererbung:	Ja
Elemente:	Blockelemente, th, td
Werte:	Einer der folgenden Werte: **left** (linksbündig, Default bei links-nach-rechts-Schreibrichtung) | **right** (rechtsbündig, Default bei rechts-nach-links-Schreibrichtung) | center (zentriert) | justify (Blocksatz) | inherit (wie Elternelement)

Tabelle 17.102: Referenz zur CSS-Eigenschaft »text-align«

text-decoration

Bedeutung:	Textgestaltung
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Einer der folgenden Werte: **none** (keine spezielle Textgestaltung) | underline (unterstrichen) | overline (überstrichen) | line-through (durchgestrichen) | blink (blinkend) | inherit (wie Elternelement)

Tabelle 17.103: Referenz zur CSS-Eigenschaft »text-decoration«

text-indent

Bedeutung:	Texteinrückung
Typ:	Visuell
Vererbung:	Ja
Elemente:	Blockelemente, th, td
Werte:	Numerischer Wert mit Maßeinheit (auch Prozentualwert) oder der folgende Wert: inherit (wie Elternelement)

Tabelle 17.104: Referenz zur CSS-Eigenschaft »text-indent«

text-transform

Bedeutung:	Buchstabenumformung innerhalb von Text
Typ:	Visuell
Vererbung:	Ja
Elemente:	x,y,z
Werte:	Einer der folgenden Werte: capitalize (Kapitälchen) | uppercase (Klein- in Großbuchstaben umwandeln) | lowercase (Groß- in Kleinbuchstaben umwandeln) | **none** (keine Buchstabenumformung) | inherit (wie Elternelement)

Tabelle 17.105: Referenz zur CSS-Eigenschaft »text-transform«

top

Bedeutung:	Startposition von oben
Typ:	Visuell
Vererbung:	Nein

Tabelle 17.106: Referenz zur CSS-Eigenschaft »top«

Elemente:	Alle, die eine Angabe zur Eigenschaft position enthalten
Werte:	Numerischer Wert mit Maßeinheit (auch Prozentualwert) oder einer der folgenden Werte: auto (automatisch berechnen) | inherit (wie Elternelement)

Tabelle 17.106: Referenz zur CSS-Eigenschaft »top« (Fortsetzung)

unicode-bidi

Bedeutung:	Verhalten bei bidirektionaler Schrift
Typ:	Visuell
Vererbung:	Nein
Elemente:	x,y,z
Werte:	Einer der folgenden Werte: **normal** (kein besonderes Verhalten) | embed (Einbettung von Inline-Elementen mit anders gerichteter Schreibrichtung in den Textfluss) | bidi-override (Neuordnung von inneren Elementen mit unterschiedlicher gerichteter Schreibrichtung) | inherit (wie Elternelement)

Tabelle 17.107: Referenz zur CSS-Eigenschaft »unicode-bidi«

vertical-align

Bedeutung:	Vertikale Ausrichtung zu Nachbarelementen
Typ:	Visuell
Vererbung:	Nein
Elemente:	Inline-Elemente, td, th
Werte:	Numerischer Wert mit Maßeinheit (auch Prozentualwert), gemessen an der Basislinie, oder einer der folgenden Werte: **baseline** (Basislinie des Elements an der Basislinie des Elternelements ausrichten) | sub (Basislinie des Elements an der Position für tiefgestellten Text des Elternelements ausrichten) | super (Basislinie des Elements an der Position für hochgestellten Text des Elternelements ausrichten) | top (an der obersten möglichen Grenze ausrichten) | text-top (Elementobergrenze an der Schriftobergrenze des Elternelements ausrichten) | middle (vertikale Boxenmitte des Elements mittig an der Basislinie des Elternelements ausrichten) | bottom (an der untersten möglichen Grenze ausrichten) | text-bottom (Elementuntergrenze an der Schriftuntergrenze des Elternelements ausrichten) | inherit (wie Elternelement)

Tabelle 17.108: Referenz zur CSS-Eigenschaft »vertical-align«

visibility

Bedeutung:	Sichtbarkeit
Typ:	Visuell

Tabelle 17.109: Referenz zur CSS-Eigenschaft »visibility«

Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Einer der folgenden Werte: **visible** (sichtbar) | hidden (versteckt) | collapse (zusammenfallend) | inherit (wie Elternelement)

Tabelle 17.109: Referenz zur CSS-Eigenschaft »visibility« (Fortsetzung)

voice-family

Bedeutung:	Sprechertyp
Typ:	Aural
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Spezifischer Sprechertyp oder einer der folgenden generischen Sprechertypen: male (männliche Stimme) | female (weibliche Stimme) | child (Kinderstimme) oder der folgende Wert: inherit (wie Elternelement)

Tabelle 17.110: Referenz zur CSS-Eigenschaft »voice-family«

volume

Bedeutung:	Sprechlautstärke
Typ:	Aural
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Numerischer Wert zwischen 0 und 100 oder einer der folgenden Werte: x-soft (sehr leise) | soft (leise) | **medium** (mittel) | loud (laut) | x-loud (sehr laut) | inherit (wie Elternelement)

Tabelle 17.111: Referenz zur CSS-Eigenschaft »volume«

white-space

Bedeutung:	Darstellung von Leerraumzeichen
Typ:	Visuell
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist

Tabelle 17.112: Referenz zur CSS-Eigenschaft »white-space«

Werte:	Einer der folgenden Werte: **normal** (wie für das HTML-Element definiert) | pre (alle Leerraumzeichen werden dargestellt, wie im pre-Element von HTML) | nowrap (automatischen Zeilenumbruch verhindern im Element) | pre-wrap (alle Leerraumzeichen werden dargestellt, wie im pre-Element von HTML, aber zusätzlich greift der automatische Zeilenumbruch) | pre-line (Leerraumzeichen werden zu einem Leerzeichen zusammengefasst, der automatische Zeilenumbruch greift, aber auch im Quelltext eingegebene Zeilenumbrüche werden als Zeilenumbrüche dargestellt) | inherit (wie Elternelement)

Tabelle 17.112: Referenz zur CSS-Eigenschaft »white-space« (Fortsetzung)

widows

Bedeutung:	Alleinstehende Zeilen am Seitenanfang
Typ:	Visuell, seitenorientiert
Vererbung:	Ja
Elemente:	Alle Blockelemente
Werte:	Numerischer Wert (Anzahl zu einem Block gehörender Zeilen, die am Seitenanfang nach einem Seitenumbruch stehen müssen) | der folgende Wert: inherit (wie Elternelement)

Tabelle 17.113: Referenz zur CSS-Eigenschaft »widows«

width

Bedeutung:	Breite eines Elements
Typ:	Visuell
Vererbung:	Nein
Elemente:	img, object und Blockelemente außer tbody, tfoot, thead und tr
Werte:	Numerischer Wert mit Maßeinheit (auch Prozentualwert) oder einer der folgenden Werte: **auto** (automatisch berechnen) | inherit (wie Elternelement)

Tabelle 17.114: Referenz zur CSS-Eigenschaft »width«

word-spacing

Bedeutung:	Wortabstand
Typ:	Visuell
Vererbung:	Ja
Elemente:	Alle, bei denen die Eigenschaft sinnvoll anwendbar ist
Werte:	Numerischer Wert mit Maßeinheit oder einer der folgenden Werte: **normal** (normaler Wortabstand) | inherit (wie Elternelement)

Tabelle 17.115: Referenz zur CSS-Eigenschaft »word-spacing«

z-index

Bedeutung:	Schichtposition bei sich überdeckenden Elementen
Typ:	Visuell
Vererbung:	Nein
Elemente:	Alle, die eine Angabe zur Eigenschaft position enthalten
Werte:	Numerischer Wert (je höher, desto weiter vorne, je niedriger, desto weiter hinten) oder einer der folgenden Werte: **auto** (automatisch vergeben) | inherit (wie Elternelement)

Tabelle 17.116: Referenz zur CSS-Eigenschaft »z-index«

Index

!

" 381
.htaccess 481, 583
.NET-Framework 597
@charset (CSS) 302
@import (CSS) 297
@media (CSS) 298
@page (CSS) 300
@-Regeln (CSS) 297

A

a (HTML-Element) 181, 1027
abbr (HTML-Attribut) 108, 1051
abbr (HTML-Element) 97, 1028
Abgeleitete Klassen (PHP) 678
Abmahnungen 1004
abs() (JavaScript) 356
abs() (PHP-Funktion) 748
Absatzformate (HTML) 88
Absatzschaltungen 91
Absolute Adressierung 185
Absolute Positionierung 144
Abstände 131
abstract PHP) 693
Abstrakte Klassen 686, 693
accept (HTML-Attribut) 217, 1052
accept-charset (HTML-Attribut) 1052
access_module (Apache-Modul) 477
AccessFileName (Apache-Direktive) 481
Accessibility 37, 41
accesskey (HTML-Attribut) 192, 223, 1052
acos() (JavaScript) 356
acos() (PHP-Funktion) 748
acronym (HTML-Element) 97, 1028
Action (Apache-Direktive) 601, 603
action (HTML-Attribut) 212, 1052
Active Server Pages (ASP) 597
AddAlt (Apache-Direktive) 564
AddDescription (Apache-Direktive) 564
AddHandler (Apache-Direktive) 576, 601, 603
AddIcon (Apache-Direktive) 564
address (HTML-Element) 96, 1028
AddType (Apache-Direktive) 601f.
admin-c 24
Adobe Photoshop 33
Adressbasis 87
Änderungsmarkierungen 99
after (CSS-Pseudoformat) 178
Aggregatoren 880
alert() (JavaScript) 368
alias (Linux-Kommando) 540
alias_module (Apache-Modul) 477
Aliasnamen (SQL) 803
align (HTML-Attribut) 1052
Allgemeine Geschäftsbedingungen (AGB) 996
allow_url_fopen (PHP-Option) 609
AllowOverride (Apache-Direktive) 480
Alpha-Filter 198
alt (HTML-Attribut) 201, 1053
alter table (SQL) 787
Alternative Teilausdrücke (reguläre Ausdrücke) 701
Animierte Grafiken 200
Anker 185
Ankernamen 187
Anleitung zu Straftaten 992
Anti-Aliasing 199
Anweisungen (JavaScript) 326
Anweisungen (PHP) 642
Anweisungsblöcke (JavaScript) 326

