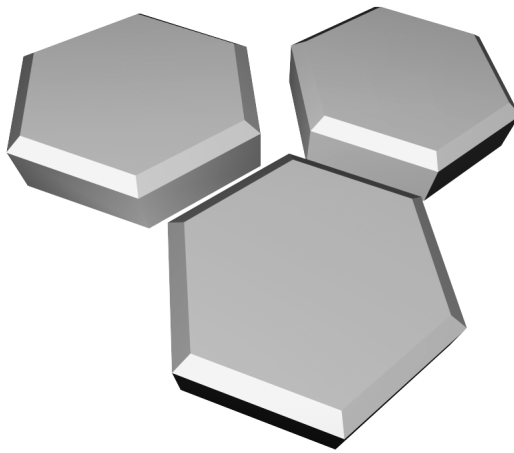


Das JavaScript Codebook



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, die in diesem Buch erwähnt werden, sind gleichzeitig auch eingetragene Warenzeichen oder sollten als solche betrachtet werden.

Umwelthinweis:

Dieses Buch wurde auf chlorfrei gebleichtem Papier gedruckt.

Die Einschrumpffolie – zum Schutz vor Verschmutzung – ist aus umweltverträglichem und recyclingfähigem PE-Material.

10 9 8 7 6 5 4 3 2 1
07 06 05

ISBN 3-8273-2162-X

© 2005 by Addison-Wesley Verlag,
ein Imprint der Pearson Education Deutschland GmbH,
Martin-Kollar-Straße 10–12, D-81829 München/Germany
Alle Rechte vorbehalten

Korrekturat: Sandra Gottmann, Bonn
Lektorat: Frank Eller, feller@pearson.de
Herstellung: Monika Weiher, mweiher@pearson.de
Satz: reemers publishing services gmbh, Krefeld – gesetzt aus der Minion
Umschlaggestaltung: Marco Lindenbeck, webwo GmbH (mlindenbeck@webwo.de)
Druck und Verarbeitung: Kösel, Krugzell (www.Koeselbuch.de)

Printed in Germany

Inhaltsverzeichnis

Vorwort	15
1 Über den Autor	16
2 Wozu ein Codebook?	16
 Teil I Einführung	 19
Einführung	21
1 Aufbau des Buches	21
2 Die Beispiele	22
3 Was ist JavaScript?	22
4 Der Aufbau von Skriptsprachen und die Rolle des Interpreters	24
5 Die Versionszyklen von JavaScript	25
6 HTML- bzw. XHTML-Grundlagen	26
7 JavaScript-Grundlagen	34
 Teil II Rezepte	 77
Grundlagen	79
1 Wie kann ich JavaScript in Webseiten einbinden?	79
2 Wie kann ich eine JavaScript-Version bei der Einbindung angeben?	84
3 Wie kann ich sicherstellen, dass ein Browser nur bekannte JavaScript-Anweisungen ausführt?	88
4 Wie kann ich testen, ob bei einem Browser JavaScript aktiviert ist?	90
5 Wie kann ich testen, welche JavaScript-Version von einem Browser unterstützt wird?	92
6 Wie kann ich den Browser eines Anwenders abfragen?	93
7 Wie kann ich die Version eines bekannten Browsers bei einem Anwender abfragen?	97
8 Wie kann ich die Spracheinstellung eines Browsers bei einem Anwender abfragen?	99
9 Wie kann ich die Bildschirmauflösung eines Besuchers ermitteln?	100
10 Wie kann ich die eingestellte Anzahl an Farben bei einem Besucher ermitteln?	102
11 Wie kann ich testen, ob bei einem Browser Java unterstützt wird?	104
12 Wie kann ich bestimmen, welche Plug-Ins ein Browser unterstützt?	105
13 Wie kann ich bestimmen, welche MIME-Typen ein Browser unterstützt?	109
14 Wie kann ich eine Browserweiche erstellen?	112

Core	125
15 Wie kann ich einen Test auf Unendlichkeit durchführen?	125
16 Wie kann ich testen, ob ein Ausdruck numerisch ist?	127
17 Festlegung des Variablen- oder eines Literaltyps bzw. des Funktionsrückgabetyps	129
18 Wie kann ich den Typ einer Variablen, eines Literals beziehungsweise den Rückgabewert einer Funktion bestimmen?	130
19 Wie kann ich den Typ einer Variablen bzw. den Rückgabewert einer Funktion gegen automatische Typkonvertierung schützen?	131
20 Wie kann ich eine Festlegung auf string realisieren?	134
21 Wie kann ich eine Festlegung auf number realisieren?	134
22 Wie kann ich eine Festlegung auf boolean realisieren?	137
23 Wie kann ich einen Wertebereich für eine Variable festlegen?	139
24 Wie kann ich testen, ob eine Variable definiert wurde?	140
25 Wie kann ich eine Variable als lokal festlegen?	142
26 Wie kann ich eine Variable als global beziehungsweise permanent festlegen?	145
27 Wie übergebe ich einer Funktion Werte?	145
28 Wie kann ich eine Funktion mit Defaultwerten für Parameter erstellen?	148
29 Wie kann ich bei einer Funktion einen Wert zurückgeben?	149
30 Wie erzeuge ich einen rekursiven Aufruf einer Funktion?	152
31 Wie erzeuge ich eine Objektinstanz?	155
32 Wie erzeuge ich in JavaScript ein eigenes Objekt?	156
33 Wie erweitere ich ein bestehendes Objekt beziehungsweise eine Klasse?	159
34 Wie kann ich den Inhalt von einem Objekt als Wert ausgeben?	164
35 Wie kann ich Zeichenketten kodieren und dekodieren?	165
36 Wie kann ich Sonderzeichen in Strings verwenden?	167
Allgemeines zu Formularen und Benutzereingaben	169
37 Wie kann ich ein Webformular mit HTML aufbauen?	170
38 Wie generiere ich einen Formularcontainer?	172
39 Wie kann ich ein einzeliges Eingabefeld realisieren?	181
40 Wie kann ich ein einzeliges Passwortfeld realisieren?	183
41 Wie kann ich eine einfache Formulschaltfläche realisieren?	184
42 Wie kann ich nur mit HTML eine Formulschaltfläche zum Abschicken von Formulardaten realisieren?	185
43 Wie kann ich rein mit HTML eine Grafik zum Abschicken von Formulardaten realisieren?	186
44 Wie kann ich nur mit HTML eine Formulschaltfläche zum Zurücksetzen von Formulardaten realisieren?	189
45 Wie kann ich rein mit HTML eine Grafik zum Zurücksetzen von Formulardaten realisieren?	190
46 Wie kann ich ein Kontrollfeld realisieren?	190
47 Wie kann ich eine Gruppe mit Optionsfeldern realisieren?	191
48 Wie kann ich rein mit HTML ein einzeliges Eingabefeld realisieren, in dem nur die Eingabe eines Kalenderdatums erlaubt ist?	192

49	Wie kann ich rein mit HTML ein einzeliges Eingabefeld realisieren, in dem nur die Eingabe von Dezimalkommazahlen gestattet ist?	193
50	Wie kann ich rein mit HTML ein einzeliges Eingabefeld realisieren, in dem nur die Eingabe von ganzen Zahlen gestattet ist?	193
51	Wie kann ich rein mit HTML ein einzeliges Eingabefeld realisieren, in dem nur die Eingabe eines URL gestattet ist?	194
52	Wie kann ich rein mit HTML ein einzeliges Eingabefeld realisieren, in dem nur Werte kleiner als ein vorgegebener Grenzwert gestattet sind?	194
53	Wie kann ich rein mit HTML ein einzeliges Eingabefeld realisieren, in das nur Werte größer als ein vorgegebener Grenzwert gestattet werden?	195
54	Wie kann ich ein Dateiauswahlfenster realisieren, das die ausgewählte Datei in das damit verbundene Eingabefeld übernimmt?	195
55	Wie kann ich ein verstecktes Formularfeld realisieren?	197
56	Wie kann ich ein mehrzeiliges Eingabefeld generieren?	198
57	Wie kann ich eine Auswahlliste mit einzeligem Listenfeld erstellen?	200
58	Wie kann ich eine Auswahlliste mit mehrzeiligem Listenfeld erstellen?	202
59	Wie kann ich eine Auswahlliste mit Mehrfachauswahl erstellen?	204
60	Wie kann ich einen Eintrag in einer Auswahlliste vorselektieren?	205
61	Wie kann ich ohne ein Webformular mit einem Besucher per JavaScript interagieren?	206
62	Wie greife ich mit JavaScript grundsätzlich auf ein Webformular zu?	211
63	Wie bestimme ich die Anzahl der Elemente in einem Formular?	214
64	Wie greife ich auf den Namen eines Formulars zu?	215
65	Wie greife ich auf die Versandmethode eines Formulars zu?	216
66	Wie greife ich auf die Zieladresse eines Formulars zu?	221
67	Wie greife ich auf das Ziel der Antwort auf ein Formular zu?	223
68	Wie greife ich auf die Kodierung eines Formulars zu?	227
69	Wie kann ich die Formulardaten ohne einen Submit-Button wegschicken?	229
70	Wie kann ich die Formulardaten ohne einen Reset-Button zurücksetzen?	234
71	Wie greife ich grundsätzlich auf die Formularelemente zu?	235
72	Wie greife ich auf den Wert eines Formularelements zu?	238
73	Wie greife ich auf den Namen eines Formularelements zu?	242
74	Wie erfolgt ein verkürzter Zugriff auf ein Formularelement über this?	243
75	Wie greife ich aus einem Formularelement auf das umgehende Formular zu?	245
76	Wie bestimme ich den Typ eines Formularelements?	247
77	Wie kann ich auf den Selektionszustand von einem Optionsfeld oder einem Kontrollkästchen zugreifen?	250
78	Wie greife ich auf die Elemente einer Auswahlliste in einem Listenfeld zu?	251
79	Wie greife ich auf den Wert eines Eintrags in einer Auswahlliste zu?	254
80	Wie greife ich auf die Beschriftung eines Eintrags in einer Auswahlliste zu?	256
81	Wie ermittle ich die Anzahl der Auswahlmöglichkeiten in einer Auswahlliste?	259
82	Wie kann ich testen oder festlegen, welcher Eintrag in einer Auswahlliste selektiert ist?	261
83	Wie kann ich mit JavaScript den Fokus von einem Formularelement nehmen?	264
84	Wie kann ich mit JavaScript einem Formularelement den Fokus geben?	266

85	Wie kann ich ein Formularelement selektieren bzw. deselektieren?	267
86	Wie kann ich mit JavaScript einen Klick eines Anwenders auf ein Formularelement simulieren?	276
87	Wie kann ich JavaScript-Funktionen aus Formularen heraus aufrufen?	277
88	Wie setze ich onMouseOver bei Formularelementen ein?	281
89	Wie setze ich onFocus bei Formularelementen ein?	282
90	Wie setze ich onBlur bei Formularelementen ein?	286
91	Wie setze ich onChange bei Formularelementen ein?	289
92	Wie kann ich auf das Zurücksetzen eines Formulars reagieren?	295
93	Wie kann ich mit JavaScript bei freier Texteingabe eine maximale Anzahl der einzugebenden Zeichen festlegen?	299
94	Wie ist die Kontrolle der Benutzereingabe beim unmittelbaren Eingeben eines Zeichens möglich?	300
95	Wie erfolgt die Kontrolle der Benutzereingabe beim Verlassen des Feldes beziehungsweise bei der Entgegennahme eines Rückgabewerts?	306
96	Wie erfolgt die Kontrolle der Benutzereingabe beim Verlassen beziehungsweise Absenden des kompletten Formulars?	313
97	Wie kann ich bei freier Texteingabe eine minimale Anzahl einzugebender Zeichen festlegen?	317
98	Wie erfolgt die Kontrolle der Benutzereingabe beim Verlassen des Feldes beziehungsweise der Entgegennahme eines Rückgabewerts?	317
99	Wie erfolgt die Kontrolle der Benutzereingabe beim Verlassen beziehungsweise Absenden des kompletten Formulars?	320
100	Wie kann ich bei freier Texteingabe eine maximale Anzahl an angezeigten Zeichen festlegen?	324
101	Wie kann ich mit JavaScript bei freier Texteingabe einen Vorgabewert festlegen?	334
102	Wie kann ich bei freier Texteingabe gewährleisten, dass nur Dezimalkommazahlen gestattet sind?	336
103	Wie kann ich bei freier Texteingabe gewährleisten, dass nur Ganzzahlen gestattet sind?	344
104	Wie kann ich bei freier Texteingabe sicherstellen, dass dort nur Werte kleiner als ein vorgegebener Grenzwert gestattet sind?	347
105	Wie kann ich sicherstellen, dass dort nur Werte größer als ein vorgegebener Grenzwert gestattet sind?	353
106	Wie kann ich mit JavaScript ein Pflichtfeld erzwingen?	355
107	Wie kann ich mit JavaScript bei freier Texteingabe einen bestimmten Inhalt erzwingen?	357
108	Wie kann ich eine gültige E-Mail-Adresse überprüfen?	359
109	Wie kann ich ein freies Eingabefeld realisieren, in dem nur die Eingabe eines URLs gestattet ist?	371
110	Wie kann ich ein Eingabefeld realisieren, in dem nur die Eingabe eines Kalenderdatums erlaubt ist?	380
111	Wie kann ich die Eingabe bestimmter Zeichen in einem freien Formularfeld verhindern?	387
112	Wie kann ich ein Webformular grundsätzlich plausibilisieren?	400

113	Welche Problemstellungen müssen beachtet werden?	401
114	Wie kann ich sicherstellen, dass ein Formular nur plausibilisiert abgeschickt wird?	439
Ausnahmebehandlung		443
115	Was ist eine Ausnahmen, und wie kann ich sie behandeln?	443
116	Wie kann ich mehrere Ausnahmen behandeln?	451
117	Wie kann ich selbst definierte Ausnahmen erzeugen und verwenden?	451
118	Wie kann ich freie Benutzereingaben mit dem Ausnahmekonzept absichern?	459
Datenfelder		465
119	Wie kann ich ein Datenfeld anlegen?	466
120	Wie greife ich auf ein Datenfeld zu?	468
121	Wie erfolgt ein anonymer Zugriff auf ein Datenfeld?	469
122	Wie greife ich auf die Elemente in einem Datenfeld zu?	469
123	Wie erfolgt der Zugriff über einen numerischen Index?	470
124	Wie erfolgt der Zugriff über einen String-Index? Assoziative Datenfelder	472
125	Wie erfolgt der Zugriff auf alle Elemente eines Datenfelds über for...in?	475
126	Wie kann ich die Datenfeldgröße bzw. die Anzahl der enthaltenen Elemente abfragen?	477
127	Wie kann ich die Größe eines Datenfelds festlegen?	482
128	Wie lege ich ein mehrdimensionales Datenfeld an und greife darauf zu?	484
129	Wie kann ich ein Datenfeld sortieren?	490
130	Wie kann ich die Sortierung eines Datenfelds umdrehen?	503
131	Wie kann ich die Inhalte eines Datenfelds zu einem String verbinden?	505
132	Wie kann ich ein Datenfeldelement aus dem Datenfeld entfernen?	506
133	Wie kann ich an das Ende eines Datenfelds ein oder mehrere Elemente anfügen?	514
134	Wie kann ich einen Teil eines Datenfelds extrahieren?	516
135	Wie kann ich zwei Datenfelder zusammenfügen?	519
136	Wie kann ich am Anfang eines Datenfelds Elemente einfügen?	521
137	Wie kann ich an einer beliebigen Stelle eines Datenfelds Elemente einfügen?	525
138	Wie ermittle ich den kleinsten oder den größten Wert in einem Datenfeld?	527
139	Wie kann ich den Inhalt eines Datenfelds zufällig mischen?	531
140	Wie kann ich eine Lottoziehung realisieren?	534
Stringmanipulation		539
141	Wie finde ich die Länge eines Strings heraus?	539
142	Wie kann ich die Buchstaben in einem String in Großbuchstaben konvertieren?	541
143	Wie kann ich die Buchstaben in einem String in Kleinbuchstaben konvertieren?	542
144	Wie ermittle ich das Zeichen, das in einem String an einer bestimmten Stelle steht?	543
145	Wie ermittle ich die Unicode-Kodierung von einem bestimmten Zeichen, das in einem String steht?	544
146	Wie verbinde ich zwei oder mehr Strings zu einem neuen String?	546
147	Wie kann ich aus einer Unicode-Sequenz ein Zeichen erstellen?	547
148	Wie kann ich in Strings HTML-Formatierungen vornehmen?	550