Anwenderkommunikation 39
Anwendungsprotokolle 463
Apache 469
  .htaccess 481
  apachectl 473
  Download 470
  httpd.conf 474
  Installation 471f.
  Logdateien 484
  Module 476
  starten, stoppen, neustarten 473
  Versionen 470
  virtuelle Hosts 481
appendChild() (DOM) 385
Apple Macintosh 28
appName (JavaScript) 373
appVersion (JavaScript) 373
Arbeitsspeicher 27, 526
Arbeitsumgebung 27
Arbeitsverzeichnis 749
archive (HTML-Attribut) 1053
Archivieren (Linux) 518
area (HTML-Element) 204, 1029
Arithmetische Operatoren (JavaScript) 337
Arithmetische Operatoren (PHP) 661
array_key_exists() (PHP-Funktion) 740
array_map() (PHP-Funktion) 741
array_pop() (PHP-Funktion) 738
array_push() (PHP-Funktion) 736
array_reverse() (PHP-Funktion) 741
array_search() (PHP-Funktion) 740
array_shift() (PHP-Funktion) 738
array_splice() (PHP-Funktion) 737
array_sum() (PHP-Funktion) 741
array_unique() (PHP-Funktion) 741
array_unshift() (PHP-Funktion) 736
Arrays (JavaScript) 362
  assoziative 363
Arrays (PHP) 649, 735
  durchsuchen 739
  sortieren 738
arsort() (PHP-Funktion) 738
ASCII-Zeichensatz 65
asin() (JavaScript) 357
asin() (PHP-Funktion) 748
asort() (PHP-Funktion) 738
asp_tag (PHP-Option) 606
Assoziative Arrays (JavaScript) 363
atan() (JavaScript) 357
atan() (PHP-Funktion) 748
atan2() (PHP-Funktion) 748
Attributbedingte Selektoren 174
Attribute 864
  HTML 62
Attribute definieren (DTD) 912
Aufmerksamkeit 38
Aufzählungslisten 92
Ausdrücke (JavaScript) 326
Ausdrücke (PHP) 643
Ausgaben (mit PHP) 714
Ausklappmenüs 273
Auskunftspflicht 1001
Ausrichtung 132
Auswahllisten 218
Auszeichnungssprachen 25
auth_module (Apache-Modul) 477
AuthGroupFile (Apache-Direktive) 584
AuthName (Apache-Direktive) 583
AuthUserFile (Apache-Direktive) 584
auto_increment (SQL) 786
autoindex_module (Apache-Modul) 477
Automatische Typumwandlung (JavaScript) 329
Automatische Überschriftennummerierung 292
Automatische Weiterleitung 586
Autorensoftware Flash 34
availHeight (JavaScript) 373
availWidth (JavaScript) 373
avg() (MySQL-Funktion) 802
axis (HTML-Attribut) 1053
azimuth (CSS-Eigenschaft) 1081

B

b (HTML-Element) 98, 1029
back() (JavaScript) 369
background (CSS-Eigenschaft) 142, 1083
background-attachment (CSS-Eigenschaft) 142, 1082
background-color (CSS-Eigenschaft) 123, 137, 139, 1082
background-image (CSS-Eigenschaft) 138, 1082
background-position (CSS-Eigenschaft) 140, 142, 1082
background-repeat (CSS-Eigenschaft) 138, 1083
BAD REQUESTNO CONTENT (HTTP-Statuscode) 466
Banner 1014

Banner-Blindheit 38
Barrierefreiheit 41
base (HTML-Element) 87, 283, 1029
bash 494
bashrc 540
Batch-Betrieb (MySQL-Client) 782
bcadd() (PHP-Funktion) 768
bcdiv() (PHP-Funktion) 768
bcmul() (PHP-Funktion) 768
bcpow() (PHP-Funktion) 768
bcsqrt() (PHP-Funktion) 768
bcsub() (PHP-Funktion) 768
bdo (HTML-Element) 1030
before (CSS-Pseudoelement) 294
before (CSS-Pseudoformat) 178
begin (SQL) 811
Behindertengleichstellung 42
Beleidigung 992
Benutzer (Linux) 504, 541
Benutzer (MySQL) 811
Benutzer wechseln 510
Benutzer-ID (uid) 507
Benutzertypen (Linux) 521
Benutzerverzeichnis (Linux) 498
Berechtigungen (MySQL) 812
Bestätigungsfenster (JavaScript) 368
Betriebssysteme 27
big (HTML-Element) 98, 1030
bigint (MySQL) 784
Bildbearbeitung 197
bin (Linux-Verzeichnis) 497
bindec() (PHP-Funktion) 748
Bit-Operatoren (PHP) 663
blob (MySQL) 784
Block-Elemente 62, 88, 126
blockquote (HTML-Element) 95, 1030
Boards 1019
body (HTML-Element) 59, 1031
Boolean (JavaScript) 330
Boolean (PHP) 648
boot (Linux-Verzeichnis) 497
Boot-Konfiguration 539
border (CSS-Eigenschaft) 134, 1083
border (HTML-Attribut) 101, 1053
border-bottom (CSS-Eigenschaft) 134, 1084
border-bottom-color (CSS-Eigenschaft) 1084
border-bottom-style (CSS-Eigenschaft) 1084, 1087
border-bottom-width (CSS-Eigenschaft) 1084
border-collapse (CSS-Eigenschaft) 156, 1085
border-color (CSS-Eigenschaft) 1085
border-left (CSS-Eigenschaft) 134, 1085
border-left-color (CSS-Eigenschaft) 1085
border-left-style (CSS-Eigenschaft) 1086
border-left-width (CSS-Eigenschaft) 1086
border-right (CSS-Eigenschaft) 134, 1086
border-right-color (CSS-Eigenschaft) 1087
border-right-width (CSS-Eigenschaft) 1087
border-spacing (CSS-Eigenschaft) 161, 1087
border-style (CSS-Eigenschaft) 1088
border-top (CSS-Eigenschaft) 134, 1088
border-top-color (CSS-Eigenschaft) 1088
border-top-style (CSS-Eigenschaft) 1089
border-top-width (CSS-Eigenschaft) 1089
border-width (CSS-Eigenschaft) 1089
bottom (CSS-Eigenschaft) 1089
Bourne Again Shell (bash) 494
Bourne-Shell 494
Boxmodell (CSS) 125
br (HTML-Element) 97, 1031
Brainstorming, geräteunabhängiges 44
break (JavaScript) 343, 347
break (PHP) 667
Browser 31
Browser-Markt 32
Browser-Weichen 302
Bullet-Symbol 92, 155
button (HTML-Element) 1031
Buttons 213
bzip2 (Linux-Kommando) 532

C

C (Programmiersprache) 490
C Shell 494
Caching (Datenbankabfragen) 808
caption (HTML-Element) 101, 1032
caption-side (CSS-Eigenschaft) 1090
Cascading Style Sheets 109
case (JavaScript) 343
case (Linux-Kommando) 555
case (PHP) 666
cat (Linux-Kommando) 508, 517
catch (PHP) 694
cd (Linux-Kommando) 512

CDATA 316, 873
CDTATA 872
ceil() (JavaScript) 357
ceil() (PHP-Funktion) 748
cellpadding (HTML-Attribut) 1053
cellspacing (HTML-Attribut) 161, 1054
cgi_module (Apache-Modul) 477
CGI-Aufrufmöglichkeiten (in HTML) 577
CGI-Schnittstelle 574
CGI-Scripts 575f.
CGI-Scripts (mit SSI) 573
CGI-Verzeichnisse 575
Channel (RSS) 884
char (HTML-Attribut) 1054
char (MySQL) 785
charAt() (JavaScript) 359
charCodeAt() (JavaScript) 359
charoff (HTML-Attribut) 1054
charset (HTML-Attribut) 192, 1054
charset (JavaScript) 372
chdir() (PHP-Funktion) 750
Checkboxen 213
checkdate() (PHP-Funktion) 745
checked (HTML-Attribut) 214, 1054
chgrp (Linux-Kommando) 524
childNodes (DOM) 380, 430
chmod (Linux-Kommando) 524
chown (Linux-Kommando) 524
cite (HTML-Attribut) 95, 97, 1055
cite (HTML-Element) 97, 1032
class (HTML-Attribut) 171, 225, 1055
class (PHP) 675
class_exists() (PHP-Funktion) 768
classid (HTML-Attribut) 206f., 210, 1055
clear (CSS-Eigenschaft) 150, 1090
clientHeight (JavaScript-Eigenschaft) 288
Client-Server-Struktur 35
Client-Server-Struktur (DBMS) 772
clientWidth (JavaScript-Eigenschaft) 288
clip (CSS-Eigenschaft) 1090
clone() (PHP-Funktion) 694
closedir() (PHP-Funktion) 751
cm (CSS-Maßeinheit) 117
CMYK-Modell 198
code (HTML-Element) 97, 1032
codebase (HTML-Attribut) 206f., 1055
Code-basierte Editoren 29
codetype (HTML-Attribut) 207, 1055
col (HTML-Element) 105, 1033
colgroup (HTML-Element) 105, 1033
color (CSS-Eigenschaft) 122, 1091
colorDepth (JavaScript) 373
cols (HTML-Attribut) 217, 280, 1056
colspan (HTML-Attribut) 106, 1056
commit (SQL) 811
Compliant Mode 304
concat() (JavaScript) 364
Conditional Comments 305
config (SSI-Anweisung) 570
configure (Script) 532
confirm() (JavaScript) 368
construct() (PHP-Funktion) 691
content (CSS-Eigenschaft) 178, 294, 1091
content (HTML-Attribut) 79, 1056
Content-Breaks 1015
Content-Syndication 882
continue (JavaScript) 347
convert_uudecode() (PHP-Funktion) 735
convert_uuencode() (PHP-Funktion) 735
cookie (JavaScript) 373
COOKIE (PHP-Array) 655
Cookies (PHP) 655
COOKIES (PHP-Array) 719
coords (HTML-Attribut) 204, 1056
copy() (PHP-Funktion) 756
Copyright-Zeichen 1073
CorelDRAW 33
cos() (JavaScript) 357
cos() (PHP-Funktion) 748
count() (MySQL-Funktion) 796, 802
count() (PHP-Funktion) 735
counter-increment (CSS-Eigenschaft) 294, 1091
counter-reset (CSS-Eigenschaft) 294, 1091
cp (Linux-Kommando) 502, 516
crc32() (PHP-Funktion) 734
create database (SQL) 782
create table (SQL) 783
createAttribute() (DOM) 388
createElement() (DOM) 384
createTextNode() (DOM) 384
cron.allow 546
cron.deny 546
crond 544
crontab (Datei) 545
crontab (Linux-Kommando) 544
crypt (Linux-Kommando) 585
CSE HTML-Validator 36

CSS 109
Boxmodell 125
Farbangaben 112
Maßeinheiten 117
numerische Werte 118
Seitenlayouts 227
Syntax 111
CSS-Dateien 164
CSS-Eigenschaften 1081
CSS-Eigenschaften (DOM) 389
CSS-Formatdefinitionen 111
ctype_alnum() (PHP-Funktion) 769
ctype_alpha() (PHP-Funktion) 769
ctype_digit() (PHP-Funktion) 769
ctype_lower() (PHP-Funktion) 769
ctype_print() (PHP-Funktion) 769
ctype_upper() (PHP-Funktion) 769
ctype_xdigit() (PHP-Funktion) 769
cue (CSS-Eigenschaft) 1092
cue-after (CSS-Eigenschaft) 1092
cue-before (CSS-Eigenschaft) 1092
cursor (CSS-Eigenschaft) 314, 1092
CustomLog (Apache-Direktive) 486

D

Daemon 461
data (HTML-Attribut) 206, 210, 1056
Date (JavaScript) 352
date (Linux-Kommando) 530
date (MySQL) 785
date() (PHP-Funktion) 742
Datei (Linux)
anzeigen 517
kopieren 516
löschen 517
suchen 513
umbenennen 516
verschieben 516
Datei (PHP)
lesen 708
öffnen 708
Datei-Management (PHP) 756
Dateirechte (Linux) 525
Datei-Upload 216, 657
Dateiverwaltung 36
Dateiverwaltung (Linux) 510
Dateizeiger 712
Datenbank Management System (DBMS) 771
Datenbanken (MySQL)
anlegen 782
löschen 782
Datenbank-Management-System (DBMS) 860
Dateninseln 874
Datenintegritätsprüfung 855
Datenkapselung 685f.
Datensätze (MySQL)
ändern 792
hinzufügen 790
löschen 792
Datenschutz 1000
Datensicherung 457
Datentypen (JavaScript) 328
Datentypen (PHP) 645
Datenübergabe (JavaScript) 370
Datenzellen 102
datetime (HTML-Attribut) 99, 1057
datetime (MySQL) 785
Datum (JavaScript) 352
DBMS 771, 860
dd (HTML-Element) 96, 1033
Dearchivieren (Linux) 518
Debian Linux 492
decbin() (PHP-Funktion) 748
dechex() (PHP-Funktion) 748
decimal (MySQL) 785
declare (HTML-Attribut) 1057
decoceURI() (JavaScript) 374
decoct() (PHP-Funktion) 748
Deeplinks 1000
auf fremde Quellen 183
default (SQL) 786
Default-Dateinamen 562
DefaultIcon (Apache-Direktive) 564
Default-Route 438
defer (HTML-Attribut) 316, 1057
define() (PHP-Funktion) 654
Deklaration, Dokumenttyp 69
Dekomprimieren (Linux) 518
Dekrementierung (PHP) 661
del (HTML-Element) 99, 1034
delete from (SQL) 792
Denic 442
Deny from (Apache-Direktive) 480
describe (SQL) 781
Design, geräteunabhängiges 43
destruct() (PHP-Funktion) 691

dev (Linux-Verzeichnis) 497
df (Linux-Kommando) 525
dfn (HTML-Element) 1034
dir (HTML-Attribut) 225, 1057
dir_module (Apache-Modul) 477
direction (CSS-Eigenschaft) 1093
Directory-Browsing 563
DirectoryIndex (Apache-Direktive) 481, 562
Direktaufruf Methoden (PHP) 679
Direktiven (httpd.conf) 475
disabled (HTML-Attribut) 1057
display (CSS-Eigenschaft) 149f., 1093
display_errors (PHP-Option) 607
display_startup_errors (PHP-Option) 607
Dithering 199
div (HTML-Element) 90, 1034
dl (HTML-Element) 96, 1035
DNS (Domain Name Service) 441
do...while (JavaScript) 346
do...while (PHP) 670
doc_root (PHP-Option) 609
DocBook 877
document (JavaScript) 372, 378
Document Object Model (DOM) 311
Document Root 561
Document Type Definition 866
Document Type Definition (DTD) 70
DocumentRoot (Apache-Direktive) 476, 479
Dokument-Element 866
Dokumenttitel 77
Dokumenttyp 69
Dokumenttyp-Definitionen 903
Dokumenttyp-Deklaration 865, 867
DOM 864, 945
DOM (Document Object Model) 311
Domain Name Service (DNS) 441
Domain-Namen 455, 1002
Domain-Vergabestellen 442
DOM-Klasse 950
DOM-Knotenobjekte 955
double (MySQL) 785
Dreispaltiges Layout 231
drop database (SQL) 782
drop table (SQL) 783
Drop-down-Listen 218
DSL 440
dt (HTML-Element) 96, 1035
DTD 866
DTDs 903
Dublin Core 81
Dynamische Banner 1015
Dynamische IP-Adressen 440
Dynamische Webseiten 20
Dynamisches SQL 774

E

E (JavaScript) 356
easter_date() (PHP-Funktion) 746
echo (Linux-Kommando) 494
echo (PHP-Konstrukt) 615, 619, 714
echo (SSI-Anweisung) 571
ECMA-262 310
EDI 891
Editoren
  code-basierte 29
  Linux 30
  Macintosh 30
  MS Windows 30, 925
  WYSIWYG 29
Eigenschaften (JavaScript) 351
Eigenschaften (PHP-Objekte) 675
Eingabefelder 213
Eingabefenster (JavaScript) 369
Eingebettete Frames 289
Einrückung 134
Einwahlknoten 435
Electronic Document Interchange 891
Elemente 864
  HTML 60
Elemente definieren (DTD) 906
Elementinhalt 129
Elementverschachtelung 170
  HTML 61
elevation (CSS-Eigenschaft) 1094
else (JavaScript) 341
else (PHP) 665
else if (JavaScript) 342
elseif (PHP) 665
em (CSS-Maßeinheit) 117
em (HTML-Element) 97, 1035
E-Mail-Verweise 190
Empfehlungen (W3C) 49
empty() (PHP-Funktion) 706
empty-cells (CSS-Eigenschaft) 1094
encodeURI() (JavaScript) 374
enctype (HTML-Attribut) 212, 216, 1058

engine (PHP-Option) 606
Entities 65, 68, 1072
Entities (HTML) 64
Entities definieren (DTDs) 915
Entwickler 23
Entwicklungsumgebung 35
enum (MySQL) 785
ENV (PHP-Array) 655
env_module (Apache-Modul) 477
Envelope(SOAP) 891
Ergebnisdaten (PHP/MySQL) 832
error_log (PHP-Option) 608
error_reporting (PHP-Option) 607
ErrorDocument (Apache-Direktive) 481, 590
Erweiternde Klassen (PHP) 678
Erzwungener Zeilenumbruch 97
Erzwungenes Leerzeichen 1073
Escape-Sequenzen (JavaScript) 332
escapeshellcmd() (PHP-Funktion) 767
etc (Linux-Verzeichnis) 497
European Computer Manufacturers Association (ECMA) 310
eval() (JavaScript) 374
Event-Handler 312, 314, 317
exec (SSI-Anweisung) 573
exec() (PHP-Funktion) 766
ExecCGI (Apache-Option) 479
Existenz prüfen (JavaScript) 374
exit (Linux-Kommando) 510
exp() (JavaScript) 357
exp() (PHP-Funktion) 748
Expat 960
Expat (XML-Parser) 945
expires_module (Apache-Modul) 477
explode() (PHP-Funktion) 621, 729
Explorer-Menü (JavaScript) 408
expression() (CSS-Funktion) 288
extends (PHP) 678
Extensible Business Reporting Language 877
Extensible Markup Language 857
Extensible Markup Language (XML) 55
Extensible Stylesheet Language 923
Eye-Catcher 38

F

Fallunterscheidung (JavaScript) 343
Fallunterscheidung (PHP) 666
Fallunterscheidungen (Shellscripts) 555
false (JavaScript) 330
FancyIndexing (Apache-Direktive) 564
Farbangaben (CSS) 112
Farben (websichere) 116, 1
Farbmodelle 197
Farbnamen 114, 1
Umgebungsfarben 115
Farbtiefe 197
Faults (SOAP) 896
Favicons 1023
fclose() (PHP-Funktion) 621, 711
Fehlerseiten (Apache) 589
Fehlertoleranz 60
Feiertagsberechnung 746
Feldoptionen (MySQL) 786
Feldtypen (MySQL) 784
feof() (PHP-Funktion) 714
Feste IP-Adressen 440
Festplattenspeicher 525
fgetc() (PHP-Funktion) 713
fgets() (PHP-Funktion) 621, 712
fieldset (HTML-Element) 222, 1036
file() (PHP-Funktion) 628, 709, 958ff.
file_cache_module (Apache-Modul) 477
file_exists() (PHP-Funktion) 708
file_get_contents() (PHP-Funktion) 627, 710
file_put_contents() (PHP-Funktion) 722
file_uploads (PHP-Option) 609
fileatime() (PHP-Funktion) 754
filectime() (PHP-Funktion) 754
filegroup() (PHP-Funktion) 754
fileinode() (PHP-Funktion) 754
File-Locking 722
filemtime() (PHP-Funktion) 754
fileowner() (PHP-Funktion) 755
fileperms() (PHP-Funktion) 755
FILES (PHP-Array) 655
filesize() (PHP-Funktion) 616, 711, 755
filetype() (PHP-Funktion) 755
File-Upload 216, 657
filter (CSS-Eigenschaft) 155
final (PHP) 692
Finale Klassen/Methoden 686, 692
find (Linux-Kommando) 513
Firefox 31
first-child (CSS-Pseudoformat) 177
first-letter (CSS-Pseudoformat) 176
first-line (CSS-Pseudoformat) 177
Fixe Bereiche 284

Fixe Positionierung 144
Flash 34, 205, 309
  ohne HTML 207
Flash MX 34
flastmod (SSI-Anweisung) 570
FlaX 34
Fließkommazahlen (JavaScript) 330
float (CSS-Eigenschaft) 149, 202, 1094
Float (JavaScript) 330
float (MySQL) 785
flock() 722
floor() (JavaScript) 357
floor() (PHP-Funktion) 748
flush() (PHP-Funktion) 717
Flussdiagramme 45
FollowSymLinks (Apache-Option) 479
font (CSS-Eigenschaft) 1094
font-family (CSS-Eigenschaft) 119, 1095
font-size (CSS-Eigenschaft) 121, 1095
font-style (CSS-Eigenschaft) 123, 1095
font-variant (CSS-Eigenschaft) 124, 1096
font-weight (CSS-Eigenschaft) 123, 1096
fopen() (PHP-Funktion) 621, 710
for (HTML-Attribut) 223, 1058
for (JavaScript) 344
for (Linux-Kommando) 556
for (PHP) 667
for in (JavaScript) 345
FORBIDDENNO CONTENT (HTTP-Statuscode) 467
foreach (PHP) 669
Foren 1019
form (HTML-Element) 212, 1036
Formatstring (date) 742
Formatstring (printf) 715
Formatting objects 923
Form-Mailer
  in Perl 577
  in PHP 629
Formulare 210
Formularelemente 213
Formularüberprüfung (JavaScript) 393
Formularverarbeitung (PHP) 629
forward() 369
fputs() (PHP-Funktion) 721
frame (HTML-Attribut) 103, 1058
frame (HTML-Element) 281, 284
frameborder (HTML-Attribut) 284, 291
Frame-Eigenschaften 283
Frames 277
  eingebettete 289
frameset (HTML-Element) 280
Frame-Technik 20
fread() (PHP-Funktion) 711
free (Linux-Kommando) 526
FreeBSD 492
fromCharCode() (JavaScript) 359
fseek() (PHP-Funktion) 713
fsize (SSI-Anweisung) 570
ftell() (PHP-Funktion) 713
Full Table Scan 787
function (PHP) 671
function_exists() (PHP-Funktion) 769
Funktionen (JavaScript) 347
Funktionen (PHP) 671
Funktionen (Shellscripts) 552
Funktionsaufrufe (PHP) 672
fwrite() (PHP-Funktion) 721

G

Ganzzahlen (JavaScript) 330
Gateways 437
Gecko-Engine 311
Geistiges Eigentum 997
Generische Schriftart 120
Gentoo Linux 492
Geography Markup Language 877
Geräteunabhängiges Design 43
Geschäftsmäßige Teledienste 995
Geschützte Webseiten 583
Gesetz 993
  der Ähnlichkeit 39
  der Nähe 38
GET (Methode) 468
GET (PHP-Array) 655, 704
get_class_methods() (PHP-Funktion) 768
get_class_vars() (PHP-Funktion) 769
get_declared_classes() (PHP-Funktion) 769
get_object_vars() (PHP-Funktion) 769
getAttribute() (DOM) 388
getcwd() (PHP-Funktion) 750
getDate (JavaScript) 353
GET-Daten 565
getDay (JavaScript) 353
getElementById() (DOM) 378
getElementsByName() (DOM) 378
getElementsByTagName() (DOM) 378
getFullYear (JavaScript) 353
get-Funktionen (Getter) 688
getHours (JavaScript) 353
GetImageSize() (PHP-Funktion) 616

getMilliseconds (JavaScript) 353
getMinutes (JavaScript) 353
getMonth (JavaScript) 353
GET-Request 464
getSeconds (JavaScript) 353
getTime (JavaScript) 353
getTimezoneOffset (JavaScript) 353
getUTC... (JavaScript) 353
gid (Gruppen-ID) 507
Gierigkeitsprinzip (reguläre Ausdrücke) 700
GIF-Format 195
Gitternetz 103
global (PHP-Konstrukt) 652
Globale Variablen (JavaScript) 327
Globale Variablen (PHP) 652
GLOBALS (PHP-Array) 653
Glossar 96
gmdate() (PHP-Funktion) 615
GML 877
GNU-Projekt 492
Google-Suche (Webserive) 972
Grafikformate 195
Grafikkarte 27
Grafikreferenzen 201
Grafiksoftware 33
Grafische Oberflächen (Linux) 495
grant (SQL) 814
Graustufen 197
Griechische Buchstaben 1075
Groß- und Kleinschreibung, HTML 61
group (Datei) 541
group by (SQL) 802
groupadd (Linux-Kommando) 506
groupdel (Linux-Kommando) 508
groupmod (Linux-Kommando) 507
groups (Linux-Kommando) 508
Gruppen (Linux) 504, 541
Gruppen-ID (gid) 507
Gültige Dokumente 868f.
gunzip (Linux-Kommando) 520
gzip (Linux-Kommando) 519