149	Wie kann ich das erste Vorkommen eines Zeichens oder einer Zeichenkette innerhalb eines Strings ermitteln?	556
150	Wie kann ich das letzte Vorkommen eines Zeichens oder einer Zeichenkette innerhalb eines Strings ermitteln?	558
151	Wie kann ich innerhalb eines Strings einen Text ersetzen?	559
152	Wie kann ich mit regulären Ausdrücken in Strings suchen und sie ersetzen?	560
153	Wie kann ich einen Teil eines Strings extrahieren (slice)?	591
154	Wie erfolgt die Extrahierung mit substr()?	592
155	Wie erfolgt die Extrahierung mit substring()?	593
156	Wie kann ich einen String an einem Trennzeichen in ein Datenfeld aufspalten?	594
Datumsoperationen		597
157	Wie kann ich ein Datumsobjekt mit dem aktuellen Systemdatum erzeugen?	597
158	Wie kann ich ein Datumsobjekt mit einem vorgegebenen Datum erzeugen?	599
159	Wie kann ich den Tag des Monats aus einem Datumsobjekt extrahieren?	602
160	Wie kann ich in einem existierenden Datumsobjekt den Monatstag ändern?	603
161	Wie kann ich den Wochentag des Objekts aus einem Datumsobjekt extrahieren?	604
162	Wie kann ich in einem existierenden Datumsobjekt den Wochentag ändern?	607
163	Wie kann ich die Stunden aus einem Datumsobjekt extrahieren?	607
164	Wie kann ich in einem existierenden Datumsobjekt die Stunden ändern?	608
165	Wie kann ich die Minuten der Uhrzeit aus einem Datumsobjekt extrahieren?	609
166	Wie kann ich in einem existierenden Datumsobjekt die Minuten ändern?	610
167	Wie kann ich die Sekunden der Uhrzeit aus einem Datumsobjekt extrahieren?	611
168	Wie kann ich in einem existierenden Datumsobjekt die Sekunden ändern?	611
169	Wie kann ich die Jahreszahl aus einem Datumsobjekt extrahieren?	612
170	Wie kann ich in einem existierenden Datumsobjekt die Jahreszahl ändern?	618
171	Wie kann ich den Monat aus einem Datumsobjekt extrahieren?	619
172	Wie kann ich in einem existierenden Datumsobjekt den Monat ändern?	622
173	Wie kann ich die Zeit, die seit dem 1.1.1970, 0:00:00 vergangen ist, extrahieren?	623
174	Wie kann ich in einem existierenden Datumsobjekt die Millisekunden ändern?	625
175	Wie kann ich den Unterschied zwischen der lokalen Zeit und Greenwich Meridian Time ermitteln?	626
176	Wie kann ich eine Zeit in den IETF-Standard umwandeln?	627
177	Wie kann ich eine Zeit in eine Zeichenkette mit lokaler Darstellung umwandeln?	628
178	Wie kann ich ein Schaltjahr bestimmen?	632
179	Wie kann ich mit Datumsangaben rechnen?	635
180	Wie kann ich die Differenz zwischen zwei Datumsangaben bestimmen?	635
181	Wie kann ich das Alter einer Person bestimmen?	635
182	Wie kann ich ein Ablaufdatum für den Inhalt einer Webseite festlegen?	642
183	Wie kann ich die Zeitspanne bis zu einem bestimmten Termin berechnen?	644
184	Wie kann ich mit Datumsobjekten Summenberechnungen durchführen?	646
185	Wie kann ich einem Anwender suggerieren, dass eine Webseite topaktuell ist?	649

Fenster und Dokumente	669
186 Wie kann ich den Inhalt einer Webseite dynamisch schreiben?	670
187 Wie kann ich in ein Dokument schreiben?	671
188 Wie kann ich ein Dokument explizit öffnen oder schließen?	675
189 Wie habe ich Zugriff auf Layoutinformationen in einer Webseite über document?	677
190 Wie kann ich auf alle Anker in einer Webseite zugreifen?	682
191 Wie kann ich auf Java-Applets in einer Webseite zugreifen?	684
192 Was ist LiveConnect, und wie kann ich es nutzen? Der direkte Java-Zugriff	687
193 Wie kann ich die Höhe und die Breite einer Webseite abfragen?	693
194 Wie kann ich den Termin der letzten Änderung einer Webseite abfragen?	695
195 Wie kann ich auf alle Hyperlinks in einer Webseite zugreifen?	696
196 Wie kann ich auf den vollständigen URL einer HTML-Datei zugreifen?	697
197 Wie kann ich auf alle Plug-Ins in einer Webseite zugreifen?	699
198 Wie kann ich den Namen eines Fensters bestimmen?	699
199 Wie kann ich auf den Titel einer HTML-Datei zugreifen?	700
200 Wie kann ich ermitteln, welchen Text ein Anwender im Dokument selektiert hat?	701
201 Wie kann ich ein einfaches Mitteilungsfenster anzeigen?	702
202 Wie kann ich ein Fenster zum Bestätigen einer Aktion anzeigen?	704
203 Wie kann ich mit einem Dialogfenster eine freie Benutzereingabe entgegennehmen?	705
204 Wie kann ich die Startseite des Browsers aufrufen?	706
205 Wie kann ich auf die Statuszeile des Browsers zugreifen?	707
206 Wie kann ich Standardanzeigen in der Statuszeile des Browsers ermitteln oder festlegen?	708
207 Wie kann ich eine Laufschrift in der Statuszeile des Webrowsers anzeigen?	710
208 Wie kann ich den Inhalt einer Webseite ausdrucken?	711
209 Wie kann ich den Ladevorgang einer Webseite abbrechen?	712
210 Wie kann ich eine Aktion zeitverzögert aufrufen?	714
211 Wie kann ich einen Aufruf von setTimeout() anhalten?	715
212 Wie kann ich auf die Adresszeile des Webrowsers zugreifen?	716
213 Wie kann ich mit JavaScript auf eine Seite weiterleiten?	718
214 Wie kann ich eine aktuelle Webseite neu laden?	718
215 Wie kann ich ein Browserfenster öffnen?	719
216 Wie kann ich beim Öffnen eines Fensters dessen Verhalten und Aussehen beeinflussen?	721
217 Wie kann ich ein Browserfenster dynamisch positionieren?	724
218 Wie kann ich ein Browserfenster dynamisch in der Größe verändern?	726
219 Wie kann ich den Inhalt eines Browserfensters dynamisch scrollen?	727
220 Wie kann ich ein Browserfenster schließen?	728
221 Wie kann ich überprüfen, ob ein zuvor geöffnetes Browserfenster noch offen ist?	729
222 Wie kann ich auf die History des Webrowsers zugreifen?	730
223 Wie kann ich die Anzahl der Einträge in der History eines Besuchers ermitteln?	730
224 Wie kann ich die zuletzt besuchte Seite eines Anwenders aufrufen?	731
225 Wie kann ich mit Cookies arbeiten?	732

226	Wie kann ich überprüfen, ob ein Anwender die Verwendung von Cookies aktiviert hat?	736
227	Wie kann ich den Wert einer Variablen zu einer anderen Webseite weitergeben?	736
228	Wie kann ich sicherstellen, dass eine Webseite nur mit Zugangsbeschränkung aufgerufen werden kann?	740
229	Wie kann ich mit Frames umgehen?	741
230	Wie erfolgen Verweise zu anderen Frames mit HTML?	744
231	Wie kann ich JavaScript auf Frames anwenden?	746
232	Wie kann ich gleichzeitig mehrere Frames aktualisieren?	747
233	Wie kann ich verhindern, dass meine Webseite in einem Frameset angezeigt wird?	753
Animation und DHTML		755
234	Was versteht man unter Dynamic-HTML?	755
235	Wie kann ich das all-Objekt verwenden?	756
236	Wie kann ich das style-Objekt einsetzen?	760
237	Wie kann ich das style-Objekt so einsetzen, dass es in allen neueren Browsern funktioniert?	766
238	Wie kann ich das style-Objekt in dem alten Netscape-Modell verwenden?	767
239	Wie kann ich dynamisch Bilder erzeugen und darauf zugreifen?	770
240	Wie kann ich die Höhe und die Breite eines Bildes dynamisch verändern?	772
241	Wie kann ich durch Setzen der src-Eigenschaft einen Rollover-Effekt erzielen?	775
242	Wie kann ich mit JavaScript und der src-Eigenschaft eine Animation realisieren?	777
243	Wie kann ich mit JavaScript durch Veränderung der Größe einer Grafik eine Animation realisieren?	781
244	Wie kann ich mit JavaScript dynamisch auf Style Sheets zugreifen?	783
245	Wie kann ich Style Sheets in eine Webseite einbinden?	783
246	Wie sieht eine CSS-Deklaration grundsätzlich aus?	786
247	Wie kann ich JavaScript auf Style Sheets anwenden?	789
248	Wie kann ich den Hover-Effekt als Rollover-Effekt einsetzen?	794
249	Wie kann ich Elemente einer Webseite frei positionieren?	795
250	Wie kann ich mit JavaScript und Style Sheets dynamisch positionieren?	797
251	Wie kann ich anspruchsvollere Animationen mit JavaScript und Style Sheets durchführen?	798
252	Wie kann ich eine Animation per Bildverschiebung realisieren?	799
253	Wie kann ich eine mit JavaScript generierte Animation erstellen?	803
254	Wie kann ich ein Element in Abhängigkeit von der Mausposition positionieren?	804
Ereignisbehandlung		807
255	Wie funktioniert Ereignisbehandlung grundsätzlich in JavaScript?	807
256	Wie kann ich beim Laden bzw. Verlassen der HTML-Datei eine Anweisung aufrufen?	808
257	Wie kann ich beim Laden einer Grafik eine Anweisung aufrufen?	810
258	Wie kann ich auf die Unterbrechung eines Bildladevorgangs reagieren?	810
259	Wie kann ich auf einen Klick auf eine Referenz reagieren?	811

260	Wie kann ich auf das Überstreichen einer Referenz mit dem Mauszeiger reagieren?	814
261	Wie kann ich auf Mausbewegungen reagieren?	815
262	Wie kann ich auf Tastaturereignisse reagieren?	816
263	Wie kann ich auf die Aktivierung und das Verlassen eines Elements reagieren?	816
264	Wie kann ich grundsätzlich auf Änderungen reagieren?	817
265	Wie erfolgt die Reaktion auf Zurücksetzen oder Absenden eines Formulars?	817
266	Wie erfolgt die Reaktion auf die Selektion von Text?	817
267	Wie kann ich einen Eventhandler explizit per JavaScript aufrufen?	817
268	Wie kann ich globale Ereignisbehandlung in JavaScript realisieren?	818
269	Wie kann man das event-Objekt auswerten und darauf reagieren?	819
270	Wie kann ich über das Netscape-Ereignismodell global auf Ereignisse reagieren?	820
271	Wie kann ich über das Microsoft-Ereignismodell global auf Ereignisse reagieren?	824

Stichwortverzeichnis**829**

Vorwort

Herzlich willkommen zur JavaScript-Programmierung. Mit JavaScript können Sie Leben und Aktivität in Ihre Webseiten bringen. JavaScript stellt die ideale Ergänzung zu HTML dar, um Webseiten zu dynamisieren, den Browser des Besuchers zu steuern 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 da 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.
- ▶ Sie bekommen 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 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 weitere Browserfenster (meist mit – nerviger – Werbung) zusätzlich angezeigt.
- ▶ In einer Framestruktur werden auf einen 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. JavaScript ist sicher nicht die Lösung für jedes Problem im Web, aber es bietet so viele interessante Möglichkei-

ten, dass eine Beschäftigung damit äußerst nützlich und vor allem spannend und interessant ist. Und das alles, ohne ein einziges Programm kaufen zu müssen. Dabei ist JavaScript dennoch einfach zu lernen 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 Language) hinausgehende Fähigkeiten benötigt, ein sehr heißer Tipp, wobei oft noch weitere Techniken wie Style Sheets hinzugenommen werden.

1 Über den Autor

In diesem Vorwort möchte ich mich Ihnen ebenfalls kurz mit ein paar wichtigen Eckdaten vorstellen. Ich bin Diplommathematiker 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 Programmierertätigkeit mit Turbo Pascal und C/C++ begonnen und damit Versicherungsprogramme für PCs erstellt. Seit 1995 schlage ich mich als Freelancer 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. Privat bin ich auf eine tägliche Kaffeedosis angewiesen (sonst bin ich ungenießbar), treibe viel Sport, treibe mein Motorrad oder Mountainbike durch die Taunus-Berge, spiele Saxophon in einer Rockband und verteidige täglich mein Büro gegen das Eindringen meiner Zwillinge. Außerdem stehe ich auf »Per Anhalter durch die Galaxis« und den Film »Blues Brothers«.

2 Wozu ein Codebook?

Was ist ein Codebook, und was ist besonders daran? Und wozu braucht man ein Codebook für JavaScript? Einerseits können mittlerweile sehr viele Leute mit JavaScript programmieren. Diese Skriptsprache ist unbestritten die am meisten eingesetzte Sprache zur Programmierung von clientseitigen Webaktionen. 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 bestimmte Funktionalität in einer Webseite als Ergänzung benötigt, die 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 im weiteren Sinn beschäftigen, diese Sprache. Der Einstieg in JavaScript ist auch alles andere als

schwer, und erste Beispiele und Projekte werden schnell zu Erfolgen führen. Aber aufgrund des oft nur seltenen Einsatzes von JavaScript kommen nur wenige JavaScript-Programmierer zu komplexeren Aufgaben, und vor allem fehlt die Praxis, um für spezifische Situationen eine funktionierende Lösung schnell greifbar zu haben. Ebenso gibt es zahlreiche Feinheiten im Umgang mit JavaScript, die zum einen auf den unendlichen Spezifika der unterschiedlichen Browser und zum anderen vielen Automatismen der Sprache selbst beruhen. Deshalb macht eine Rezeptsammlung wie in diesem Codebook für die gängigsten Fragestellungen gerade für JavaScript sehr viel Sinn. Das Buch hier ist deshalb kein Lehrbuch, um den Einstieg in JavaScript zu schaffen, sondern wendet sich an JavaScript-Programmierer mit erster Erfahrung, um diesen möglichst universell einsetzbare Hilfestellung bei konkreten Problemen aus der Praxis zu geben. Dazu umfasst das Buch ein breites Spektrum mit 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.

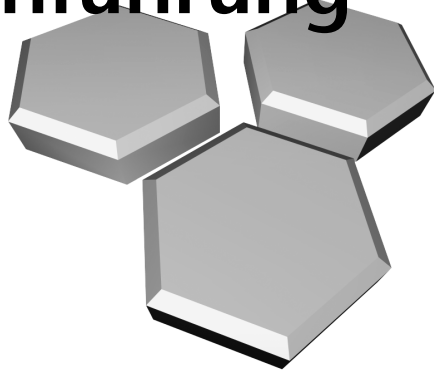
Jetzt aber genug der Vorrede. Los geht es mit JavaScript. Mir persönlich (und Ihnen hoffentlich auch) macht die Arbeit mit JavaScript neben der extremen Nützlichkeit im Rahmen einer Webseite noch Spaß, was ein sicher weiterer wichtiger Grund ist, diese Technologie zu lernen.

Ralph Steyer

www.rjs.de

TEIL I

Einführung



Einführung

1 Aufbau des Buches

Das Buch ist als Nachschlagewerk konzipiert, in dem Sie Lösungen für spezifische Aufgabenstellungen finden und kein Lehrbuch im klassischen Sinn mit einem didaktischen Aufbau. Die Rezepte, wie sie in Teil 2 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 durchgängig 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. Für die Klassifizierung der 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 Einschätzung selbstverständlich nicht universell gültig sein; und was für das eine Projekt unabdingbar ist, kann für ein anderes Webprojekt überflüssig sein. Dennoch soll die Wertung eine Anleitung darstellen, ob ein bestimmtes Rezept für Sie sinnvoll ist oder nicht. 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. Und obwohl JavaScript 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 gängigsten Browser Internet Explorer 6, Netscape Navigator 7.x (beziehungsweise die verwandten Mozilla 1.4 und Firefox), den Konqueror für die KDE 3.2 (Linux) und Opera 7.5. Ältere Browser finden bei Bedarf Beachtung. Explizit soll der auch heute noch hin und wieder verwendete Netscape Navigator 4.7 gelegentlich betrachtet werden. Als Referenzbetriebssysteme kommen Linux¹ und Windows XP Pro zum Einsatz.

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.

1. Konkret hauptsächlich SuSE 9.1, aber auch gelegentlich Mandrake 10.0 und Knoppix 3.4 (also Debian).

Aber auch im Fließtext werden bei Bedarf Begriffe so dargestellt, dass Sie Quellcodepassagen erkennen. Ebenso MENÜBEFEHLE, `Tasten`, *URLs* und noch einige weitere Besonderheiten. Diese Formatierungen werden konsequent verwendet. Und ebenso werden in dem Buch Bereiche verwendet, die über die Markierung mit verschiedenen Symbolen besondere Aufmerksamkeit erzeugen sollen.

Hinweis

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

Tipp

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

Achtung

Hier droht Gefahr.

2 Die Beispiele

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

3 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 sich als Prophet leichtsinnig zu versuchen – es wird auch in Zukunft so bleiben. HTML wird von einem Konsortium mit Namen *W3C* (<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 notwendig im Web. 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 Grafiken (bereits von Anfang an im Web als Vervollständigung von dem reinen Text auf Basis von HTML dabei) und geht über Multimediatechniken wie Flash (ein proprietäres – d.h. herstel-

lerabhängiges – Programm beziehungsweise Format zum Erzeugen von Animationen im Rahmen einer Webseite) hin zu Style Sheets und echten Programmiertechniken in Webseiten.

Zu den wichtigsten Erweiterungen von HTML zählen die *Skriptsprachen*, von denen es zahlreiche Varianten gibt. JavaScript ist eine der gewichtigsten Ergänzungstechnologie zu HTML und eine solche Skriptsprache für das Internet. Diese Sprache ist an Java angelehnt, offen, plattformunabhängig und leicht zu erlernen. Sie wurde Mitte der 90er-Jahre von der Firma Netscape (<http://www.netscape.com>) 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 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 *LiveScript* und letztendlich *JavaScript* genannt, um eine Anlehnung an *Java* zu verdeutlichen (denn das war der Name der aus Oak hervorgegangenen Sun-Technologie).

Achtung

Aber Vorsicht! Die Ähnlichkeit im Namen zeigt eine gewisse Verwandtschaft zwischen JavaScript und Java an. Jedoch ist diese Verwandtschaft weitläufiger, als es Laien erscheint. Java ist viel leistungsfähiger als jede Skriptsprache, aber auch viel komplizierter zu lernen. Die Sprache ist sowohl von der Syntax als auch dem Konzept 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 ganze Entwicklungsplattform, die zwar einen Schwerpunkt im Bereich Internet/Intranet hat (dabei kommen so genannte Java-Applets auf Seiten des Clients oder aber Java Servlets bzw. JSP – Java Server Pages – 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. Viele Aktionen im Bereich des Online-Bankings, des elektronischen Zahlungsverkehrs oder sonstiger sensibler Datenkommunikation beruhen auf Java. Mit Java lassen sich vollständige Programme entwickeln (im Gegensatz zu Skripten, die nur bereits fertige Programme steuern und nutzen). Sowohl in Form von Java-Applets, die mit HTML-Anweisungen in eine Webseite integriert werden und auch nur dort lauffähig sind, als auch als vollständig eigenständige Applikationen, die im Gegensatz zu den Applets keinen Container wie einen Browser mehr benötigen.

Eine direkte Konkurrenz zu JavaScript ist jedoch *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 nur der Internet Explorer VBScript. Zwar ist die Technologie dort auch leistungsfähiger als JavaScript, aber da dies mit erheblichen Sicherheitsrisiken einhergeht, ist das eher von Nachteil. Bedeutung hat VBScript vor allem bei der serverseitigen Programmierung. Auf Seiten von Microsoft-Webservern ist VBScript die Haussprache für die ASP-Technologie (*Active Server Pages*). ASP kann freilich ebenso mit JavaScript realisiert werden.

Seit geraumer Zeit ist auf Serverseite jedoch hauptsächlich *PHP* sehr populär. *PHP* ist sehr einfach zu lernen und hochspezialisiert auf bestimmte Vorgänge im Rahmen von Servervorgängen wie Formularauswertungen, Datenbankzugriffen oder Dateizugriffe auf dem Serverrechner.

Neben JavaScript fällt oft der Name *JScript*. Bei *JScript* handelt es sich um einen Ableger von JavaScript, welcher 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-Clone 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. 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 neuen Browsern sind die Probleme weitgehend beseitigt).

4 Der Aufbau von Skriptsprachen und die Rolle des Interpreters

Wie schon erwähnt (und sicher bekannt), bestehen Webseiten im Wesentlichen aus *Klartext* (HTML, 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-daten wie Grafiken, Videos, Animationen etc. hinzu. Das ist die optische Darstellung, die jeder Anwender in Form von Textformatierungen, Absatzformatierungen, Grafikdarstellungen etc. wahrnimmt. Für die Umsetzung der HTML-Anweisungen ist der HTML-Interpreter vom Browser 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 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 Bestandteil 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 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 auf einem Webserver, stellt dieser einen passenden Interpreter bereit. Im Fall von JavaScripts beim Anwender ist es der Webbrowser, der einen internen JavaScript-Interpreter bereitstellt.

JavaScript als Sprache beinhaltet einen *internen Befehlssatz*, eine *Syntax* und eine definierte *Struktur*, mit denen diverse Objekte rund um den Browser kontrolliert werden können. JavaScript enthält sowohl 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, Java-Applets und anderen Objekten verwendet werden. JavaScript ist jedoch nur eine *objektbasierende*, aber keine *objektorientierte* Sprache, denn zu einer echten objektorientierten Sprache fehlen einige zentrale Merkmale (etwa die Vererbung). Eigentlich ist es so, dass clientseitige Skriptsprachen im Wesentlichen den relativ dummen Browser intelligenter machen. Und 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 deaktivieren kann.

5 Die Versionszyklen von JavaScript

JavaScript ist seit seiner ersten Vorstellung im Jahre 1995 durch mehrere Versionszyklen gegangen. Grundsätzlich muss jedoch 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 vollständig Sprachzyklus verdaut hatten! Das gilt bis heute, obwohl die Entwicklung von JavaScript derzeit nicht sonderlich schnell voranschreitet. Aber zurück zur geschichtlichen Entwicklung. 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-Clone 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 kompatibel zu der ECMA-262-Norm. 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 wurde der JavaScript-Standard 1.4 voll kompatibel mit dem ECMA-262-Standard designt. War er deshalb ein durchschlagender Erfolg? Das Gegenteil ist der Fall. Die Version 1.4 kann man nur als Zwischenversion betrachten, denn sie wurde von kaum einem Browser richtig umgesetzt. Genau genommen richteten sich die nachfolgenden Browser nach der ECMAScript Edition 3 (obwohl die meisten diesen nur sukzessive umsetzen), die im Dezember 1999 freigegeben wurde. Die dritte Version des ECMAScript-Standards entspricht dem derzeit immer noch im Web maximal verfügbaren JavaScript 1.5 und beinhaltet als Erweiterung reguläre Ausdrücke, wie sie aus Perl etabliert sind, verbesserte Behandlung von Strings, neue Kontrollanweisungen, ein `try/catch`-Excep-

tion-Handling, wie es aus Java bekannt ist², erweiterte Fehlerdefinitionen, Formatierungen für numerische Ausgaben und einige weitere kleinere Erweiterungen im Rahmen der Internationalisierung.

Und obwohl JavaScript 1.5 mit seinem Geburtsdatum im letzten Jahrtausend sicher kein Frischling mehr ist, kann man auch heute sogar nur die Version 1.3 als allgemein im Web akzeptierten Standard betrachten. Das soll bedeuten, den Standard 1.3 verstehen nahezu alle heute noch von Anwendern verwendete Browser. Bei Techniken aus Nachfolgeversionen gibt es immer noch bestimmte Konstellationen zwischen JavaScript-Anweisungen und Browsern, wo es Probleme geben kann. Ein paar explizite Beispiele mit Browsern sollen das belegen. In der ersten Hälfte 2000 kam mit dem Navigator 6 der erste Browser heraus, der offiziell JavaScript 1.5 implementiert³. Der zeitgleich aktuelle Internet Explorer 5.5 beinhaltete eine JScript-Version, die ungefähr JavaScript 1.4 entsprach. Der Internet Explorer 5.5 unterstützt darüber hinaus bereits einige Features von JavaScript 1.5, weshalb er eigentlich JavaScript 1.5 näher ist. Dennoch kann man mit einfachen Beispielen zeigen, dass der Internet Explorer selbst in der Version 6 noch Skript-Container ignoriert, die mit der Versionsangabe 1.4 oder 1.5 gekennzeichnet sind, obwohl er die meisten erweiterten Syntaxstrukturen verstehen würde. Andere Browser wie Opera 6 oder Mozilla 1 (die ja auch nicht gerade uralt sind) verstehen ebenso Teile von JavaScript 1.5 nicht vollständig. Tests auf verschiedenen Browser sind leider – insbesondere bei Anweisungen jenseits von JavaScript 1.3 – deshalb unumgänglich. Aber glücklicherweise nehmen die schwerwiegenden Probleme in den neuen Browsern rapide ab.

6 HTML- bzw. XHTML-Grundlagen

Die Einbindung von JavaScript in eine Webseite gehört nicht zu JavaScript, sondern betrifft ausschließlich HTML. Die Verbindung von JavaScript mit einer Webseite wird ausschließlich mit HTML erledigt. Das ist wichtig, denn es muss klar sein, wann bei der Bearbeitung einer Webseite der HTML-Interpreter aktiv ist und wann der JavaScript-Interpreter. Die Tatsache hat wichtige Konsequenzen. So ist *Groß- und Kleinschreibung* in purem HTML *irrelevant*, in JavaScript spielt sie jedoch eine entscheidende Rolle. Der Abschnitt hier kann sogar noch allgemeiner verstanden werden, denn anhand von JavaScript kann die Einbindung von beliebigen Skriptelementen in Webseiten exemplarisch behandelt werden. Auch andere Skriptelemente werden sich vollkommen analog zu JavaScript in eine Webseite einbinden lassen. Es gibt verschiedene Techniken, wie Skripte mit einer Webseite verbunden werden können. Bevor diese behandelt werden, sollen die wichtigsten HTML-Grundlagen besprochen werden.

-
2. Im Grunde nur eine Etablierung des Ausnahmebehandlungskonzeptes auf JavaScript 1.4.
 3. Auf der einen Seite bot der Navigator 6 eine gute und stabile Unterstützung für JavaScript 1.5 respektive die ECMA-262-Norm. Auf der anderen Seite ist dieser Browser in sehr vielen wichtigen Details nicht mehr zu den Standards kompatibel, die in den Vorgängerversionen des Navigators zentrale Bestandteile waren. Zwar waren diese nicht vollständig standardisiert, aber das sind Elemente von anderen Browsern auch nicht. Die vollständige Wegnahme der Unterstützung von Features, die mit Netscape Navigator 4.x eingeführt wurden (etwa `document.layers` oder `document.embeds`), führen in, auf diesen älteren Browsern optimierten, Skripten zu erheblichen Problemen, wenn diese in der Nachfolgeversion geladen werden.