H

h1-h6 (HTML-Elemente) 90, 1036
Haftung 999
Hardware 27
Harte Links (Linux) 520
Hash-Funktion 733
head (HTML-Element) 59, 1037
header() (PHP-Funktion) 718
HeaderName (Apache-Direktive) 565
headers (HTML-Attribut) 108, 1058
height (CSS-Eigenschaft) 143, 217, 1096
height (HTML-Attribut) 201, 206, 1058
height (JavaScript) 373
here-Dokumente (PHP) 619
Hexadezimale RGB-Farbangaben 114
Hexadezimalschreibweise (JavaScript) 333
hexdec() (PHP-Funktion) 748
Hierarchische DBMS 772
high_priority (SQL) 807
Hintergrundgrafik 137
history (JavaScript) 369
home (Linux-Verzeichnis) 498
HOME (Umgebungsvariable) 500
Homepage 21
Horizontale Navigationsleiste 269, 273
Host 439
Hosting-Provider 443
hosts (Datei) 441
hr (HTML-Element) 96, 1037
href (HTML-Attribut) 85, 87, 181, 205, 1059
href (JavaScript) 370
hreflang (HTML-Attribut) 192, 1059
HTML 55f., 58
  Attribute 62
  Darstellung 59
  Elemente 60
  Entities 64
  Groß- und Kleinschreibung 61
  Kommentare 72
  Quelltext 58
  Strict 70
  Tags 60
  Transitional 71
  Zeichen 63
html (HTML-Element) 59, 1037
HTML PowerTools 36
HTML Tidy 36
html_entity_decode() (PHP-Funktion) 732
html_errors (PHP-Option) 608
HTML-Attribute 224
HTML-Attribute (DOM) 385
HTML-Elemente 1027
HTML-Entities 65, 68
htmlentities() (PHP-Funktion) 731
htmlspecialchars() (PHP-Funktion) 731
HTML-Templates
  mit PHP 622
  mit SSI 568

httpd.conf 474
http-equiv (HTML-Attribut) 78, 587, 1059
HTTP-Header (mit PHP) 717
HTTP-Protokoll 463
HTTP-Statuscodes 466, 591
Hyperlinks 178
  gestalten 193
hypot() (PHP-Funktion) 748

I

i (HTML-Element) 98, 1037
id (HTML-Attribut) 90, 171, 225, 1059
id (Linux-Kommando) 509
ID-Namen 173
IETF 52
if (JavaScript) 341
if (Linux-Kommando) 553
if (PHP) 665
iframe (HTML-Element) 290
ignore_repeated_errors (PHP-Option) 608
ignore_repeated_source (PHP-Option) 608
Image-Maps 203
img (HTML-Element) 201, 1038
implode() (PHP-Funktion) 629, 741
Impressumpflicht 994
in (CSS-Maßeinheit) 117
in_array() (PHP-Funktion) 740
include (SSI-Anweisung) 569
include() (PHP-Funktion) 639
include_module (Apache-Modul) 477
include_once() (PHP-Funktion) 638
Includes (Apache-Option) 479
IncludesNOEXEC (Apache-Option) 480
Indexe (MySQL) 787, 805
Indexes (Apache-Option) 480
indexOf() (JavaScript) 359
IndexOptions (Apache-Direktive) 565
IndexOrderDefault (Apache-Direktive) 565
inetd 461
info (Linux-Kommando) 503
Informations- und Kommunikationsdienstegesetz (IuKDG) 993
ini_get() (PHP-Funktion) 770
ini_get_all() (PHP-Funktion) 770
ini_restore() (PHP-Funktion) 770
ini_set() (PHP-Funktion) 770
inittab 539
Inkrementierung (PHP) 661
Inline-Elemente 62, 97, 127
Inline-Frames 289
Innenabstand 129, 134
innerHTML (JavaScript/DOM) 313
InnoDB-Tabellen 809
Inode 510
input (HTML-Element) 213, 1038
ins (HTML-Element) 99, 1038
instanceof (PHP) 694
Instanzen (JavaScript) 351
int (MySQL) 785
Integer (JavaScript) 330
Interlacing 200
INTERNAL SERVER ERRORNO CONTENT (HTTP-Statuscode) 467
Interne DTD 871
Internet Draft 52
Internet Engineering Task Force (IETF) 52
Internet Explorer 303
Internet Protocol (IP) 439, 459
Internetanbindung 27
Internetverbindung 435
Interpretierte Inhalte (PCDATA) 873
Interpunktionszeichen 1079
IP-Adresse 561
Irfan View 33
is_array() (PHP-Funktion) 705
is_dir() (PHP-Funktion) 753
is_executable() (PHP-Funktion) 754
is_file() (PHP-Funktion) 709, 753
is_finite() (PHP-Funktion) 749
is_infinite() (PHP-Funktion) 749
is_link() (PHP-Funktion) 753
is_nan() (PHP-Funktion) 749
is_readable() (PHP-Funktion) 708, 754
is_writable() (PHP-Funktion) 709, 754
ISAM 809
isFinite() (JavaScript) 374
ismap (HTML-Attribut) 1060
isNaN() (JavaScript) 374
ISO 10646 63
ISO/IEC 10646 872
ISO-8859-1 64f., 1073
ISO-8859-15 64, 66
isset() (PHP-Funktion) 665, 705
IuKDG 993

J

Java Server Pages (JSP) 597
Java-Applets 207

JavaScript 310
in HTML einbetten 312
javascript (Pseudo-URI-Schema) 318
JavaScript-Bereiche 315, 319
JavaScript-Dateien 321
JavaScript-Sammlungen 393
join() (JavaScript) 364
JPEG-Format 195
JScript 310

K

Kalenderberechnung 746
Kanal (RSS) 884
kbd (HTML-Element) 97, 1039
KDE-Oberfläche 495
KeepAlive (Apache-Direktive) 476
KeepAliveTimeOut (Apache-Direktive) 476
Kernel 493
key (SQL) 787
KK-Antrag 456
Klasse-A-(B-, C-)Netze 439
Klassen (CSS) 171
Klassen (PHP) 675
abgeleitete 678
abstrakte 686, 693
erweiternde 678
Finale 686, 692
Klassennamen 173
Knoppix 492
Knoten (DOM) 378
Kommandos (Shell) 495, 501
Kommentare, HTML 72
Kommentare (JavaScript) 325
Kommentare (PHP) 644
Kommunikation 39
Komprimieren (Linux) 518
Konfigurationsdateien 539
Konnektivitäts-Koordination 456
Konstanten (PHP) 653
Konstruktorfunktion (PHP) 677
Kontrast 123
Kontrollstrukturen (JavaScript) 341
Kontrollstrukturen (PHP) 664
Kontrollstrukturen (Shellscripts) 553
Konzeptphase, geräteunabhängige 44
KoolMoves 34
Kopfzellen 102
Kopieren, Dateien (Linux) 516
KornShell 494
Kostennote 1006
krsort() (PHP-Funktion) 739
Kryptografische Funktionen (PHP) 733
ksort() (PHP-Funktion) 738

L

label (HTML-Attribut) 219, 1060
label (HTML-Element) 223, 1039
LAMP-Konfiguration 457
LAN 437
lang (HTML-Attribut) 225, 1060
language (JavaScript) 373
Lastenheft, geräteunabhängiges 45
lastIndexOf() (JavaScript) 359
lastModified (JavaScript) 372
Launch 1007
Layer 151
Layer-Technik 198
left (CSS-Eigenschaft) 145, 1096
Left Joins (MySQL) 804
legend (HTML-Element) 222, 1039
length (JavaScript, Arrays), assoziative 364
length (JavaScript, Zeichenketten) 358
Lesen von Dateien (PHP) 708
less (Linux-Kommando) 518
letter-spacing (CSS-Eigenschaft) 124, 1097
li (HTML-Element) 92, 1040
lib (Linux-Verzeichnis) 498
libXML 950
lilo.conf 539
limit (SQL) 798
line-height (CSS-Eigenschaft) 1097
link (HTML-Element) 85, 305, 1024, 1040
Linkanker 185
Linkchecker 36
Linkhaftung 1000
Linkpopularität 1010, 1012
Links 179
zu anderen Frames 283
Links (Linux) 520
Linux 28, 490
Verzeichnisbaum 496
Linux-Kernel 494
Listen, nummerierte 93
Listen (HTML) 92, 155
list-style (CSS-Eigenschaft) 1097
list-style-image (CSS-Eigenschaft) 156, 1098
list-style-position (CSS-Eigenschaft) 156, 1098

list-style-type (CSS-Eigenschaft) 155, 1098
ln (Linux-Kommando) 520, 587
LN10 (JavaScript) 356
LN2 (JavaScript) 356
LoadModule (Apache-Direktive) 476, 601f.
localhost 442, 562
location (JavaScript) 369
Löschen
  Datei (Linux) 517
  Verzeichnis (Linux) 512
log($x) (PHP-Funktion) 748
log() (JavaScript) 357
log_config_module (Apache-Modul) 478
log_errors (PHP-Option) 608
log_errors_max_len (PHP-Option) 608
log10() (PHP-Funktion) 748
LOG10E (JavaScript) 356
LOG2E (JavaScript) 356
Logdateien (Apache) 484
LogFormat (Apache-Direktive) 486
Logging-Einstellungen (Linux) 541
Logische Operatoren (JavaScript) 338
Logische Operatoren (PHP) 661
Logische Verlinkung 84
LogLevel (Apache-Direktive) 485
Lokale Variablen (JavaScript) 327
Lokale Variablen (PHP) 652, 672
longblob (MySQL) 785
longdesc (HTML-Attribut) 1060
Loopback-Adresse 438, 441
lost+found (Linux-Verzeichnis) 498
ls (Linux-Kommando) 496, 502
ltrim() (PHP-Funktion) 734
Lynx 31, 243