Eine aus HTML aufgebaute Webseite beinhaltet nur zwei grundsätzlich zu unterscheidende Strukturen:

- Steueranweisungen
- reiner Text

Steueranweisungen sind die HTML-Befehle. Alle HTML-Steueranweisungen werden in so genannten *Tags* angegeben, die von spitzen Klammern – dem »Kleiner«- und dem »Größer«-Zeichen – begrenzt werden. Ein HTML-Tag sieht also von der Struktur her immer so aus:

```
<xyz>
```

Listing 1: Das Schema eines HTML-Tags

Alles, was kein Tag ist (d.h., es ist nicht von einem »Kleiner«- und einem »Größer«-Zeichen eingeschlossen), wird beim Laden einer Webseite von einem Browser als reiner Text verstanden. Das ist nicht so trivial, wie es im ersten Moment klingt. Es hat fatale Auswirkungen. Ein großes Problem beim Interpreterkonzept (das ja zur Darstellung von HTML-Dokumenten zum Tragen kommt) ist das schnelle Veralten der Interpreter. Wenn in einer Interpretersprache ein oder mehrere neue Befehle hinzugefügt werden, können die bis dahin entwickelten Interpreter diese Anweisung(en) naturgemäß noch nicht kennen und entsprechend nicht ausführen. Anwender, die ein Dokument mit solch neuen Anweisungen in einem davor veröffentlichten Interpreter laden, bekommen Schwierigkeiten. Da HTML über die Zeit immer weiter entwickelt und vor allem durch browserspezifische Zusatzanweisungen angereichert wurde, gibt es dementsprechend eine Vielzahl von Dialekten, und nahezu kein Browser kann alle Anweisungen sämtlicher Dialekte vollständig. Was soll aber geschehen, wenn ein Browser in einer HTML-Seite eine Anweisung findet, die er nicht versteht? Abstürzen, wie es normalerweise Windows-Programme tun, wenn sie an einer unglücklichen Programmanweisung Schluckauf bekommen? Oder eine Fehlermeldung bringen, mit der üblicherweise kein Anwender etwas anfangen kann?

Es gibt noch eine dritte Lösung – ignorieren! Und das ist, obwohl es erst einmal nicht besonders positiv erscheinen mag, zumindest bei der Beschreibung von Dokumenten im Web die beste der drei Varianten. Das Ignorieren von unbekannten Anweisungen durch den Browser basiert auf dem *Prinzip der Fehlertoleranz*, welches zu den Eckdaten von HTML gehört. Dieses Prinzip veranlasst die Programme zur Auswertung von HTML-Dokumenten, sowohl fehlende Strukturen quasi unsichtbar im Hintergrund zu ergänzen, wenn diese offensichtlich fehlen und eindeutig ergänzt werden können⁴, als auch bei der Interpretation einer HTML-Datei so fehlertolerant wie irgend möglich zu sein. Der äußerst positive Effekt ist, dass dann auch syntaktisch unkorrekte Dokumente so weit wie möglich ausgewertet werden können. Soweit Browser korrekte HTML-Anweisungen vorfinden, werden diese Anweisungen ausgeführt und angezeigt. Falsche oder unvollständige Anweisungen werden ganz einfach ignoriert. Und alles, was sich außerhalb eines Tags befindet, wird als reiner Text im Anzeigebereich des Browsers dargestellt.

4. Etwa ein fehlendes Grundgerüst in einer Webseite.

Das Prinzip der Fehlertoleranz ist unumgängliche Voraussetzung dafür, dass immer neue HTML-Anweisungen oder neue Technologien im Internet eingeführt werden können (das betrifft natürlich auch JavaScript) und dennoch die älteren Browser beim Laden solcher Webseiten nicht »abschmieren«. Was sie nicht kennen, wird einfach ignoriert. Der Rest wird dargestellt. Eine Abwärtskompatibilität der HTML-Dokumente ist also immer sichergestellt, auch wenn im schlimmsten Fall signifikante Informationen beim Anzeigen verloren gehen können. Dummerweise kann man bei Skript- und Programmiersprachen nicht so tolerant verfahren, da viele Schritte aufeinander aufbauen.

Es gibt nun in HTML Tags, die zwingend einen Beginn- und einen Ende-Befehl benötigen. Beide zusammen bilden einen so genannten *Container*, der den Bereich für die Wirkung einer HTML-Anweisung festlegt, etwa von wo bis wo ein Text fett dargestellt werden soll oder wo der Beginn oder das Ende einer Tabelle ist. Bei Tags nennt man das öffnende Tag das *Einleitungs-* und das den Container beschließende Tag das *Abschluss-Tag*. Abschluss-Tags sind bis auf einen der öffnenden Klammer folgenden Slash (/) identisch mit dem Einleitungs-Tag. Der Abschluss-Tag zum fiktiven Einleitungs-Tag `<xyz>` würde wie folgt aussehen: `</xyz>`

Beispiele:

```
<b> ... </b>
<h6> ... </h6>
<script> ... </script>
```

Listing 2: Verschiedene HTML-Container

Es gibt neben Containern in der HTML-Syntax auch Tags, die nur als Einleitungs-Tag vorkommen, beispielsweise ein Zeilenumbruch, der durch `<br>` auch ohne Ende-Tag vollständig beschrieben ist. Oder auch die Einbindung von Grafiken mit dem Tag `<img>`. Solche einzelnen Tags werden da verwendet, wo eine Wirkung nicht explizit beschränkt werden muss.

Bei HTML-Steueranweisung spielt es im Gegensatz zu den Befehlsanweisungen vieler anderer Sprachen keine Rolle, ob sie groß- oder kleingeschrieben werden (bei JavaScript wird das jedoch eine Rolle spielen). Die Anweisung `<h4>` bewirkt das Gleiche wie `<H4>`. Auch eine unterschiedliche Groß- und Kleinschreibung im Einleitungs- und Abschluss-Tag oder sogar innerhalb einer Anweisung hat keine negativen Auswirkungen⁵, erhöht jedoch nicht gerade die Lesbarkeit.

Achtung

Diese letzte Aussage als auch der Verzicht auf einen Abschluss-Tag bei einigen Befehlen gilt nicht für *XHTML*, worauf gleich noch kurz eingegangen wird.

Viele Tags sind erst dann sinnvoll einzusetzen, wenn sie genauer spezifiziert werden. Nicht jeder, denn ein Zeilenumbruch ist immer durch die Anweisung `<br>` vollständig beschrieben.

5. Das wäre eine gültige, wenngleich nicht sonderlich glücklich gewählte Syntax: `<CeNTeR>`

Dann gibt es Anweisungen, die mit oder ohne Parameter funktionieren. Etwa der `<body>`-Tag, bei dem Sie optional beispielsweise Hintergrundfarbe und Textfarbe der Webseite angeben können. Wenn Parameter angegeben werden, werden bestimmte Aspekte genauer spezifiziert. Fehlen sie, werden Grundeinstellungen verwendet. Aber nicht alle Anweisungen sind so eindeutig oder flexibel. So gibt es beispielsweise die Steueranweisung `<font>`, mit der die Bedingungen für die Wahl der Schrift in der Webseite angegeben werden. Aber ohne genauere Angaben ist die Anweisung sinnlos. Die Schriftgröße, die Schriftart oder die Schriftfarbe sind genauere Spezifikationen, welche durch so genannte *Parameter* erfolgen. Diese werden bei Bedarf den einleitenden Tag – durch Leerzeichen abgetrennt – erweitern. Dabei 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 – dem Gleichheitszeichen (=) – den entsprechenden Wert zugeordnet. Dies kann ein Text oder eine Zahl sein oder aber auch andere Dinge (etwa eine Internet-Adresse). Meist werden die Werte in Hochkommata eingeschlossen, aber das ist in HTML⁶ so gut wie nie notwendig⁷. Ein Tag mit einem Parameter mit einer Wertzuweisung sieht schematisch so aus:

```
<Anweisung Parameter = Wert>
```

Beispiele:

```
<font color=blue>  

```

Listing 3: HTML-Tags mit Parameter

Parameter, die bereits einen Wert repräsentieren, brauchen bloß durch ein Leerzeichen 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:

```
<Anweisung Parameter>
```

Listing 4: Schema für einen Parameter ohne Wertzuweisung

6. In XHTML schon.

7. Eine Ausnahme ist eine Bildreferenz mit dem `<img>`-Tag, bei der der URL der Grafik in Hochkommata eingeschlossen werden muss. Allgemein können Sie auf die Hochkommata nicht verzichten, wenn der zugewiesene Wert solche Zeichen enthält, die in HTML als Steuerzeichen behandelt werden (also Leerzeichen, der Slash etc.).

Beispiel:

```
<table border>
```

Listing 5: Ein Parameter ohne Wertzuweisung – diese Anweisung setzt einen Tabellenrahmen mit einer Dicke, die der Browser festlegt.

Achtung

Es wird immer nur der einleitende Tag durch Parameter erweitert. Beim beendenden Tag wird *niemals* ein Parameter angegeben!

Viele Befehle lassen sich über mehr als einen Parameter spezifizieren. Diese werden dann einfach durch Leerzeichen getrennt aufgelistet. *Dabei spielt die Reihenfolge der Parameter keine Rolle.* Beispiele:

```
<body bgcolor=white text=red>
```

Listing 6: Der `<body>`-Tag mit zwei Parametern

```
<body text=red bgcolor=white>
```

Listing 7: Der `<body>`-Tag mit zwei Parametern in umgekehrter Reihenfolge hat die vollkommen gleiche Wirkung.

Das Grundgerüst einer Webseite

Da HTML eine äußerst fehlertolerante Beschreibungssprache ist, werden Webseiten selbst in Situationen, in denen in anderen Sprachen geschriebene Dokumente oder Programmstrukturen einen Fehler oder einen Programmabbruch auslösen würden, trotzdem oft noch brauchbare Resultate liefern. Reine HTML-Seiten sind Klartextdateien mit der Dateiendung `.htm`, `.html` oder `.shtml` (bis auf Sonderfälle). Alleine diese Endungen lassen das Dokument zu einem HTML-Dokument werden. Für eine HTML-Seite gibt es jedoch dennoch wie für jedes andere Dokument oder andere Programmiersprache immer ein Grundgerüst und gewisse Grundregeln.

Eine korrekt aufgebaute HTML-Seite wird immer in die Anweisung `<html>` am Anfang und `</html>` am Ende eingeschlossen. Die beiden Anweisungen bilden immer das äußere Gerüst einer HTML-Seite. Davor dürfen höchstens Kommentarzeilen stehen, die natürlich auch überall im Inneren einer HTML-Seite verwendet werden können. Die Steuerzeichen für Kommentarzeilen sind ein Ausrufezeichen und zwei Striche am Anfang und zwei Striche am Ende des Tags. Jeder in diesem Tag stehende Text wird vom interpretierenden Browser als Kommentar betrachtet. Ein Kommentar kann über mehrere Zeilen gehen und natürlich ebenfalls im Inneren einer HTML-Seite verwendet werden. Auch ein Kommentar-Tag muss von spitzen Klammern eingeschlossen werden, also sehen die Zeichenfolgen so aus:

```
<!--  
...  
-->
```

Listing 8: Ein HTML-Kommentar

Das weitere Grundgerüst einer HTML-Datei besteht grundsätzlich aus folgenden zwei Teilen:

- ▶ dem (optionalen) Header (Kopf)
- ▶ dem Body (Körper)

Nach der einleitenden HTML-Anweisung steht in einer Webseite normalerweise ein Header-Teil, das heißt ein Kopfteil, in dem die allgemeinen Hintergrundinformationen über eine Webseite notiert werden, beispielsweise der Titel oder so genannte Metainformationen, etwa Copyright-Informationen und vor allem wichtige Angaben für Suchmaschinen. Dies erfolgt mittels des `<meta>`-Tags, der auf verschiedenste Art und Weise verwendet werden kann. Oft finden Sie `<meta>`-Tags in Verbindung mit den Attributen `name` und `content`. Über `name` wird eine Eigenschaft angegeben (etwa der Autor, Schlüsselwörter, aufgrund denen die Webseite in einer Suchmaschine gefunden werden soll, oder eine Beschreibung der Webseite), über `content` der Wert dieser damit benannten Variablen. Diese beiden Informationen bilden zusammen ein Wertepaar, das auch nur in Verbindung Sinn macht.

Grundsätzlich wird ein Header mit dem Tag `<head>` begonnen und mit `</head>` entsprechend wieder geschlossen. Die eigentlichen Daten, die der WWW-Browser einem Anwender auf dem Bildschirm anzeigen soll, werden in den Body geschrieben. Die Tags `<body>` und `</body>` umschließen den Anzeigebereich einer Webseite. In dem Body ist der eigentliche Text mit Überschriften, Verweisen, Grafikreferenzen und auch die Referenz für Java-Applets zu notieren. Das vollständige Grundgerüst einer normalen HTML-Datei sieht also schematisch immer so aus.

```
<!-- Kommentar -->  
<html>  
  <head>  
    <title>...</title>  
    <meta ...>  
    ...  
    <meta ...>  
  </head>  
  <body>  
    Überschriften, Text,  
    Verweise, Grafiken, Java-Applet-Referenz usw.  
  </body>  
</html>
```

Listing 9: Ein schematisches Grundgerüst einer Webseite

HTML versus XHTML

HTML ist wie JavaScript durch verschiedene Sprachzyklen gegangen und liegt derzeit in der Version 4 vor. Diese wurde bereits 1997 (!) verabschiedet. Es ist ein Phänomen, dass sich eine dermaßen populäre und wichtige Technologie über so viele Jahre nicht weiter entwickelt hat. In der gesamten EDV der letzten Jahrzehnte werden Sie – vielleicht mit Ausnahme der Diskette – keine Technologie finden, die sich dermaßen lange auf einem Stand gehalten hat und dennoch nicht »tot« ist. Erklärbar ist dieser Entwicklungsstillstand nur, weil sich die Mitglieder des W3-Konsortiums teils spinnefeind sind und gegenseitig nach allen Regeln der Kunst zu blockieren versuchen.

Andererseits stimmt es auch nicht ganz, dass HTML nicht vorangeht. Im Grunde gibt es bereits geraume Zeit einen Nachfolger von HTML 4, der aber nicht mehr HTML, sondern *XHTML* (*Extensible HyperText Markup Language* = erweiterbare Hypertext-Auszeichnungssprache) genannt wird. HTML wurde ursprünglich auf Basis von SGML (*Standard Generalized Markup Language* – <http://www.w3.org/MarkUp/SGML/>) entwickelt. SGML ist eine bereits seit den Sechzigerjahren verwendete Beschreibungssprache zur Darstellung von Inhalten auf unterschiedlichen Plattformen. Diese Metasprache erlaubt das Definieren von Auszeichnungssprachen mit Hilfe so genannter *DTD-Anweisungen* (*Document Type Definitions* = Dokumenttypdefinitionen). In den DTD-Anweisungen wird festgelegt, welche Elemente zu einer Auszeichnungssprache gehören und welche jeweils dazugehörenden Attribute. Dazu kommen noch Regeln, welche Elemente innerhalb welcher anderen Elemente vorkommen können und andere Beziehungen wie die Notwendigkeit von Abschluss-Tags etc. Aus SGML ist nun in jüngster Vergangenheit XML (*Extensible Markup Language* = erweiterbare Auszeichnungssprache) hervorgegangen. XML gestattet wie SGML die Definition von Auszeichnungssprachen mit Hilfe von DTD-Anweisungen.

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, die aber einen neuen Namen und eine neue Versionsverwaltung erhielt. Seit Januar 2000 gibt es XHTML 1.0, und das entspricht im Grunde HTML 4.0 (dem aktuellen HTML-Standard). 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 Stylesheet-Sprachen, die eine gemeinsame syntaktische Grundlage auf der Basis von XML erhalten können.