M

Macintosh 28
mail() (PHP-Funktion) 635f.
make (Linux-Kommando) 533
man (Linux-Kommando) 503
Mandrake Linux 492
Manpages 503
map (HTML-Element) 204, 1040
margin (CSS-Eigenschaft) 131, 1099
margin-bottom 1099
margin-bottom (CSS-Eigenschaft) 131
margin-left (CSS-Eigenschaft) 131, 1099
margin-right (CSS-Eigenschaft) 131, 1099
margin-top (CSS-Eigenschaft) 131, 1100
Marken und Patente 993
Markenrecht 1002
Markenschutz 1001
Markup 56
Markup-Korrektur 36
Markup-Sprachen 25
Maskierung (JavaScript) 332
Maßangaben (CSS) 117
match() (JavaScript) 359
Mathematische Symbole 1077
MathML 57, 861, 878
max() (JavaScript) 357
max() (MySQL-Funktion) 802
max() (PHP-Funktion) 748
max_execution_time (PHP-Option) 607
max_input_time (PHP-Option) 607
max-height (CSS-Eigenschaft) 144, 1100
MaxKeepAliveRequests (Apache-Direktive) 476
maxlength (HTML-Attribut) 216, 1060
max-width (CSS-Eigenschaft) 144, 1100
md5() (PHP-Funktion) 733
MdStV 993
media (HTML-Attribut) 1061
Media Player 210
Mediendienste 994
Mediendienstestaatsvertrag (MdStV) 993
Medientypen 298
mediumblob (MySQL) 785
mediumint (MySQL) 785
Mehrfenstertechnik (Frames) 277
Mehrspaltenindexe (MySQL) 806
Mehrzeilige Eingabebereiche 217
Meldungsfenster (JavaScript) 368
memory_get_usage() (PHP-Funktion) 770
memory_limits (PHP-Option) 607
Menüleisten 264
Messages (SOAP) 891
meta (HTML-Element) 79, 587, 1041
Meta-Angaben 78
Meta-Tags 78
method (HTML-Attribut) 212, 1061
method_exists() (PHP-Funktion) 769
Methoden (JavaScript) 351
Methoden (PHP-Objekte) 675
Microsoft Office Open XML 877
Midnight Commander 36
mime_magic_module (Apache-Modul) 478
mime_module (Apache-Modul) 478

Mime-Type 56
min() (JavaScript) 357
min() (MySQL-Funktion) 802
min() (PHP-Funktion) 748
min-height (CSS-Eigenschaft) 144, 1101
Minix 492
min-width (CSS-Eigenschaft) 144, 1101
mkdir (Linux-Kommando) 512
mkdir() (PHP-Funktion) 752
mktime() (PHP-Funktion) 615, 744
mm (CSS-Maßeinheit) 117
Modifikatoren (reguläre Ausdrücke) 703
Modulare DTDs 917
Module (Apache) 476
Monitor 27
more (Linux-Kommando) 518
MOVED PERMANENTLYNO CONTENT (HTTP-Statuscode) 466
MOVED TEMPORARILYNO CONTENT (HTTP-Statuscode) 466
MS Internet Explorer 31
MS Windows 27
Multimedia-Software 33
multiple (HTML-Attribut) 219, 1061
Multi-Tasking-Fähigkeit 490
Multi-User-Fähigkeit 490
MultiViews (Apache-Option) 480
mustUnderstand (SOAP) 894
mv (Linux-Kommando) 516
MyISAM 809
MySQL
  Installation 776
  Versionen 776
mysql.allow_persistent (PHP-Option) 825
mysql.connect_timeout (PHP-Option) 826
mysql.default_host (PHP-Option) 826
mysql.default_password (PHP-Option) 826
mysql.default_port (PHP-Option) 826
mysql.default_socket (PHP-Option) 826
mysql.default_user (PHP-Option) 826
mysql.max_links (PHP-Option) 826
mysql.max_persistent (PHP-Option) 826
mysql.trace_mode (PHP-Option) 826
mysql_affected_rows() (PHP-Funktion) 837
mysql_close() 829
mysql_connect() (PHP-Funktion) 828
mysql_errno() (PHP-Funktion) 838
mysql_error() (PHP-Funktion) 838
mysql_fetch_array() (PHP-Funktion) 830
mysql_fetch_assoc() 830
mysql_fetch_field() (PHP-Funktion) 834
mysql_fetch_object() (PHP-Funktion) 831
mysql_fetch_row() (PHP-Funktion) 830
mysql_install_db 779
mysql_num_rows() (PHP-Funktion) 832
mysql_query() (PHP-Funktion) 829
mysql_real_escape_string() (PHP-Funktion) 854
mysql_select_db() (PHP-Funktion) 838
mysqladmin 775
MySQL-Client 775, 778
  Batch-Betrieb 782
  starten und beenden 778
MySQL-Server 775
  starten und beenden 777
MySQL-Verbindung (PHP) 827

N

name (HTML-Attribut) 186, 206, 214f., 218, 282, 1061
name (JavaScript-Fenster) 371
Namenskonventionen (MySQL) 784
Namensräume (XML) 874
Namensraum 74
Namensraumnamen 876
Namensraum-Präfix 875
Namensrecht 1002
Namensschutz 1001
natsort() (PHP-Funktion) 739
Navigation (JavaScript) 408
Navigationsleiste 264
  horizontale 269, 273
  vertikale 264
navigator (JavaScript) 373
Negationsoperator (JavaScript) 340
Negative Feststellungsklage 1007
negotiation_module (Apache-Modul) 478
Netscape 33
Netscape 4.x 303
Neustart (Linux) 530
new (JavaScript) 351
new (PHP) 676
News-Breaks 1015
Newsfeed 878
Newsletter 1018
Newsticker 1015
Nicht interpretierte Inhalte (CDATA) 873
Nicht unterstrichene Links 194

Nicht-validierende Parser 864
nl2br() (PHP-Funktion) 732
NO CONTENT (HTTP-Statuscode) 466
nodeName (DOM) 381
nodeType (DOM) 381
nodeValue (DOM) 388
noframes (HTML-Element) 282
nohref (HTML-Attribut) 1062
Non-Scrolling-Region 284
noresize (HTML-Attribut) 284
noscript (HTML-Element) 1041
NOT FOUNDNO CONTENT (HTTP-Statuscode) 467
NOT MODIFIEDNO CONTENT (HTTP-Statuscode) 466
not null (SQL) 786
null (SQL) 800
Numerische Werte (CSS) 118
Nummerierte Listen 93

O

object (HTML-Element) 206, 210, 1041
Objekte (HTML) 209
Objekte (JavaScript) 351
  eigene 375
Objekte (PHP) 675
Objekteigenschaften (PHP) 675
Objekt-Handles 685, 689
Objektmethoden (PHP) 675
Objektorientierte DBMS 772
Objektorientierung (PHP5) 685
Objektstruktur (JavaScript) 350
Objektvariablen 651
octdec() (PHP-Funktion) 748
ODF 877
Öffnen von Dateien (PHP) 708
Oktalzahl (JavaScript) 333
ol (HTML-Element) 93, 1042
onblur (Event-Handler) 315
onblur (HTML-Attribut) 1062
onchange (Event-Handler) 315
onchange (HTML-Attribut) 1062
onclick (Event-Handler) 314
onclick (HTML-Attribut) 1062
ondblclick (Event-Handler) 314
ondblclick (HTML-Attribut) 1063
onfocus (Event-Handler) 315
onfocus (HTML-Attribut) 1063
onkeydown (Event-Handler) 315
onkeydown (HTML-Attribut) 1063
onkeypress (Event-Handler) 315
onkeypress (HTML-Attribut) 1063
onkeyup (Event-Handler) 315, 334
onkeyup (HTML-Attribut) 1064
onload (Event-Handler) 314
onload (HTML-Attribut) 1064
onmousedown (Event-Handler) 314
onmousedown (HTML-Attribut) 1064
onmousemove (Event-Handler) 315
onmousemove (HTML-Attribut) 1064
onmouseout (Event-Handler) 313, 315
onmouseout (HTML-Attribut) 1065
onmouseover (Event-Handler) 313f.
onmouseover (HTML-Attribut) 1065
onmouseup (Event-Handler) 314
onmouseup (HTML-Attribut) 1065
onreset (Event-Handler) 315
onreset (HTML-Attribut) 1065
onselect (Event-Handler) 315
onselect (HTML-Attribut) 1066
onsubmit (Event-Handler) 315
onsubmit (HTML-Attribut) 1066
onunload (Event-Handler) 314
onunload (HTML-Attribut) 1066
opacity (CSS-Eigenschaft) 154
opendir() (PHP-Funktion) 751
OpenDocumentFormat 877
OpenSSH 455
Opera 31
Operator (JavaScript), Negation 340
Operatoren (JavaScript) 333
  Logische 338
  Rechenoperatoren 337
  Vergleichsoperatoren 336
Operatoren (PHP) 659
  Bit-Operatoren 663
  Logische 661
  Rechenoperatoren 661
  Vergleichsoperatoren 660
  Zeichenkettenverknüpfung 662
  Zuweisung 659
optgroup (HTML-Element) 219, 1042
option (HTML-Element) 218, 1042
Optionen (Shell-Kommandos) 501
Options (Apache-Direktive) 479
order by (SQL) 798

orphans (CSS-Eigenschaft) 301, 1101
outline (CSS-Eigenschaft) 1101
outline-color (CSS-Eigenschaft) 1102
outline-style (CSS-Eigenschaft) 1102
outline-width (CSS-Eigenschaft) 1102
overflow (CSS-Eigenschaft) 143, 158, 288, 1103

P

p (HTML-Element) 91, 1043
padding (CSS-Eigenschaft) 161, 1103
padding-bottom (CSS-Eigenschaft) 161, 1103
padding-left (CSS-Eigenschaft) 136, 161, 1104
padding-right (CSS-Eigenschaft) 161, 1104
padding-top (CSS-Eigenschaft) 136, 161, 1104
page-break-after (CSS-Eigenschaft) 301, 1104
page-break-before (CSS-Eigenschaft) 301, 1105
page-break-inside (CSS-Eigenschaft) 301, 1105
Paletten 197
Paragraphen (HTML) 91
param (HTML-Element) 206, 208, 1043
Parameter (JavaScript) 347
Parameter (PHP) 672
Parameter (Shell-Kommandos) 501
Parameter (Shellscripts) 549
parseFloat() (JavaScript) 374
parseInt() (JavaScript) 403
Parser 56
passthru() (PHP-Funktion) 766
passwd (Datei) 542
passwd (Linux-Kommando) 505
Passwort (Linux) 505
Passwort (MySQL) 815
Patterns 695
pause (CSS-Eigenschaft) 1105
pause-after (CSS-Eigenschaft) 1106
pause-before (CSS-Eigenschaft) 1106
pc (CSS-Maßeinheit) 117
PCDATA 872f.
Perl 573, 577
Persistente Verbindungen 476
Persönlichkeitsrecht 993
Pfeilsymbole 1078
Pflichtenheft, geräteunabhängiges 45
Photoshop 33
PHP 595
  als Apache-Modul 601
  Download 598
  Formularverarbeitung 629
  Geschichte 597
  in HTML 612
  Installation 599
  Konfiguration 605
  MySQL-Unterstützung 825
  Versionen 598
php.ini 605, 770
PHP_INI_ALL 606
PHP_INI_PERDIR 605
PHP_INI_SYSTEM 605
PHP_INI_USER 605
phpinfo() (PHP-Funktion) 610
PHP-Scripts 603
PI (JavaScript) 356
Pipes 503
pitch (CSS-Eigenschaft) 1106
pitch-range (CSS-Eigenschaft) 1106
Pixelgrafik 197
platform (JavaScript) 373
Plattenspeicher 525
Platzhalterzeichen (reguläre Ausdrücke) 698
play-during (CSS-Eigenschaft) 1107
Plusminus-Zeichen 1073
PNG-Format 195
Podcast 878
Pointer (PHP) 674
pop() (JavaScript) 364
Pop-up- und Pop-under-Fenster 1014
Pop-up-Divs 1014
Pornografiegesetze 992
Portal 21, 228
Portalseite 21
Portnummern 462
position (CSS-Eigenschaft) 145, 285, 1107
Positionierung von Elementen 142
POST (Methode) 468
POST (PHP-Array) 655, 704
post_max_size (PHP-Option) 609
POST-Daten 565
Post-Inkrementierung 661
Potenzschreibweise (JavaScript) 333
pow() (JavaScript) 357