XHTML hat sich in der Web-Community jedoch nie durchgesetzt. Der Grund ist banal – der Vorteil von XHTML ist, syntaktisch bedeutend strenger als HTML zu sein und dem Wildwuchs nichtstandardisierter Anweisungen entgegenzusteuern, die die Interpretation von Webseiten in verschiedenen Browsern so unterschiedlich werden lassen⁸. Aber genau diesen Sinn und Zweck von XHTML kann man dem gemeinen Webdesigner oder auch Hobby-Webseitenersteller kaum begreiflich machen, wenn dieser bisher bestimmte Dinge tun durfte, die plötzlich verboten sein sollen. Und da die Masse der Webseitenersteller nicht pro-

8. Streng genommen kann XHTML also weniger als HTML.

fessionell ist⁹, werden sich Browser-Hersteller hüten, kein HTML mehr zu unterstützen und die strengen XHTML-Regeln zu fordern. Damit dreht sich die Entwicklung im Kreis. Die Webseitenersteller sind das Prinzip der Fehlertoleranz und die laxen HTML-Regeln gewohnt und erstellen Webseiten, die sich darauf verlassen. Die Browserhersteller trauen sich nicht, strenges XHTML zu fordern, und dementsprechend sehen nur wenige Webseitenersteller ein, warum sie sich viel Arbeit machen sollen, um ein maximal gleichwertiges Resultat zu erreichen, das sie auch einfacher bekommen.

Zwar gibt es in XHTML nahezu die gleichen Elemente, Attribute und Verschachtelungsregeln wie in HTML, und auch die Unterstützung von XHTML ist in den neuesten Browsern gewährleistet. Aber es gibt insoweit Unterschiede, dass bestimmte Dinge gegenüber HTML erweitert, während syntaktisch Regeln verschärft wurden. Hier sind einige Unterschiede:

- Zu dem unter HTML üblichen *Mime-Typ* (Multipurpose Internet Mail Extensions) `text/html` gibt es für XHTML-Dokumente noch `text/xml` oder `application/xml`. MIME bedeutet einen Internet-Standard zur Spezifizierung von Dateitypen bei der Kommunikation zwischen Servern 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. Aufgrund seiner Liste mit MIME-Typen kann der Browser eine Datei eines bekannten Typs korrekt behandeln. Werden unbekannte Dateitypen geladen, kann das Programm nicht direkt damit umgehen, sondern muss die Datei speichern oder über zusätzliche Informationsquellen (etwa 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). MIME-Typen geben eindeutig das zu verwendende Programm an. 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`. Die meisten Skripteinbindungen verzichten auf diese Angabe, da Browser und Server so gut wie immer auch so zurecht kommen. Wir werden in diesem Buch ebenfalls weitgehend darauf verzichten.

- Die Dateierweiterungen spielen in XML-fähigen Browsern eine differenzierte Rolle, als es bei nicht-XML-fähigen Browsern der Fall ist. Je nach Systemkonfiguration kann der Browser verschiedene Interpreter bei den Endungen `.htm` beziehungsweise `.html` und `.xhtml` einsetzen. Im ersten Fall wird unter Umständen der HTML-Interpreter verwendet, im zweiten der XML-Interpreter. In der Praxis heißt das, beim HTML-Interpreter greift das Prinzip der Fehlertoleranz, und syntaktisch nicht ganz fehlerfreie Dokumente werden dennoch so weit wie möglich angezeigt, während im zweiten Fall die Darstellung mit einer Fehlermeldung abgebrochen wird.

9. Das ist nicht negativ gemeint, sondern trägt der Tatsache Rechnung, dass sehr viele Leute Webseiten privat erstellen.

- ▶ Jede XHTML-Datei fordert als erste Anweisung eine so genannte *XML-Deklaration*, etwa `<?xml version="1.0" encoding="UTF-8"?>`. HTML kennt zwar die verwandte `<doctype>`-Anweisung, aber die ist nicht zwingend.
- ▶ Das einleitende `<html>`-Tag besitzt in HTML normalerweise keine Attribute, während man in XHTML einen Namensraum angeben kann. Etwa so: `<html xmlns="http://www.w3.org/1999/xhtml">`.
- ▶ XHTML erzwingt ein strengeres Einhalten des HTML-Grundgerüsts (inklusive Header und Body).
- ▶ In HTML spielt Groß- und Kleinschreibung grundsätzlich keine Rolle. XHTML und XML unterscheiden strikt zwischen Groß- und Kleinschreibung. In XHTML sollen alle Elemente und Attribute kleingeschrieben werden.
- ▶ HTML-Elemente ohne Abschluss-Tag (etwa `<br>` oder `<hr>`) müssen in XHTML mit der Zeichenfolge `<br />` am Ende gekennzeichnet werden. Vor dem Schrägstrich sollte ein Leerzeichen stehen (etwa `<br />`). Alternative ist die Notation von einem Anfangs- und End-Tag ohne irgendetwas (auch kein Leerzeichen) dazwischen (etwa `<br></br>`).
- ▶ In XHTML müssen alle Elemente, die in HTML einen optionalen Abschluss-Tag haben (etwa `<td>` oder `<option>`), mit Abschluss-Tag notiert werden.
- ▶ Attribute ohne Wertzuweisung, die es in HTML gibt, gibt es in XHTML nicht. Allen Attributen muss ein Wert zugewiesen werden.
- ▶ HTML und XHTML unterscheiden sich bezüglich der Behandlung von Skript- und Style-Containern. In HTML wird der Inhalt der Container 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 werden. In XHTML ist das nicht möglich. Relativ triviale Dinge wie Vergleiche auf größer oder kleiner werden deshalb ein Problem. Obwohl XHTML deshalb einen Befehl bietet, bestimmte Bereiche in dem Skript-Container so zu behandeln, wie es unter HTML üblich ist (das nennt man *CDATA*), kann das wiederum für JavaScript-Interpreter zu einem Problem werden. Die Auslagerung von dem Skript in eine externe Datei ist beim Zusammenspiel zwischen XHTML und Skript- beziehungsweise Style-Anweisungen die sinnvollste Lösung.

7 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 interessieren¹⁰.

Hinweis

Die nachfolgenden zentralen Aspekte von JavaScript werden im Rahmen der Rezepte in diesem Buch ausführlich verwendet und dabei tiefer gehend besprochen, als es in diesem Überblick der Fall ist.

Schlüsselwörter

Jede Programmier- und Skriptsprache besitzt bestimmte Zeichenkombinationen, die ein essenzieller Teil der Sprachdefinition sind. Diese lassen sich in verschiedene Kategorien unterteilen. Das sind einmal Wörter, die bereits eine feste Bedeutung haben, so genannte *Schlüsselwörter*, bei deren Auftreten der JavaScript-Interpreter automatisch vorgegebene Aktionen ausführen wird. Etwa unter JavaScript der Begriff *var*, welches bei der Variablen-deklaration verwendet wird. Oder *function* zur Einleitung einer Funktionsdeklaration. 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 reserviert, aber auch bestimmte noch nicht verwendete Zeichenkombinationen für zukünftige Sprachvarianten. Die erste nachfolgende Tabelle enthält alphabetisch die verwendeten Schlüsselwörter von JavaScript 1.3. Diese Aufstellung ist umso wichtiger, weil nur dieser Standard in absehbarer Zeit im Web vorausgesetzt werden kann.

break	case	continue	default	delete	do
else	export	false	for	function	if
import	in	new	return	switch	this
true	typeof	var	void	while	with

Tabelle 1: Die verwendeten Schlüsselwörter von JavaScript 1.3

10. Ein Beispiel zur *if*-Anweisung oder zur Verwendung des Vergleichsoperators dürfte kaum dazu zählen.

Beachten Sie bei der Auflistung, dass streng genommen die Token `true` und `false` keine Schlüsselwörter, sondern Boole'sche Literale sind. Sie sind nichtsdestotrotz gegenüber einer andersartigen Verwendung in Skripten gesperrt und besitzen selbstverständlich eine Funktion. Die Schlüsselwörter `import` (Objekte oder Funktionen eines fremden Skripts importieren) und `export` (Objekte oder Funktionen für fremde Skripts ausführbar machen) zählen nicht zum offiziellen Standard 1.3 (auch nicht 1.5) und werden nur von wenigen Browsern unterstützt.

In JavaScript 1.3 sind noch die folgenden Schlüsselwörter reserviert, aber ohne konkrete Bedeutung:

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

Tabelle 2: Die reservierten, aber nicht verwendeten Schlüsselwörter von JavaScript 1.3

Die nächste Tabelle enthält alphabetisch die Schlüsselwörter von JavaScript 1.4/1.5 beziehungsweise der zugrunde liegenden ECMA-262-Norm, d.h. ECMAScript Version 3, die bereits im Einsatz sind. Beachten Sie die Schlüsselwörter `catch`, `finally`, `instanceof`, `throw` und `try`, die gegenüber JavaScript 1.3 neu hinzugekommen sind (dort waren sie nur reserviert, aber ohne Bedeutung), und dass `import` und `export` hier explizit nicht angegeben sind (sie sind offiziell – wie schon gesagt – nur reserviert):

Schlüsselwort	Beschreibung
<code>break</code>	Abbruch in Schleifen
<code>case</code>	Fallunterscheidungen
<code>catch</code>	Neu in JavaScript 1.4/1.5. Ein Schlüsselwort, das im Rahmen des Ausnahmebehandlungskonzepts von JavaScript eingesetzt wird.
<code>continue</code>	Fortsetzung in Schleifen
<code>default</code>	Fallunterscheidungen
<code>delete</code>	Löschen eines Datenfeld-Elementes oder einer selbst definierten Objekteigenschaft
<code>do</code>	Beginn einer Erweiterung der while-Schleife (do-while)
<code>else</code>	Einleitung des alternativen Blocks in einer if-Schleife
<code>false</code>	Der Wert falsch

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

Schlüsselwort	Beschreibung
finally	Neu in JavaScript 1.4/1.5. Ein Schlüsselwort, das im Rahmen des Ausnahmebehandlungskonzepts von JavaScript eingesetzt wird.
for	Einleitung von for-Schleifen
function	Einleitung von Funktionen
if	Einleitung von if-Fallunterscheidungen
in	Bedingte Anweisungen in if-Fallunterscheidungen
instanceof	Neu in JavaScript 1.4. Testet, was für ein Typ ein Objekt ist
new	Definition von Objekten
return	Übergabe eines Rückgabewertes in Funktionen
switch	Fallunterscheidung
this	Bezug auf die aktuelle Instanz eines Objekts
throw	Neu in JavaScript 1.4/1.5. Ein Schlüsselwort, das im Rahmen des Ausnahmebehandlungskonzepts von JavaScript eingesetzt wird.
true	Der Wert wahr
try	Neu in JavaScript 1.4/1.5. Ein Schlüsselwort, das im Rahmen des Ausnahmebehandlungskonzepts von JavaScript eingesetzt wird.
typeof	Typ eines Elements
var	Definition einer Variablen
void	Leerer Funktionstyp
while	Einleitung einer while-Schleife
with	Erlaubt mehrere Anweisungen mit einem Objekt durchzuführen

Tabelle 3: 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 der 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 (offiziell ist es immer noch nur reserviert) wurde. 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 4: Die reservierten, aber nicht verwendeten Schlüsselwörter von ECMAScript Version 3 beziehungsweise JavaScript 1.5

Variablen, Datentypen und Literale

Ein *Literal* ist in einer Programmiertechnik eine Darstellung eines eindeutig festgelegten und unveränderlichen Werts, etwa die Ganzzahl 42 oder die Gleitkommazahl 3.14 oder der Text (String) "Trillion". Ein Literal ist also ein Wert, der entweder selbst verwendet oder einer Variablen zugewiesen werden kann. *Variablen* gehören zu den wichtigsten Charakteristika einer Programmiertechnik. 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. JavaScript unterscheidet aber streng 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 im Hauptspeicher an, welche Art von Information in diesem gespeichert werden kann und wie mit diesem bei einer Operation zu verfahren ist. Gerade Letzteres muss sinnvoll geregelt werden. Was gibt $5 * 6$? Was ergibt $3 + 3.14$? Das ist sicher klar. Zumindest 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 – analog wie in den meisten Skriptsprachen, aber abweichend von der Vorgehensweise bei strengeren Programmiersprachen wie Java – bei der Deklaration (das bedeutet die erste Einführung) einer Variablen nicht explizit über ein Schlüsselwort ein Datentyp festgelegt. Eine Festlegung des Datentyps erfolgt implizit durch die Art von Wert, die der Variablen irgendwann zugewiesen wird. Bei einer erneuten Zuweisung mit einem andersartigen Wert verändert sich sogar der Datentyp der Variable während der Laufzeit des Skripts. Die Datentypen werden also intern von JavaScript verwaltet.

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. Nehmen wir eine Variable, die einen Datentyp hat, der ein Byte (8 Bit) groß ist. In einem Byte können genau $2 \text{ hoch } 8 = 256$ 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 Hauptdatentypen und zwei Sonderdatentypen, die aber – wie gerade besprochen – in JavaScript implizit verwaltet werden¹¹. JavaScript unterstützt folgende vier

11. Genau genommen ist es so, dass bei Zuweisung eines anderen Datentyps zu einer vorhandenen Variablen eine neue Variable entsteht, die aber unter dem bisherigen Namen erreichbar ist. Mit anderen Worten – ein Variablenbezeichner verweist nach einer Veränderung des Datentyps in der Regel auf einen anderen Speicherbereich als vor der Zuweisung.

Grundtypen von Variablen (die ersten vier) sowie die am Ende der Tabelle notierten Sonderdatentypen:

Datentyp	Beschreibung
Boolean	Werte vom Typ Boolean (auch <i>Boole'sche Werte</i> genannt – nach dem Mathematiker George Boole) sind Wahrheitswerte, die nur einen der beiden Werte true oder false (wahr oder falsch) annehmen. Solche Datentypen werden im Rahmen von Vergleichen genutzt.
Number	Dieser Datentyp kann sowohl <i>Ganzzahlwerte</i> als auch <i>Gleitkommawerte</i> 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 Kommazahlen. Dabei muss man mit dem Begriff »Kommazahl« in JavaScript aufpassen, denn diese werden in JavaScript anstelle eines Kommas einen Punkt verwenden, um den Nachkommateil abzutrennen. Neben der normalen Zahlenschreibweise ist die wissenschaftliche Notationsmöglichkeit über die Angaben e oder E möglich. In diesem Fall wird der Potenzwert hinter dieser Exponentialkennung notiert. In JavaScript können Variablen vom Typ Number nicht nur dezimal, sondern auch <i>oktal</i> ¹ (mit einer führenden 0 eingeleitet) oder <i>hexadezimal</i> ² (eingeleitet durch das Präfix 0x) dargestellt werden.
Object	Der allgemeine Datentyp Object 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 + verknüpft werden.
undefined	Der Datentyp beschreibt eine nichtdefinierte Situation. Eine Variable besitzt so lange diesen Wert, bis ihr nach dem Anlegen explizit ein Wert zugewiesen worden ist.
null	Dieser Sonderdatentyp entspricht der Situation, wenn ein Objekt noch keinen Wert hat und entspricht »keiner Bedeutung« oder null. Das ist durchaus ein sinnvoller Wert, der sich von einer leeren Zeichenkette oder der Zahl 0 unterscheidet. Der Datentyp null kann beispielsweise zurückgegeben werden, wenn in einem Dialogfenster mit Abbrechen -Schaltfläche diese betätigt wird.