pow() (PHP-Funktion) 748
Präformatierter Text 94
pre (HTML-Element) 94, 1043
precision (PHP-Option) 606
preg_grep() (PHP-Funktion) 697
preg_match() (PHP-Funktion) 697, 706, 726
preg_match_all() (PHP-Funktion) 697, 726
preg_quote() (PHP-Funktion) 697
preg_replace() (PHP-Funktion) 628, 697, 729
preg_replace_callback() (PHP-Funktion) 697
preg_split() (PHP-Funktion) 697, 730
Pre-Inkrementierung 661
Pressearbeit 1008
primary key (SQL) 787
print (PHP-Konstrukt) 715
print() (JavaScript) 369
Print-Eigenschaften (CSS) 301
printf() (PHP-Funktion) 715
private (PHP) 687
proc (Linux-Verzeichnis) 498
Produkthaftungsgesetz 993
profile (Datei) 540
profile (HTML-Attribut) 1066
Programmablaufdiagramme 45
Programmausführung (PHP) 765
Programmierer 23
Programmiersprachen 25
Prompt (Shell) 499
prompt() (JavaScript) 369
protected (PHP) 688
Prozesse (Linux) 527, 529
Prozessnummer (PID) 529
Prüfsumme 733
PS1 (Umgebungsvariable) 499
Pseudoelemente 176
pt (CSS-Maßeinheit) 117
PUBLIC (Identifier) 870
public (PHP) 689
push() (JavaScript) 364
PuTTy 451
pwd (Linux-Kommando) 511
px (CSS-Maßeinheit) 117

Q

q (HTML-Element) 97, 1044
Quelltext (HTML) 58
Quirks Mode 304
quotes (CSS-Eigenschaft) 1107

R

Radiobuttons 213
Rahmen 134
Rahmen (Frames) 283
Rahmen (Tabellen) 103
Rahmendicke 101
random() (JavaScript) 357
Ranking 1010
Ranking-Algorithmus 1010
Rastergrafik 197
RDF 878
readdir() (PHP-Funktion) 751
README (Datei) 532
ReadmeName (Apache-Direktive) 565
readonly (HTML-Attribut) 1066
Real Validator 36
Realisierungsphase 46
Really Simple Syndication 878
Reboot (Linux) 530
Rechenoperatoren (JavaScript) 337
Rechenoperatoren (PHP) 661
Rechtestring (Linux) 523
Rechtsbereiche 992
Rechtslage 991
Recommendations (W3C) 49, 857
Red Hat Paket Manager (RPM) 530
Redhat Linux 492
Redirect (Apache-Direktive) 588
Referenzen (PHP) 674
referrer (JavaScript) 373
register_globals (PHP-Option) 609
Reguläre Ausdrücke 695, 801
rel (HTML-Attribut) 85, 191, 1067
Relationale DBMS 772
Relationen (Datenbanken) 819
Relative Adressierung 184
Relative Positionierung 144
Relaunch 1007
relay (SOAP) 894
reload() (JavaScript) 369
Remote Procedure Call 891
rename (SQL) 788
rename() (PHP-Funktion) 756
repair (SQL) 809
replace() (JavaScript) 359, 369
report_memleaks (PHP-Option) 608

REQUEST (PHP-Array) 655
Requests for Comments (RFCs) 51
require groups (Apache-Direktive) 584
require user (Apache-Direktive) 584
require() (PHP-Funktion) 653
require_once() (PHP-Funktion) 653
Reservierte Schlüsselwörter 327
Reset-Button 215
Resource Description Framework 878
RESPONSE-Kommando 466
Return-Werte (JavaScript) 347
Return-Werte (PHP) 672
rev (HTML-Attribut) 1067
reverse() (JavaScript) 364
revoke (SQL) 815
rewrite_module (Apache-Modul) 478
RFCs 51
RGB-Farbmodell 112
RGB-Modell 197
richness (CSS-Eigenschaft) 1108
right (CSS-Eigenschaft) 1108
rm (Linux-Kommando) 512, 517
rmdir (Linux-Kommando) 512
rmdir() (PHP-Funktion) 752
robots.txt 1011
role (SOAP) 894
Rollback 810
Rollover-Effekt 264
  vertikal 264
root (Benutzer, Linux) 505
root (MySQL) 812
Root-Server 446
Root-Verzeichnis 476
round() (JavaScript) 357
round() (PHP-Funktion) 748
Router 437
Routing 436
Routing-Tabellen 438
rows (HTML-Attribut) 217, 280, 1067
rowspan (HTML-Attribut) 107, 1067
RPC 891
rpm (Linux-Kommando) 531
RPM (Red Hat Paket Manager) 530
rsort() (PHP-Funktion) 738
RSS 878
RSS-Datei 883
RSS-Feed 968
RSS-Newsfeed 1018
rtrim() (PHP-Funktion) 734
Ruby-Text 72
Rückgabewerte (JavaScript) 347
Rückgabewerte (PHP) 672
rules (HTML-Attribut) 103, 1068
Runlevel-Definitionen 539

S

samp (HTML-Element) 97, 1044
SAX 864
sbin (Linux-Verzeichnis) 498
Scalable Vector Graphics 878
Scalable Vektor Graphics (SVG) 208
scandir() (PHP-Funktion) 752
Schaltflächen 213
scheme (HTML-Attribut) 1068
Schleifen (JavaScript)
  do...while 346
  for 344
  for in 345
  while 346
Schleifenkontrolle (JavaScript) 347
Schlüssel (MySQL) 787
Schriftart, Farbangaben 119
Schriftfarbe 122
Schriftformatierung, Farbangaben 118
Schriftgewicht 123
Schriftgröße 121
Schriftstil 123
scope (HTML-Attribut) 109, 1068
SCP 450
screen 373
script (HTML-Element) 316, 1044
ScriptAlias (Apache-Direktive) 575, 601, 603
Script-Einbindung (mit SSI) 572
Scriptsprachen 310
scrolling (HTML-Attribut) 284, 292
Search Engine Optimizers (SEOs) 1013
search() (JavaScript) 359
Secure Shell 450
Seitenaufteilung 228
Seitenlayouts 227
select (HTML-Element) 218, 1045
select (SQL) 780, 795
selected (HTML-Attribut) 219, 1068
Selektoren 167
Semantisches Web 859f.
serialize() (PHP-Funktion) 760
SERVER (PHP-Array) 655
Server Side Includes (SSI) 566
ServerAdmin (Apache-Direktive) 478

Server-Hosting 446
Server-Housing 447
ServerName (Apache-Direktive) 478
ServerRoot (Apache-Direktive) 476
Service-Client 890
Service-Server 890
SESSION (PHP-Array) 655, 759
session.cache_expire (PHP-Option) 610
session.name (PHP-Option) 610
session.save_handler (PHP-Option) 610
session.save_path (PHP-Option) 610
session.use_cookies (PHP-Option) 610
session.use_only_cookies (PHP-Option) 610
session.use_trans_sid (PHP-Option) 610
session_cache_expire() (PHP-Funktion) 764
session_cache_limiter() (PHP-Funktion) 763
session_destroy() (PHP-Funktion) 761
session_get_cookie_params() (PHP-Funktion) 762
session_id() (PHP-Funktion) 762
session_name() (PHP-Funktion) 759
session_start() (PHP-Funktion) 759
session_write_close() (PHP-Funktion) 763
Sessions (PHP) 756
set (MySQL) 786
set password (MySQL) 815
setAttribute() (DOM) 388
setAttributeNode() (DOM) 388
setcookie() (PHP-Funktion) 720
setDate (JavaScript) 353
setenvif_module (Apache-Modul) 478
setFullYear (JavaScript) 353
set-Funktionen (Setter) 688
setHours (JavaScript) 353
setMilliseconds (JavaScript) 353
setMinutes (JavaScript) 353
setMonth (JavaScript) 353
setSeconds (JavaScript) 353
setTime (JavaScript) 354
settype() (PHP-Funktion) 706
setUTC... (JavaScript) 354
SFTP 450
SGML 55
sha1() (PHP-Funktion) 733
shadow (Datei) 543
shape (HTML-Attribut) 204, 1068
Shebang-Zeile 547, 582
SHELL (Umgebungsvariable) 494
shell_exec() (PHP-Funktion) 765
Shell-History 500
Shell-Prompt 495, 499
Shells 494
Shellscripts 546
Shell-Variablen 501
shift() (JavaScript) 364
short_open_tag (PHP-Option) 606
show databases (SQL) 779
show tables (SQL) 780
shutdown (Linux-Kommando) 530
Sichtbarkeit 151
Simple Object Access Protocol 890
SimpleXML 945f.
simplexml_load_file() (PHP-Funktion) 948
simplexml_load_string() (PHP-Funktion) 948
sin() (JavaScript) 357
sin() (PHP-Funktion) 748
Single Source Publishing 922
Singleton-Elemente 75
Site-Verwaltung 991
size (HTML-Attribut) 218, 1069
Slashdot-Effekt 449
sleep() (PHP-Funktion) 717
slice() (JavaScript) 359, 365
small (HTML-Element) 98, 1045
smallint (MySQL) 786
SMTP (PHP-Option) 609
SOAP 890, 972
SOAP-Envelopes 890
SOAP-Faults 896
SOAP-Messages 890
Softwarepatente 1003
Softwareverwaltung 530
Sonderzeichen (in URIs) 189
Sonderzeichen (JavaScript) 332
sort (Linux-Kommando) 504
sort() (JavaScript) 365
sort() (PHP-Funktion) 738
Sortierung Datensätze (MySQL) 798
Spalten, Tabelle (HTML) 104
Spaltenbreite 157
span (HTML-Attribut) 1069
span (HTML-Element) 99, 1045
speak (CSS-Eigenschaft) 1109
speak-header (CSS-Eigenschaft) 1108
speak-numeral (CSS-Eigenschaft) 1109
speak-punctuation (CSS-Eigenschaft) 1109