Tabelle 5: Die Datentypen von JavaScript

1. Das bedeutet zur Basis 8. Mit anderen Worten – der dezimale Wert 9 wird oktal als 10 dargestellt, der dezimale Wert 11 ist oktal 13, 40 wird als 50 dargestellt oder 99 als 143.
2. Das bedeutet zur Basis 16. Da es aber in unserem Kulturkreis nur zehn Zeichen für Zahlen gibt, werden für die fehlenden sechs Darstellungen Buchstaben verwendet. Mit anderen Worten – der dezimale Wert 10 wird hexadezimal als A dargestellt, der dezimale Wert 15 ist hexadezimal F, 42 wird als 2A dargestellt oder 99 als 63.

Die Variablendeklaration in JavaScript

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

```
var [Variablenbezeichner];
```

Listing 10: 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 einmal die 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.

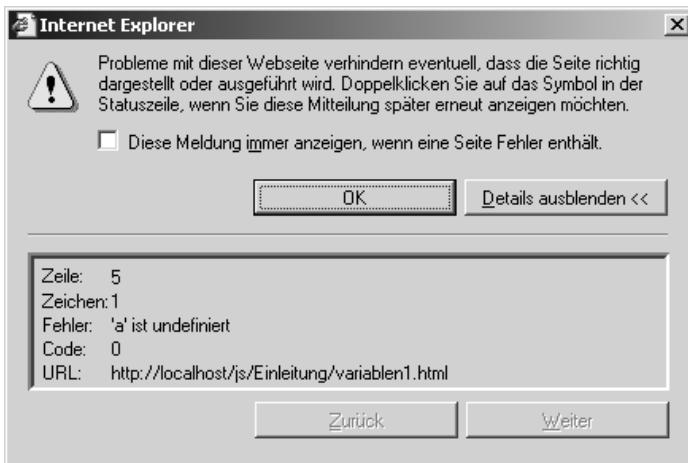


Abbildung 1: Die noch nicht initialisierte Variable a kann nicht verwendet werden, was glücklicherweise vom Browser angezeigt wird.

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).



Abbildung 2: Die Variable wurde mit `var` eingeführt und mit `undefined` initialisiert.

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

```
01 function b() {  
02   var a = 2;  
03   document.write(a+"<br>");  
04 }  
05 a = 1;  
06 document.write(a+"<br>");  
07 b();  
08 document.write(a+"<br>");
```

Listing 11: Der Einsatz von `var` in einer Funktion

Die Ausgabe ist folgende:

```
1  
2  
1
```

Listing 12: Die Ausgabe von dem Beispiel

In Zeile 5 wird die Variable `a` global eingeführt und mit dem Wert `1` versehen. Zeile 6 gibt den Wert aus. Zeile 7 ruft die Funktion `b()` auf. Dort wird in Zeile 2 eine lokale Variable `a` eingeführt und mit dem Wert `2` versehen. Die Ausgabe in Zeile 8 zeigt, dass damit jedoch die globale Variable nicht verändert wurde.

Wenn Sie in Zeile 2 das Schlüsselwort `var` weglassen, ist das die Ausgabe:

```
1  
2  
2
```

Listing 13: Die Ausgabe, wenn `var` in Zeile 2 wegbleibt

In der Funktion `b()` wird in diesem Fall keine lokale Variable `a` eingeführt, sondern der globalen Variablen `a` der neue Wert zugewiesen.

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, denn insbesondere die Bestandteile einer Webseite werden vielfach in Datenfeldern gespeichert (*siehe dazu auch Seite 57*).

Hinweis

Wir werden zum Thema Datenfelder in einem eigenen Kapitel ausführlich Rezepte durchsprechen.

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-Schlüsselworts `new` und einem nachfolgenden Konstruktor erzeugt (*siehe dazu auch Seite 59*). Und zwar mit folgender Syntax (das Schlüsselwort `var` ist in den meisten Fällen optional – es sei denn, Sie wollen ein lokales Datenfeld anlegen):

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

Listing 14: Erzeugen eines Arrays

Das erzeugt ein Datenfeld mit der vorgegebenen Anzahl von Einträgen. Die Arrayeinträge sind mit `undefined` initialisiert. Beachten Sie, dass Sie im Gegensatz zu einer normalen Variablen auch auf das Schlüsselwort `var` verzichten können und die Arrayeinträge dennoch initialisiert sind (mit `undefined`). Eine normale Variable führen Sie entweder zwingend mit dem Schlüsselwort `var` ein oder aber mit einer Wertzuweisung.

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

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

Listing 15: 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:

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

Listing 16: 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.

Beispiel:

```
01 var a = new Array(5);  
02 document.write(a.length+"<br>");  
03 a[5] = 1;  
04 document.write(a.length+"<br>");  
05 a[15] = 1;  
06 document.write(a.length+"<br>");
```

Listing 17: Anlegen und Vergrößern eines Arrays

In dem Beispielcode legen Sie ein Datenfeld der Größe 5 an. Mit der Eigenschaft `length` können Sie die Größe eines Datenfelds abfragen. In der Zeile 3 weisen Sie `a[5]` – was dem sechsten (!) Datenfeldelement entspricht – einfach einen Wert zu, und das Datenfeld wird auf sechs Elemente aufgeblasen. In Zeile 5 weisen Sie gar dem 16. Datenfeldelementeintrag einen Wert zu, und damit ist das Datenfeld 16 Elemente groß (die Elemente `a[6]` bis `a[15]` sind einfach mit `undefined` belegt), was die Ausgabe belegt. Sie sieht so aus:

```
5  
6  
16
```

Listing 18: Die Ausgabe

Das Verhalten ist noch extremer. Selbst das geht:

```
01 b = new Array(0);  
02 document.write(b.length+"<br>");  
03 document.write(b[100]+"<br>");  
04 document.write(b.length+"<br>");  
05 b[100]=0;  
06 document.write(b.length+"<br>");
```

Listing 19: Anlegen und Vergrößern eines Datenfelder – Teil 2

In dem zweiten Beispielcode legen Sie ein Datenfeld ohne Angabe der Größe an. Mit der Eigenschaft `length` fragen Sie wieder die Größe der Datenfelder ab. In der Zeile 3 testen Sie den Wert von `b[100]`. Das Feld ist mit `undefined` in einem wohl definierten Zustand. Die nachfolgende Ausgabe zeigt aber, dass die Größe von dem Datenfeld immer noch 0 ist. In Zeile 5 weisen Sie gar dem 101. Datenfeld-Eintrag einen Wert zu, und damit ist das Datenfeld 101 Elemente groß. Die Ausgabe sieht so aus:

```
0
undefined
0
101
```

Listing 20: Die Ausgabe

Aufgrund 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:

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

Listing 21: Erzeugen eines Datenfelds ohne Elemente

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

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

Listing 22: Wertzuweisung zu einem Arrayelement

Beispiel:

```
meinArray = new Array();
meinArray[0] = 2;
meinArray[1] = 4;
meinArray[2] = 6;
meinArray[3] = 8;
meinArray[4] = 10;
```

Listing 23: 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:

```
adresse = new Array();
adresse[0] = "Hotblack";
adresse[1] = "Desiato";
adresse[2] = 666;
adresse[3] = "Milliways";
adresse[4] = true;
```

Listing 24: Ein Datenfeld mit Zuweisung von Werten unterschiedlichen Datentyps

Funktionen, Prozeduren und Methoden

Die Fähigkeit zur Bildung von *Unterprogrammen* ist ein Hauptkriterium einer modernen Programmieretechnik. 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 von 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 Befehlsschritte vor der Ausführung beim Laden eines Skripts im Rahmen der Webseite schützt. Oft ist diese automatische Ausführung weder gewünscht noch sinnvoll. Erst wenn bestimmte Voraussetzungen 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 Definition von Funktionen erfolgt, indem das dafür in JavaScript vorgesehene Schlüsselwort `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 Klammern 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 25: 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 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 also so aus:

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

Listing 26: 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 abgebrochen 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 wie das 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 JavaScript Funktionen sehr locker. Von der Syntax sehr streng strukturierte Sprachen verlangen, 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 sonst verwendet wird (etwa in Vergleichen einer Kontrollstruktur). JavaScript ist da weniger streng. Eine Funktion kann aufgerufen werden, ohne dass der Rückgabewert irgendwo in der Folge 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 nur um Funktionen oder Prozeduren, die jedoch nur über ihr zugehöriges Objekt verwendet werden können. In einer syntaktisch sehr lockeren und hybrid (sowohl objektorientierte Bestandteile, aber auch prozedurale) aufgebauten Sprache wie JavaScript ist es nicht unbedingt nötig, von der Terminologie her streng zwischen Funktionen und Methoden zutrennen (*siehe dazu Seite 57*).

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 27: 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 als auch in einigen anderen Fällen eine so genannte *Funktionsreferenz* verwenden. Das ist syntaktisch erst einmal die Notation einer Funktion ohne Klammern. Etwa so:

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

Listing 28: 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.

Ausdrücke, Operatoren, Operanden und Wertzuweisungen

Ein Kernbegriff in der Programmierung ist *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 aufgrund der Verknüpfung ein neuer Wert zurückgegeben wird. Es gibt in JavaScript folgende arithmetische Operatoren:

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

Tabelle 6: Die arithmetischen Operatoren von JavaScript

1. Beachten Sie, dass dieses Beispiel unter Umständen zu Rundungsproblemen im JavaScript-Interpreter führt (es kommt 0.6000000000000001 heraus). Allgemein haben Sie in JavaScript bei Gleitkommaoperationen oft mit Rundungsproblemen zu kämpfen.

Einstellige arithmetische Operatoren sind Spezialfälle der arithmetischen Operatoren. Es gibt davon zwei. Sie verwenden nur einen Operanden, der dem Operator nachgestellt wird, und sind ziemlich 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 *arithmetischen 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 7: Die arithmetischen Zuweisungsoperatoren von JavaScript

Inkrement-/Dekrement-Operatoren

In engem Bezug zu 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 Dekrementoperator `--` ist der Gegenspieler vom Inkrement-Operator. Er erniedrigt den Wert des Operanden um 1. Ansonsten ist alles identisch zum Inkrement-Operator.

Bemerkenswert sind beide Operatoren hauptsächlich, weil beide dem Operanden sowohl vor- als auch nachgestellt werden können und dabei sich unterschiedlich verhalten. Wenn der Operator vor dem Operanden steht, erfolgt die Erhöhung des Wertes, bevor der Wert dem Operanden zugewiesen wird. Wenn er hinter dem Operanden steht, erfolgt die Erhöhung, nachdem der Wert bereits zugewiesen wurde. Das hat massive Auswirkungen, wenn der Operator nicht nur alleine als Anweisung verwendet wird.

Beispiel:

```
a=1;
a++;
document.write("Wert von a: " + a + "<br>");
document.write("Wert von ++a: " + ++a + "<br>");
document.write("Wert von a++: " + a++ + "<br>");
document.write("Wert von a: " + a + "<br>");
```

Listing 29: 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.
```

Listing 30: Die Ausgabe

Die zweite Ausgabezeile zeigt, dass eine Variable mit vorangestelltem Operator erhöht und dann ausgegeben wird, während die dritte Ausgabezeile demonstriert, dass 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
==	Gleichheitsoperator
!=	Ungleichheitsoperator
===	Strikter Gleichheitsoperator
!==	Strikter Ungleichheitsoperator
<	Kleiner-als-Operator
>	Größer-als-Operator

Tabelle 8: Die Vergleichsoperatoren von JavaScript

Operator	Beschreibung
<=	Kleiner-als-oder-gleich-Operator
>=	Größer-als-oder-gleich-Operator

Tabelle 8: Die Vergleichsoperatoren von JavaScript (Forts.)

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 ob auch 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 31: Unterschied von Gleichheit und strikter Gleichheit

Die Ausgabe sieht so aus:

```
true
false
true
false
```

Listing 32: Die Ausgabe

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 der Variablen `d` unter Verwendung von `new` und dem `String`-Konstruktor ein neues `String`-Objekt mit gleichem Inhalt erzeugt. Der erste Vergleich in Zeile 5 zeigt, dass die Werte in beiden Variablen `a` und `b` identisch sind, der zweite jedoch, dass die Variablen nicht strikt gleich sind. Ebenso sind `c` beziehungsweise `b` auf der einen 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 Boole'sche (Wahrheits-)Operanden angewandt. Meist (aber nicht immer) sind das Bedingungsüberprüfungen, deren Ergebnisse logisch behandelt werden sollen. Etwa wenn mehrere Bedingungen verknüpft werden sollen. JavaScript kennt folgende logische Vergleichsoperatoren:

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

Tabelle 9: 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 Doppeloperator arbeitet mit drei Operanden. Die Operation benötigt einen Boole'schen Ausdruck vor dem Fragezeichen. Wenn er wahr liefert, wird der Wert vor dem Doppelpunkt zurückgegeben, ansonsten der Wert hinter dem Doppelpunkt. Der Operator ist eine Kurzschreibweise für ein `if-else`-Konstrukt.

Beispiel (ist `a` kleiner dem Wert 10, wird der Text `klein` ausgegeben – sonst der Text `groß`):

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

Listing 33: Einsatz vom konditionalen Operator

Der typeof-Operator

Der Operator `typeof` liefert den Datentyp des nachgestellten Operanden in Form einer Zeichenkette zurück.

Beispiel (es wird der Wert `number` ausgegeben):

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

Listing 34: Die Anwendung von `typeof`

Bitweise Operatoren

Obwohl bitweise Operatoren in den meisten Skriptsprachen nur sehr geringe Bedeutung haben, verfügen fast alle Sprachen über solche Operatoren. So auch JavaScript, beispielsweise bitweise arithmetische Operatoren, welche direkt Bitwerte verändern. Oder bitweise Verschiebungsoperatoren, welche 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 10: Bitweise Operatoren

Operatorprioritäten

Operatoren in JavaScript haben eine festgelegte Rangordnung, die immer dann angewandt wird, wenn mehrere Operatoren in einer Anweisung verwendet werden. Die Prioritäten der JavaScript-Operatoren sind in der nachfolgenden Tabelle von der höchsten Wertigkeit bis zur niedrigsten abwärts 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 Klammernsetzen 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 11: Rangordnung von JavaScript-Operatoren

Anweisungen und Kontrollflusssteuerung

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

Allgemeine Anweisungen

Anweisungen werden in einem JavaScript der Reihe nach oder aufgrund 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. Dies haben wir in fast allen Beispielen bereits angewandt.

Deklarationsanweisungen

Deklarationsanweisungen bedeuten die Einführung eines neuen Elements im Skript, etwa eine Variable oder eine 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 von einem Zuweisungsoperator einen Wert (konstant oder berechnet) stehen hat, der dem linken Ausdruck zugewiesen wird. 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 Situationen 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 an den Aufrufer einer Funktion und zur Rückgabe des Kontrollflusses an den Aufrufer einer Funktion sowie die Anweisung `break`, um aus einem Block (in der Regel eine 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 sicher kaum noch einer Erklärung bedarf¹². Dazu kommt die `switch`-Fallunterscheidung¹³. Mit dem Schlüsselwort `switch` leiten Sie die Fallunterscheidung ein. Als Argument wird in runden Klammern eingeschlossen eine Variable oder ein Ausdruck angegeben, für dessen aktuellen

12. Grundsätzliche Syntax:

```
if ([Bedingung]) {  
  ...Anweisungen  
}  
else {  
  ...alternative Anweisungen  
}
```

13. Grundsätzliche Syntax:

```
switch([Auswahl]) {  
  case [Fall 1]:  
    ...  
    break;  
  case [Fall 2]:  
    ...  
    break;  
  ... weitere case-Fälle  
  ...  
  default:  
    ...  
    break;  
}
```

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 Werten geprüft. Die einzelnen Fälle, zwischen denen unterschieden werden soll, werden untereinander aufgelistet und innerhalb geschweifter Klammern als Blöcke hinter `case` notiert. Optional ist eine `default`-Anweisung. 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 jeden 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 Dubletten möglich (wenngleich unsinnig) sind.

Zwei eng verwandte und sehr oft eingesetzte Kontrollfluss-Anweisungen sind die `while`¹⁴- und die `do-while`-Schleife¹⁵. 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 werden, bevor die Schleifenbedingung überprüft wird (man nennt das eine *annehmende* oder *fußgesteuerte* Schleife). Natürlich gibt es in JavaScript auch die `for`-Schleife¹⁶. 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 35: Die for- und die for-in-Schleife im Einsatz

14. Allgemeine Syntax ist folgende:

```
while ([Bedingung]) {
...Anweisungen
}
```

15. Allgemeine Syntax ist folgende:

```
do {
...Anweisungen
}
```

```
while ([Bedingung]);
```

16. Grundsätzliche Syntax:

```
for ([Zählvariable];[Bedingung];[Schrittweite]) {
...Anweisungen
}
```

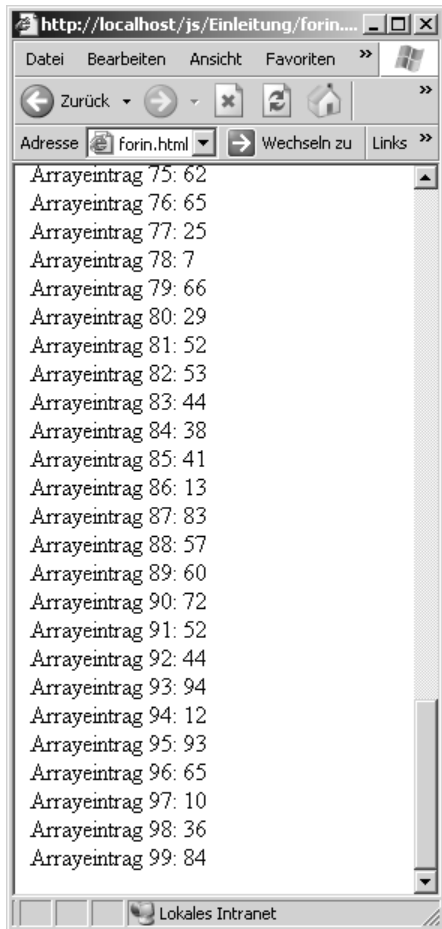


Abbildung 3: Die Ausgabe des Beispiels

DOM und Objekte in JavaScript

Nahezu die gesamte Leistungsfähigkeit von JavaScript als auch anderen clientseitigen Webskriptsprachen basiert darauf, dass man damit aus einer Webseite heraus *Objekte* nutzen kann, die einem Aufrufer eine gewisse Funktionalität bereitstellen¹⁷. Das sind entweder solche Objekte, die zu JavaScript oder einer anderen Webskriptsprache selbst gehören (so genannte *Build-In-Objekte*), oder auch solche, die von außen über eine universelle Schnittstelle 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, ein Teil dieser Seite (z.B. eine Überschrift, ein Absatz,

17. Das gilt auch, wenn man die serverseitige Programmierung betrachtet. Dort werden analog serverseitig verfügbare Objekte verwendet.

eine Grafik, eine Schaltfläche, ein Eingabefeld usw.), den Bildschirm des Besuchers oder dessen Browser. Aber auch Teile von 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.

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

Damit Objekte verwendet werden können, schickt ein potenzieller Anwender eine so genannte *Botschaft* an dieses. 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 36: Die DOT-Notation zum Senden einer Botschaft an ein Objekt (schematisch)

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.

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 *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 dann auch erst verwendet werden kann.

Alle Eigenschaften und Methoden eines spezifischen Objektes stehen erst dann zur Verfügung, wenn ein Programmierer zuvor 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 Schlüsselwörters `new` und der Angabe der passenden Konstruktormethode.

Beispiele:

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

Listing 37: Manuelles Erzeugen von Objekten

Bei Strings können Objekte auch einfach durch Wertzuweisung erzeugt werden.

Beispiel:

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

Listing 38: 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 39: 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 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 genutzt werden, sowohl aus Programmier- und Skriptsprachen als auch Anwendungen heraus. Dem Zugriff auf eine Webseite unter diesem Objektgesichtspunkt liegt ein Konzept zugrunde, das *DOM* (*Document Object Model*) heißt und eine plattform- als auch programmiersprachenübergreifende Schnittstelle bezeichnet. In diesem Konzept wird eine HTML-Seite nicht als statisch aufgebaute, fertige und nicht unterscheidbare Einheit¹⁸, sondern als differenzierbare Struktur betrachtet, deren einzelne Bestandteile Programmen und Skripten dynamisch zugänglich sind. Dieser Ansatz ermöglicht die individuelle Behandlung von Bestandteilen der 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 Teilaspekte. 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 einer Webseite bezüglich ihres Typs, ihrer relevanten Eigenschaften und ihrer Position innerhalb der Webseite indizieren. Dies ist eine Art dynamische Datenbank im Hauptspeicher des Rechners (in Form einer Baumstruktur zu verstehen), die 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, d.h., in der Denkweise einer relationalen Datenbank in den gleichen Datenbanktabellen eingetragen. Auf diese Weise hat der Browser nach dem Laden der Webseite genaue Kenntnis über alle relevanten Daten sämtlicher eigenständig für ihn ansprechbarer Elemente in der Webseite. Welche das jedoch sind und was er damit anstellen kann, das kann sich je nach Browser erheblich unterscheiden.

Jedes Element (etwa ein bestimmter HTML-Tag) ist in einem eigenen Datensatz abgespeichert, der bei Bedarf auch während der Lebenszeit der Webseite aktualisiert wird. 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 hinschreibt. 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 – ein so genanntes *Objektfeld*) mit Namen `forms` und einem Index verfügbar. In der DOT-Notation schreibt man das dann (für das erste Formular der Webseite) so:

18. Wie etwa eine Grafik.

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

Listing 40: 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 41: Zugriff auf das zweite Formularfeld im dritten Formular in einer Webseite

Das Verfahren setzt sich analog auf Eigenschaften und Methoden fort. Jedes Objekt stellt seine spezifischen Eigenschaften und objektgebundenen Methoden über die DOT-Notation bereit. Zugreifen auf Eigenschaften bedeutet entweder 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 die 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. Zugriff auf Methoden erfolgt einfach über den Namen des Objekts und dahinter durch einen Punkt getrennt den Namen der Methode. Zusätzlich – und das ist wichtig – gehören immer (bis auf ganz wenige Spezialfälle) 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 Klammerspaars 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.

Tip

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, ein Objekt ist in einem anderen Objekt als Eigenschaft enthalten. So wie eine Webseite im Browserfenster enthalten ist, müssen die die Realität beschreibenden Objekte in der EDV dieser Tatsache Rechnung tragen. Allerdings sind nicht sämtliche der Objekte in einer einzigen Hierarchiebeziehung miteinander verbunden. Das wäre auch nicht sinnvoll, denn auch in der Realität gibt es unzählige unabhängige Situationen.

Ein Unterobjekt erbt immer dessen nach außen bekannte Eigenschaften und Methoden und erweitert diese um irgendwelche zusätzlichen Funktionalitäten.

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).

Objekt	Beschreibung
all	Das Objekt ermöglicht den direkten Zugriff auf alle Elemente einer HTML-Datei. Es gehört aber nicht zum offiziellen DOM-Standard, sondern ist eine Implementation für den Internet Explorer ab der Version 4.0. Das Objekt wird zwar von fast allen Browsern heutzutage ansatzweise unterstützt, aber den vollen Umfang sollte man nur dem Internet Explorer voraussetzen! Sie sollten bei einem Einsatz auf jeden Fall alle relevanten Browser testen, die Sie unterstützen wollen.
anchor	Das Objekt beinhaltet eine Referenz auf einen Verweisanker in einer HTML-Datei.
applet	Das Objekt beinhaltet eine Referenz auf ein Java-Applet in einer HTML-Datei.
Array	Über diese Klasse werden Arrayobjekte erzeugt. Dessen Elemente können über einen gemeinsamen Bezeichner und einen Index angesprochen werden.
Boolean	Über diese Klasse wird ein Objekt mit Wahrheitswerten (true oder false) als Inhalt erzeugt.
Date	Über diese Klasse wird ein Objekt mit Informationen zu Datum und Uhrzeit erzeugt.
document	Dieses Objekt repräsentiert die Webseite selbst.
event	Ein Objekt, das bei Anwenderereignissen erzeugt wird und für die (zentrale) Ereignisbehandlung genutzt werden kann. Allerdings ist diese zentrale Ereignisbehandlung auch heutzutage inkonsequent in den verschiedenen Browserwelten umgesetzt.
form	Das Objekt beinhaltet eine Referenz auf das Objekt, das ein Formular in einer HTML-Seite repräsentiert.
frame	Das Objekt beinhaltet eine Referenz auf ein Frame in einer HTML-Seite.
Function	Ein Klasse zum Erzeugen von einem Objekt 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 HTML-Datei.
layer	Das Objekt beinhaltet eine Referenz auf einen Layer in einer HTML-Datei. 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 HTML-Datei.
location	Über das Objekt besteht Zugriff auf die Adresszeile des Browsers.

Tabelle 12: Unter JavaScript verfügbare Objekte

Objekt	Beschreibung
Math	Ein Objekt mit zahlreichen mathematischen Konstanten und Methoden.
mimeType	Ein Objekt mit MIME-Type-Informationen.
navigator	Die Objektrepräsentation mit Informationen über den verwendeten WWW-Browser.
Number	Ein Objekt mit numerischen Werten.
plugin	Ein Objekt, das die vorhandenen Plug-Ins in einem Browser repräsentiert.
RegExp	Ein Objekt mit regulären Ausdrücken.
screen	Ein Objekt mit Informationen über den verwendeten Bildschirm des Besuchers.
String	Eine Klasse zur Erzeugung von Textobjekt samt diverser Manipulationsmöglichkeiten für Zeichen und Zeichenketten.
window	Dieses Objekt enthält Statusinformationen über das gesamte Browser-Fenster. Jedes Fenster eines Browsers verwendet sein eigenes window-Objekt. Das window-Objekt ist das höchste Objekt in der Objekthierarchie der Objekte, welche den Browser direkt betreffen.

Tabelle 12: 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 über einen Feldnamen sowie eine Indexnummer identifiziert werden. Ansonsten ist die Anwendung von Eigenschaften und Methoden vollkommen identisch wie bei anderen Objekten, und wir werden im Folgenden meist von Objekten sprechen. Beispiele für solche Objektfelder sind `forms[]` (die Formulare in einer Webseite) oder `elements[]` (die Elemente in einem Webformular).

Die meisten Objektfelder 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 meist dem erzeugenden Objekt 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 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. Für den Fall von Formularen 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 dem Buch ausführlich besprechen).

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

Tabelle 13: Unter JavaScript verfügbare Objektfelder

Der Aufruf von JavaScripts

JavaScript-Anweisungen können einmal 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 das 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 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, wo ein Skriptaufruf einfach als Argument eines Links notiert wird¹⁹.

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.

Eventhandler

Eventhandler sind explizite Bestandteile von HTML (offiziell ab der Version 4.0 unterstützt), die direkt als Parameter in einen HTML-Tag geschrieben werden. Es handelt sich um universelle Attribute für Ereignisse in einer Webseite, die vollkommen unabhängig von JavaScript oder einer anderen Skriptsprache sind. Sie rufen bei einem bestimmten HTML-Tag nur ein beliebiges Skript beim Eintreten des jeweils zugehörigen Ereignisses auf. Das W3C-Konsortium hat Eventhandler offiziell in den HTML-Sprachstandard aufgenommen und auch festgelegt, in welchen HTML-Tags welcher Eventhandler vorkommen darf²⁰. Eventhandler sind ein zentrales Bindeglied zwischen HTML und Skripten: Da sie explizit zu HTML zählen, spielt damit Groß- und Kleinschreibung bei der Notation in der Webseite keine Rolle, und auch sonst gelten die üblichen HTML-Regeln. Die Grundsyntax der Verwendung von Eventhandlern sieht immer gleich aus:

```
<[HTML-Tag] [ATTRIBUTE] on[Event]="Skriptanweisungen">
```

Listing 42: Die Anwendung von Eventhandlern

Dabei ist `[HTML-Tag]` ein beliebiger HTML-Tag, welcher die Reaktion auf ein Ereignis unterstützt, und `[ATTRIBUTE]` bezeichnen die normalen HTML-Attribute, welche den HTML-Tag bezüglich seiner üblichen HTML-Auswirkungen spezifizieren. Mit `on[Event]` bezeichnen Sie das konkrete Ereignis, auf welches reagiert werden soll, und "Skriptanweisungen" sind die tatsächlichen Anweisungen, welche bei Auftreten des spezifizierten Ereignisses ausgeführt werden sollen. Ein Eventhandler verhält sich wie jedes gewöhnliche HTML-Attribut, dem ein Wert zugewiesen wird. Allgemein beginnt ein Eventhandler mit der Silbe `on` (meist kleingeschrieben, aber das ist nicht zwingend), 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`). Hinter dem Namen des Ereignisses wird die auszuführende Anweisung als Wert zugewiesen.

19. Grundsätzliche Syntax:

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

20. Zumindest von der Theorie her – was die Browser-Hersteller daraus machen, ist ein anderes Thema.

Beispiele:

```
<h1 onMouseOver="status='Willkommen'">Homepage</h1>
<button onClick="alert('Button ist kein Standard-Tag')">Klick</button>

```

Listing 43: Verschiedene Eventhandler im Einsatz

Die Technik der konkreten Ereignisbehandlung im Rahmen einer Webseite basiert im Grunde darauf, dass man mit einem Eventhandler einen Browser anweist, permanent auf das Auftreten von bestimmten Ereignissen bei einem spezifischen Element zu warten. Er lauscht in gewisser Weise wie ein Radioempfänger in einem Auto auf einer bestimmten Frequenz, ob durch ein Signal eine Verkehrsmeldung angekündigt und dann gesendet wird, um dann darauf zu reagieren (das Autoradio stellt sich beispielsweise lauter, der Browser ruft gegebenenfalls eine Funktion oder eine Objektmethode auf). Dieses »Hören auf das Auftreten von Ereignissen« läuft parallel zum normalen Programmablauf und vor allem im Hintergrund ab.

Die konkrete Umsetzung der Ereignisbehandlung ist in der Vergangenheit²¹ in verschiedenen Browsern erheblich voneinander abgewichen. Das führte in der Vergangenheit dazu, dass zahlreiche Ereignisse, die in einem Browser unterstützt wurden, in einem anderen ignoriert wurden oder gar Fehler auslösten. Insbesondere die Verwendung von dem `Event`-Objekt (siehe unten) führte zu Fehlermeldungen und sogar zu Browser-Abstürzen, wenn eine solche Seite im »falschen« Browser geladen wurde. Mittlerweile haben sich die modernen Browser bezüglich der Ausführung von Eventhandlern jedoch einigermaßen angeglichen. Dennoch – wenn Sie auf solche Ereignisbehandlungstechniken zurückgreifen, die explizit nur in einem Browserkonzept unterstützt werden, muss man unbedingt (!) Tests in verschiedensten Browsern durchführen (auch ältere Varianten) und bei potenziellen Komplikationen die Browserwelten trennen.

Der explizite Aufruf eines Eventhandlers aus JavaScript

Ein Eventhandler kann nicht nur als Attribut in einen HTML-Tag geschrieben werden, sondern auch über eine andere Art ausgelöst werden (ein expliziter Aufruf aus JavaScript). Etwa wie im nachfolgenden Beispiel:

```
01 <html>
02 <head>
03 <script language="JavaScript">
04 <!--
05 function mFunktion() {
06   status="Herzlich willkommen";
```

Listing 44: Der explizite Aufruf eines Eventhandlers per JavaScript

21. Vergangenheit bedeutet nicht, dass das vor Urzeiten war – es soll nur die Browsergenerationen beschreiben, die heutzutage nicht mehr als aktuell gelten (als Browser wie der Netscape Navigator 6, der Internet Explorer 5.5 oder Opera 6 sowie selbstverständlich deren Vorläufer).