speech-rate (CSS-Eigenschaft) 1109
splice() (JavaScript) 365
split() (JavaScript) 359
SpreadFirefox-Initiative 32
sprintf() (PHP-Funktion) 715
SQL 773
sql_big_result (SQL) 807
sql_buffer_result (SQL) 807
SQL-Injection 853
SQL-Statements 774
sqrt() (JavaScript) 357
sqrt() (PHP-Funktion) 748
SQRT1_2 (JavaScript) 356
SQRT2 (JavaScript) 356
src (HTML-Attribut) 201, 281, 322, 1069
SSH 450
SSI (Server Side Includes) 566
ssl_module (Apache-Modul) 478
Standalone-Tags 75, 865
Standardausgabe 714
Standard-Gateway 438
Standardkonformität 32
standby (HTML-Attribut) 1069
stat() (PHP-Funktion) 755
static (PHP) 691
Statische Klassenvariablen/
Funktionen 686
Statische Klassenvariablen/Methoden 691
Statische Webseiten 20
Statisches SQL 774
std() (MySQL-Funktion) 802
stdout 714
Steuerzeichen (JavaScript) 332
stop() (JavaScript) 369
str_ireplace() (PHP-Funktion) 728
str_replace() (PHP-Funktion) 728
Strafrecht 992
straight_join (SQL) 807
Streitwert 1006
stress (CSS-Eigenschaft) 1110
Strict (HTML-Variante) 70
String() (JavaScript) 374
Strings (JavaScript) 358
strip_tags() (PHP-Funktion) 731
stristr() (PHP-Funktion) 725
strlen (PHP-Funktion) 724
strlen() (PHP-Funktion) 707, 724
strong (HTML-Element) 97, 1046
strpos() (PHP-Funktion) 726
strrpos() (PHP-Funktion) 726
strrstr() (PHP-Funktion) 725
strstr() (PHP-Funktion) 725
strtolower() (PHP-Funktion) 734
strtoupper() (PHP-Funktion) 734
Structured Generalized Markup Language
(SGML) 55
Structured Query Language (SQL) 773
style (DOM) 390
style (HTML-Attribut) 109, 225, 1069
style (HTML-Element) 164, 1046
su (Linux-Kommando) 510
sub (HTML-Element) 99, 1046
Subdomains 442
Submit-Button 214
Subnetz-Maske 438
Sub-Shell 510
substr() (JavaScript) 359
substr() (PHP-Funktion) 724
Suchen, Dateien (Linux) 513
Suchen und Ersetzen (PHP) 728
Suchmaschinen 1008
Suchmaschinenoptimierung 1011
Suchmaschinen-Ranking 1010
Suchmaschinen-Robots 1011
Such-Robots 83
sum() (MySQL-Funktion) 802
summary (HTML-Attribut) 107, 1070
sup (HTML-Element) 99, 1047
Superglobals 654
Superglobals (PHP) 616
SUSE Linux 492
SVG 57, 208, 878
SVG-Dokument 865
Swift 3D 34
Swishmax 34
switch (JavaScript) 343
switch (PHP) 666
Symbole
diverse 1079
mathematische 1077
technische 1078
Symbolische Links (Linux) 520
SymLinksIfOwnerMatch (Apache-
Option) 480
syslog.conf 541
SYSTEM (Identifier) 870
system() (PHP-Funktion) 766
Systemauslastung 527
Systemkonfigurationsdateien 539
Systemüberwachung (Linux) 525

T

Tabellen 42, 100
Tabellen (MySQL)
  anlegen 783
  löschen 783
  reorganisieren 809
  Struktur ändern 787
Tabellenaußenrahmen 103
Tabellenbreite 104
Tabellenfuß 102
Tabellenkörper 102
Tabellenkopf 102
Tabellenlayouts 100
Tabellensortierung (JavaScript) 420
Tabellenspalten 104
Tabellenüberschrift 101
Tabellenzeile 102
tabindex (HTML-Attribut) 192, 1070
table (HTML-Element) 101, 1047
table-layout (CSS-Eigenschaft) 158, 1110
Tags 864
Tags (HTML) 60
tan() (JavaScript) 357
tan() (PHP-Funktion) 748
tar (Linux-Kommando) 519
target (HTML-Attribut) 283, 291
tbody (HTML-Element) 102, 1047
TCP/IP 439, 459
td (HTML-Element) 102, 1048
TDDSG 993
TDG 993
Technische Symbole 1078
Teilausdrücke (reguläre Ausdrücke) 701
Teilindexe (MySQL) 806
Teilzeichenketten (PHP) 724
Teledienste 994
  geschäftsmäßige 995
Teledienste-Datenschutzgesetz (TDDSG) 993
Teledienstegesetz (TDG) 993
Templates (XSLT) 927
test (Linux-Kommando) 554
Testphase, geräteunabhängiges 46
text-align (CSS-Eigenschaft) 132, 1110
textarea (HTML-Element) 217, 1048
text-decoration (CSS-Eigenschaft) 124, 1111
Textfarbe 122
Textfluss 202
text-indent (CSS-Eigenschaft) 134, 1111
Textrelevanz 1010
text-transform (CSS-Eigenschaft) 1111
tfoot (HTML-Element) 102, 1048
th (HTML-Element) 102, 1049
thead (HTML-Element) 102, 1049
this (JavaScript) 313, 377
throw (PHP) 694
ThumbsPlus 33
Ticker (JavaScript) 359
time (MySQL) 786
time() (PHP-Funktion) 742
TimeOut (Apache-Direktive) 476
timestamp (MySQL) 786
tinyblob (MySQL) 786
tinyint (MySQL) 786
Titel (HTML-Dokument) 77
title (HTML-Attribut) 85, 225, 1070
title (HTML-Element) 59, 77, 1049
title (JavaScript) 372
tmp (Linux-Verzeichnis) 498
toFixed() (JavaScript) 357
toGMTString() (JavaScript) 354
toLocalString() (JavaScript) 354
toLowerCase() (JavaScript) 359
top (CSS-Eigenschaft) 145, 1111
top (Linux-Kommando) 527
Top-Level-Domains 442
Total Commander 36
toUpperCase() (JavaScript) 359
tr (HTML-Element) 102, 1050
tracert 436
track_errors (PHP-Option) 608
track-vars (PHP-Option) 608
Traffic 448
Transaktion 811
Transaktionssicherheit 810
TransferLog (Apache-Direktive) 486
Transfervolumen 448
Transformation 862, 922
Transformationssprache 773
Transformationssprache XSLT 923
Transitional (HTML-Variante) 71
Transmission Control Protocol (TCP) 439, 459
Transparenz 151, 198
trim() (PHP-Funktion) 734
true (JavaScript) 330
Truecolor 197
try (PHP) 694
tt (HTML-Element) 98, 1050

Turingmaschine 25
type (HTML-Attribut) 164, 188, 206, 213f., 216, 316, 1070
Typumwandlung (JavaScript) 328
Typumwandlung (PHP) 651

U

uasort() (PHP-Funktion) 739
UDDI 900
Überschriften (HTML) 90
Überschriftennummerierung 292
Üble Nachrede 992
uid (Benutzer-ID) 507
uksort() (PHP-Funktion) 739
ul (HTML-Element) 92, 1050
umask (Linux-Kommando) 525
Umbenennen, Dateien (Linux) 516
Umgebungsfarbnamen 115
Umleitungen (Linux) 503
UNAURHORIZEDNO CONTENT (HTTP-Statuscode) 467
unicode-bidi (CSS-Eigenschaft) 1112
Unicode-System 63, 867, 872
Uniform Resource Locator (URL) 179
Uniform Resource Name (URN) 179
unique (SQL) 787
Universal Character Set (USC) 68
Universal Description, Discovery and Integration 900
Universal Resource Identifier (URI) 179
Unix 490
Unix-Derivate 490
unlink() (PHP-Funktion) 756
unserialize() (PHP-Funktion) 760
unset() (PHP-Funktion) 738
unshift() (JavaScript) 365
Unterlassungserklärung 1006
update (SQL) 792
upload_max_filesize (PHP-Option) 609
Urheberrecht 992, 997
Urheberschaft 998
URIs 179, 561
URL (JavaScript) 372
URL (JavaScript-Eigenschaft) 321
Usability 37, 40
USC Transformation Formats (UTF) 68
use (SQL) 780
UseCanonicalName (Apache-Direktive) 479
usemap (HTML-Attribut) 204, 1071
useradd (Linux-Kommando) 506
userAgent (JavaScript) 373
userdel (Linux-Kommando) 508
userdir_module (Apache-Modul) 478
usermod (Linux-Kommando) 507
users (Linux-Kommando) 509
usort() (PHP-Funktion) 739
usr (Linux-Verzeichnis) 498
UTF-8-Kodierung 67
UU-Enkodierung ((PHP) 734

V

Validator 36
Validierende Parser 864
Validierung 36
valign (HTML-Attribut) 1071
value (HTML-Attribut) 206, 215, 1071
valuetype (HTML-Attribut) 1071
var (HTML-Element) 97, 1051
var (Linux-Verzeichnis) 498
var (PHP) 676, 689
var_dump() (PHP-Funktion) 654
varchar (MySQL) 786
Variable Parameterzahl (PHP) 674
Variablen
  globale (JavaScript) 327
  globale (PHP) 652
  lokale (JavaScript) 327
  lokale (PHP) 652
Variablen (JavaScript) 326
Variablen (PHP) 644
  lokale 672
Variablen (Shell) 501
Variablen (Shellscripts) 550f.
Variablen-Debugging (PHP) 654
Variablendeklaration 326
Variablennamen (JavaScript) 327
Variablennamen (PHP) 644
variables_order (PHP-Option) 609
VBScript 310
Vektorgrafik 197
Vergleichsoperatoren (JavaScript) 336
Vergleichsoperatoren (PHP) 660
Vergleichsoperatoren (SQL) 799
Verleumdung 992
Verlinkung (logische) 84
Verschachtelung, HTML-Elemente 61
Verschieben, Dateien (Linux) 516
version (HTML-Attribut) 1071
vertical-align (CSS-Eigenschaft) 151, 161, 1112
Vertikale Navigationsleiste 264

Vertragsrecht 993
Verweis-sensitive Grafiken 203
Verzeichnis (Linux)
  anlegen 512
  löschen 512
  wechseln 511
Verzeichnisbaum (Linux) 496
Verzeichnis-Browsing 562f.
Verzeichnis-Management (PHP) 749
Verzeichnisschutz 583
Verzweigungen (Shellscripts) 553
vi-Editor 533
Viewport 126
vi-Kommandos
  Bewegen im Text 536
  Löschen von Text 538
  Modus wechseln 537
  Speichern und Beenden 536
  Suchen und Ersetzen 538
Virtual Hosting 445
Virtual Hosting (Apache) 482
Virtuelle Hosts (Apache) 481
visibility (CSS-Eigenschaft) 1112
voice-family (CSS-Eigenschaft) 1113
void() (JavaScript) 318
Volksverhetzung 992
volume (CSS-Eigenschaft) 1113
Volumenlimit 448
Vorvertragliche Widerrufsbelehrung 996

W

w (Linux-Kommando) 509
W3C 49
W3C-Recommendations 49
W3-Konsortium 49
Wahrnehmung 38
WAI 41
WAP 861
WDDX 901f., 982
WDDX-Dokumente 986
Web 2.0 859
Web Accessibility Initiative 41
Web Content Accessibility Guidelines 42
Web Distributed Data Exchange 901
Web Service Inspection Language 901
Web Services Description Language 898
Webalizer 457
Webanwendung 210
Webauftritt 21
Webbrowser 31
Webdesigner 22
Webentwickler 23
Webgrafiken 200
Web-Hosting 443
Weblog 878
Webmaster 24
Webpräsenz 21
Webprogrammierer 23
Webseite 20
Webseiten
  dynamische 20
  statische 20
Webservices 889
Websichere Farben 116, 1
Website 19
  Konzeptphase 44
  Realisierungsphase 46
  Wartung 47
Webspace 449
Weiterleitung (automatische) 586
Werbung 1013
Werk (Urheberschutz) 997
Wertzuweisungen, HTML-Attribute 62
where-Klausel 796f.
while (JavaScript) 346
while (Linux-Kommando) 557
while (PHP) 670
Whitespace 72
white-space (CSS-Eigenschaft) 1113
who (Linux-Kommando) 509
whoami (Linux-Kommando) 509
Whois-Service 442
Widerrufsbelehrung 996
widows (CSS-Eigenschaft) 301, 1114
width (CSS-Eigenschaft) 132, 143, 217, 1114
width (HTML-Attribut) 201, 206, 1072
width (JavaScript) 373
Wikipedia 1021
Wiki-Software 1023
Wiki-Systeme 1021
Wildcards (SQL) 800
Windows 27
Windows Media Player 210
Windows Scripting Host 310
WinSCP 453
WinZip 36
WML 861
Wohlgeformte Dokumente 868
word-spacing (CSS-Eigenschaft) 1114

write (Linux-Kommando) 509
write() (JavaScript) 321
WSDL 898, 978
WSIL 900
WYSIWYG -Editoren 29

X
XBRL 877
Xenu's Link Sleuth 36
XHTML 55ff., 878
  Dokumenttyp-Deklarationen 71
  JavaScript-Problematik 316
  Quelltext 76
  Syntax 73
XML 55, 311
xml_get_current_line_number() (PHP-Funktion) 965
xml_get_error_code() (PHP-Funktion) 965
xml_parse() (PHP-Funktion) 965
xml_parser_create() (PHP-Funktion) 963
xml_parser_free() (PHP-Funktion) 965
xml_parser_set_option() (PHP-Funktion) 963
xml_set_character_data_handler() (PHP-Funktion) 964
xml_set_element_handler() (PHP-Funktion) 964
XML-Deklaration 74, 865, 867
XML-Dokumente 864
XML-Editoren 862
XML-eigene Zeichen 873
XML-Klasse 960
XML-Parser 863, 870
XMLReader/xmlwriter 946
XML-Schema 892, 918
XML-Sprachen 861
XML-Versionen 867
XPath 924
XPath-Adressierung 942
XPath-Funktionen 924, 937
XSLT 57, 862, 923
XSLT-Elemente 929
XSLT-Funktionen (PHP) 966
XSLT-Prozessoren 925

Y
y2k_compliance (PHP-Option) 606
year (MySQL) 786

Z
Z Shell 494
Zahlen (MySQL) 791
Zahlen (PHP) 647
Zeichen (HTML) 63
Zeichenabstand 124
Zeichenformate 97
Zeichenketten (JavaScript) 332, 358
Zeichenketten (MySQL) 791
Zeichenketten (PHP) 645, 723
Zeichenkettenverknüpfung (PHP) 662
Zeichenkodierung 79, 867
Zeichenmaskierung (reguläre Ausdrücke) 699
Zeichensatz 79
  ISO-8859-1 65
  ISO-8859-15 66
Zeichenvorrat (XML) 872
Zeiger (PHP) 674
Zeilenhöhe 159
Zeilenumbruch 97
Zeit (JavaScript) 352
Zend Engine 597
Zentrale Formate (CSS) 163
Zerlegen (Zeichenketten, PHP) 729
Zielgruppe 40
z-index (CSS-Eigenschaft) 148, 1115
Zitat 95
Zivilrecht 992
Zombies (Linux) 528
Zugänglichkeit 41
Zugriffsarten (Linux) 521
Zugriffsrechte (Linux) 521, 523
Zuweisungsoperator (PHP) 659

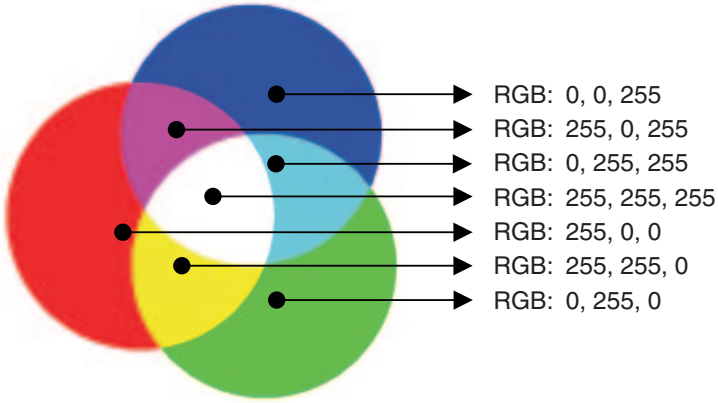
Farbtabellen

#000000	#000033	#000066	#000099	#0000CC	#0000FF
#330000	#330033	#330066	#330099	#3300CC	#3300FF
#660000	#660033	#660066	#660099	#6600CC	#6600FF
#990000	#990033	#990066	#990099	#9900CC	#9900FF
#CC0000	#CC0033	#CC0066	#CC0099	#CC00CC	#CC00FF
#FF0000	#FF0033	#FF0066	#FF0099	#FF00CC	#FF00FF
#003300	#003333	#003366	#003399	#0033CC	#0033FF
#333300	#333333	#333366	#333399	#3333CC	#3333FF
#663300	#663333	#663366	#663399	#6633CC	#6633FF
#993300	#993333	#993366	#993399	#9933CC	#9933FF
#CC3300	#CC3333	#CC3366	#CC3399	#CC33CC	#CC33FF
#FF3300	#FF3333	#FF3366	#FF3399	#FF33CC	#FF33FF
#006600	#006633	#006666	#006699	#0066CC	#0066FF
#336600	#336633	#336666	#336699	#3366CC	#3366FF
#666600	#666633	#666666	#666699	#6666CC	#6666FF
#996600	#996633	#996666	#996699	#9966CC	#9966FF
#CC6600	#CC6633	#CC6666	#CC6699	#CC66CC	#CC66FF
#FF6600	#FF6633	#FF6666	#FF6699	#FF66CC	#FF66FF
#009900	#009933	#009966	#009999	#0099CC	#0099FF
#339900	#339933	#339966	#339999	#3399CC	#3399FF
#669900	#669933	#669966	#669999	#6699CC	#6699FF
#999900	#999933	#999966	#999999	#9999CC	#9999FF
#CC9900	#CC9933	#CC9966	#CC9999	#CC99CC	#CC99FF
#FF9900	#FF9933	#FF9966	#FF9999	#FF99CC	#FF99FF
#00CC00	#00CC33	#00CC66	#00CC99	#00CCCC	#00CCFF
#33CC00	#33CC33	#33CC66	#33CC99	#33CCCC	#33CCFF
#66CC00	#66CC33	#66CC66	#66CC99	#66CCCC	#66CCFF
#99CC00	#99CC33	#99CC66	#99CC99	#99CCCC	#99CCFF
#CCCC00	#CCCC33	#CCCC66	#CCCC99	#CCCCCC	#CCCCFF
#FFCC00	#FFCC33	#FFCC66	#FFCC99	#FFCCCC	#FFCCFF
#00FF00	#00FF33	#00FF66	#00FF99	#00FFCC	#00FFFF
#33FF00	#33FF33	#33FF66	#33FF99	#33FFCC	#33FFFF
#66FF00	#66FF33	#66FF66	#66FF99	#66FFCC	#66FFFF
#99FF00	#99FF33	#99FF66	#99FF99	#99FFCC	#99FFFF
#CCFF00	#CCFF33	#CCFF66	#CCFF99	#CCFFCC	#CCFFFF
#FFFF00	#FFFF33	#FFFF66	#FFFF99	#FFFFCC	#FFFFFF

216 websichere Farben

Farbe	Farbname	Hexadezimal	RGB
	black	#000000	rgb(0, 0, 0)
	maroon	#800000	rgb(128, 0, 0)
	green	#008000	rgb(0, 128, 0)
	olive	#808000	rgb(0, 128, 128)
	navy	#000080	rgb(0, 0, 128)
	purple	#800080	rgb(128, 0, 128)
	teal	#008080	rgb(0, 128, 128)
	gray	#808080	rgb(128, 128, 128)
	silver	#C0C0C0	rgb(192, 192, 192)
	red	#FF0000	rgb(255, 0, 0)
	lime	#00FF00	rgb(0, 255, 0)
	yellow	#FFFF00	rgb(255, 255, 0)
	blue	#0000FF	rgb(0, 0, 255)
	fuchsia	#FF00FF	rgb(255, 0, 255)
	aqua	#00FFFF	rgb(0, 255, 255)
	white	#FFFFFF	rgb(255, 255, 255)
	orange	#FFA500	rgb(255, 165, 0)

Erlaubte Farbnamen



Mischfarben bei maximalen RGB-Farbanteilen

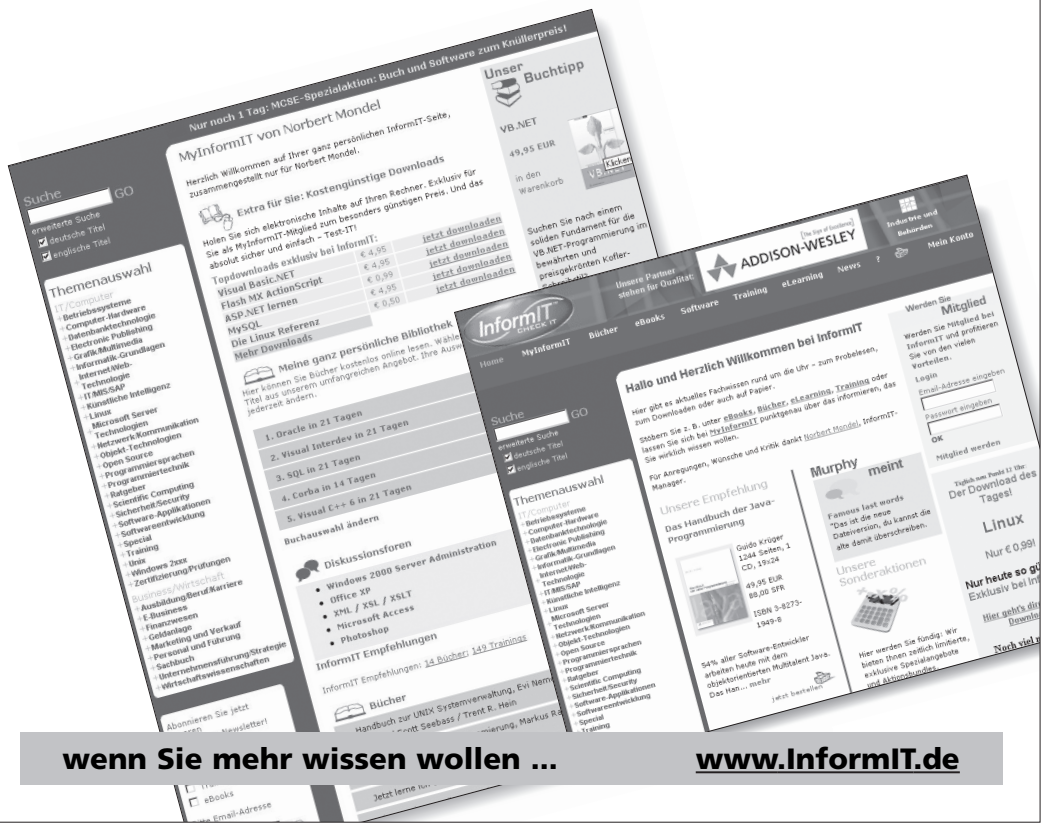
InformIT™
CHECK IT

... aktuelles Fachwissen rund
um die Uhr – zum Probelesen,
Downloaden oder auch auf Papier.

www.InformIT.de

InformIT.de, Partner von **Addison-Wesley**, ist unsere Antwort auf alle Fragen der IT-Branche.

In Zusammenarbeit mit den Top-Autoren von Addison-Wesley, absoluten Spezialisten ihres Fachgebiets, bieten wir Ihnen ständig hochinteressante, brandaktuelle Informationen und kompetente Lösungen zu nahezu allen IT-Themen.



wenn Sie mehr wissen wollen ... **www.InformIT.de**