```
07 }  
08 /-->  
09 </script>  
10 <body>  
11 <form name="mF">  
12 <input type="button" name="mB" value="OK">  
13 </formM>  
14 <script language="JavaScript">  
15 <!--  
16 document.mF.mB.onclick = mFunktion  
17 /-->  
18 </script>  
19 </html>
```

Listing 44: Der explizite Aufruf eines Eventhandlers per JavaScript (Forts.)

Die in dem Beispiel verwendete Syntax beinhaltet eine Funktion, die in dem zweiten Skript-Container dem Formular `mF` und dort dem Element `mB` (einer Schaltfläche) zugeordnet wird (in Zeile 16). Wenn der Anwender den Formularbutton anklickt, wird die Funktion aufgerufen. Dabei sind zwei Dinge zu beachten:

- ▶ Eventhandler in dieser Form sind Funktionsreferenzen in JavaScript. Deshalb dürfen bei der Zuordnung der Funktion keine (!) Klammern angegeben werden. Ansonsten würde die Funktion unmittelbar bei der Zuordnung ausgeführt und das Ergebnis `undefined` zugewiesen.
- ▶ Diese Technik funktioniert erst ab JavaScript 1.1. Und da wir uns hier bei der Angabe des Eventhandlers in JavaScript bewegen, ist Groß- und Kleinschreibung relevant. Der Eventhandler-Name (als Bestandteil des Skripts) muss vollständig in Kleinbuchstaben notiert werden (wie in dem Beispiel in Zeile 16). Wenn Sie sich grundsätzlich an die – im Grunde optionale – Regel halten, einen Eventhandler im HTML-Tag mit kleinem `on` zu beginnen und dann einen Großbuchstaben zu notieren, ist das rein von der Schreibweise eine eindeutige Unterscheidungsmöglichkeit.

Achtung

Der explizite Aufruf eines Eventhandlers per JavaScript, wie er hier besprochen wird, funktioniert in vielen relevanten (auch neueren) Browsern nicht! Der Internet Explorer und Opera unterstützen zwar das Verfahren, aber der Konqueror, der Netscape Navigator und Mozilla kommen beispielsweise auch in aktuellen Ausgaben nicht damit zurecht. Sollte es keinen zwingenden Grund für die Verwendung dieser Technik geben, sollten Sie darauf verzichten.

Globale Ereignisbehandlung und das event-Objekt

In JavaScript können Sie auf Ereignisse in einer Webseite auch über die Auswertung von einem speziellen Ereignisobjekt reagieren. Wenn Sie dieses Objekt verwenden, hat das einige weitreichende Konsequenzen. Die Ereignisüberwachung wird beispielsweise direkt in JavaScript verlagert²² und findet nicht mehr auf der HTML-Seite statt. Dabei wird ausgenutzt,

22. Wie auch beim expliziten Aufruf eines Eventhandlers per JavaScript.

dass permanent von speziellen JavaScript-Hintergrundroutinen das Auftreten eines Ereignisobjekts überwacht wird. Je nach Art des Ereignisses kann der JavaScript-Interpreter darauf reagieren. Des Weiteren können Sie mit diesem Ereignismodell eine globale Reaktion auf Ereignisse realisieren.

Wenn in einer Webseite ein Ereignis auftritt, wird dabei seit JavaScript 1.1 ein eigenes Objekt generiert – das `event`-Objekt, welches 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.

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 und enthalten jeweils spezifische Informationen zu dem auslösenden Ereignis. Beispielsweise wird der Klick mit der Maus auf irgendeinen Bestandteil der Webseite ein `event`-Objekt erzeugen, welches Informationen über die verwendete Taste, eventuell gedrückte Zusatz Tasten und die Koordinaten des Klicks enthält.

Nun ist es sicher nicht Sinn und Zweck eines Ereignismodells, dass permanent Ereignisse abgefeuert und diese nie beachtet werden (sprich darauf reagiert wird). Ein `event`-Objekt wird, wenn man unter JavaScript darauf reagieren will, an einen Mechanismus zum Behandeln von `event`-Objekten in Form einer Botschaft weitergereicht.

Auf welche Art und Weise auf welches erzeugte Ereignisobjekt dann reagiert werden soll, wird durch ein abstraktes Modell geregelt – dem Ereignismodell, das zu dem DOM-Konzept zählt. Darin sind sowohl die Verhaltensweisen auf Ereignisse festgelegt, die konkrete Syntax zur Umsetzung, aber auch erst recht diejenigen Ereignisse, die überhaupt zur Kenntnis genommen werden.

Da Ereignisobjekte während der Lebenszeit einer Webseite permanent im Hintergrund entstehen, stehen sie auch überall im Code zur Verfügung. Es gibt verschiedene Konzepte, auf welche Weise so ein `event`-Objekt angesprochen werden kann. Sie können es einfach über seinen Namen (`event`) ansprechen. Dabei müssen Sie noch angeben, von welchem Objekt aus das `event`-Objekt erzeugt wurde. Die Syntax dazu ist die in der objektorientierten Programmierung übliche Punktnotation:

```
[Objekt].event
```

Listing 45: Zugriff auf ein spezifisches event-Objekt

Beispiel:

```
document.event
```

Listing 46: Zugriff auf ein spezifisches event-Objekt für eine Webseite

Sie wollen bei einer Auswertung in der Regel dann natürlich eine ganz bestimmte Information eines Ereignisses verwenden. Dies sind bei einer objektorientierten Betrachtungsweise einfach die Eigenschaften des Objektes. So können Sie etwa einen Mausklick wie folgt ansprechen:

```
document.event.CLICK
```

Listing 47: Ein Klick auf eine Webseite

Dies verstehen vom Prinzip her sowohl der Internet Explorer als auch der Netscape Navigator, und so weit hören sich die Ausführungen sicher ganz viel versprechend an. Nur wurden die Konzepte zur Behandlung des `event`-Objekts von Microsoft und Netscape vollkommen inkompatibel umgesetzt. Wir wollen dennoch beide Welten kurz durchsprechen.

Das alte Netscape-Modell zur globalen Ereignisbehandlung

Unter dem (alten) Netscape-Modell können 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 wird²³:

```
[Objekt].captureEvents([Ereignistyp])
```

Listing 48: Auffangen eines Ereignisses im alten Netscape-Modell

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 weitergeleitet – die zentrale Ereignisbehandlungsroutine. Dies ist in diesem Fall einfach eine Funktion, die als Argument das erzeugte `event`-Objekt mit der gewünschten Eigenschaft verwendet. Als Ereignistyp können alle bei einem Objekt im alten Netscape-Konzept unterstützten Eventhandler verwendet werden. Das alte Netscape-Ereignismodell unterstützt die folgenden Events beziehungsweise Eventhandler in den beschriebenen Situationen:

Ereignis	Zugehörige Elemente	Abgefeuert bei	Eventhandler
Abort	Bilder	Abbruch des Ladevorgangs bei einem Bild	onAbort
Blur	Browserfenster und alle Formularelemente	Entziehen des Eingabefokus durch den Anwender	onBlur
Change	Textfelder, Textbereiche, Auswahllisten	Wertänderung eines Elements durch den Anwender	onChange

Tabelle 14: Globale Ereignisse im (alten) Netscape-Ereignismodell

23. Es gibt noch drei weitere Methoden, die wir aber nicht verfolgen wollen.

Ereignis	Zugehörige Elemente	Abgefeuert bei	Eventhandler
Click	Schaltflächen, Optionsfelder, Kontrollkästchen, Submit-Schaltflächen, Reset-Schaltflächen, Links	Ein Klick des Anwenders auf ein Element oder Link	onClick
DragDrop	Das Browserfenster	Der Anwender zieht ein Objekt in das Browserfenster und lässt es dort los.	onDragDrop
Error	Images und Browserfenster	Fehler beim Laden eines Dokuments oder Bilds	onError
Focus	Browserfenster und alle Formularelemente	Ein Element erhält durch den Anwender den Eingabefokus.	onFocus
KeyDown	Dokumente, Bilder, Links, Textbereiche	Der Anwender drückt eine Taste herunter.	onKeyDown
KeyPress	Dokumente, Bilder, Links, Textbereiche	Der Anwender hält eine Taste gedrückt.	onKeyPress
KeyUp	Dokumente, Bilder, Links, Textbereiche	Der Anwender lässt eine Taste los.	onKeyUp
Load	Dokumentenbody	Laden einer Seite	onLoad
MouseDown	Dokumente, Schaltflächen, Links	Drücken einer Maustaste	onMouseDown
MouseMove	Keine Defaultsituation	Allgemeine Mausbewegung	onMouseMove
MouseOut	Bereiche, Links	Der Mauszeiger bewegt sich aus dem Bereich einer Image-map oder eines Links.	onMouseOut
MouseOver	Links	Bewegung des Mauszeigers auf dem Link	onMouseOver
MouseUp	Dokumente, Schaltflächen, Links	Loslassen einer Maustaste	onMouseUp
Move	Browserfenster	Der Anwender oder ein Skript bewegt ein Fenster.	onMove
Reset	Formulare	Der Anwender klickt auf eine Reset-Schaltfläche.	onReset

Tabelle 14: Globale Ereignisse im (alten) Netscape-Ereignismodell (Forts.)

Ereignis	Zugehörige Elemente	Abgefeuert bei	Eventhandler
Resize	Browserfenster	Der Anwender oder ein Skript verändert eine Fenstergröße.	onResize
Select	Textfelder oder Textbereiche	Der Anwender selektiert in einem Formular ein Eingabefeld.	onSelect
Submit	Formulare	Der Anwender schickt ein Formular ab.	onSubmit
Unload	Dokumentenbody	Der Anwender verlässt eine Seite.	onUnload

Tabelle 14: Globale Ereignisse im (alten) Netscape-Ereignismodell (Forts.)

Grundsätzliche Vorgehensweise bei dieser Form der Ereignisbehandlung ist folgende:

1. Das Auffangen eines Events wird ermöglicht.
2. Die Definition des Eventhandlers erfolgt.
3. Die Registrierung des Eventhandlers erfolgt.

Um das Auffangen eines Events zu ermöglichen, wird bei dem entsprechenden Objekt (etwa `window`) eine Anweisung der Art notiert:

```
window.captureEvents(Event.CLICK);
```

Listing 49: Das Auffangen aller Klickereignisse im Fenster wird angezeigt.

Das Argument der Methode `captureEvents()` ist eine Eigenschaft von dem `event`-Objekt, die den Typ des Ereignisses spezifiziert. Der Aufbau ist einfach das vollständig großgeschriebene Ereignis, das per Punktnotation `Event` nachgestellt wird.

Um mehrere Ereignisse aufzufangen, kann als Argument eine Liste von solchen Eigenschaften angegeben werden. Die Eigenschaften werden mit dem Zeichen `|` separiert. Um beispielsweise die Ereignisse `Click`, `MouseDown` und `MouseUp` aufzufangen, muss Folgendes notiert werden:

```
window.captureEvents(Event.CLICK | Event.MOUSEDOWN | Event.MOUSEUP)
```

Listing 50: Auffangen von drei Ereignissen im Fenster

Es können durchaus mehrere Behandlungsfunktionen definiert werden, wenn entsprechend in der `captureEvents()`-Methode zugehörige Ereignisse aufgefangen werden.

Die Definition der Eventhandler steht für die Definition einer Funktion, die das Ereignis behandelt. Das Argument `e` ist das Ereignisobjekt für das Ereignis. Beispielsweise so:

```
function BehandleEreignis(e) {  
  ...  
}
```

Listing 51: Eine schematische Ereignisbehandlungsmethode

Das abschließende Registrieren des Eventhandlers läuft so ab:

```
window.onClick = BehandleEreignis;
```

Listing 52: Registrierung des Eventhandlers

Das Ereignis, für das eine bestimmte Funktion als Eventhandler agieren soll, wird dem zugehörigen Objekt nachgestellt und auf der rechten Seite der Bezeichner der Eventhandler-Funktion zugewiesen. Dabei sollte beachtet werden, dass dort keine Klammern notiert werden dürfen, denn es handelt sich um Funktionsreferenzen! Spielen wir ein kleines Beispiel durch:

```
<html>  
<script language="JavaScript">  
<!--  
window.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP);  
function ereig1(e) {  
  status='Gargravarr';  
}  
function ereig2(e) {  
  status='';  
}  
window.onMouseDown = ereig1;  
window.onMouseUp = ereig2;  
//-->  
</script>  
<body>  
<h1>Das funktioniert nur im alten Netscape-Modell</h1>  
</body>  
</html>
```

Listing 53: Ein Beispiel mit Anwendung des alten Eventmodells von Netscape

Das Beispiel soll auf das Drücken und Loslassen der Maustaste im gesamten Fenster reagieren und beim Drücken der Maustaste einen Text in die Statuszeile des Browsers schreiben. Beim Loslassen wird der Text wieder gelöscht. Zumindest wenn Sie das Beispiel in eine Version des Netscape Navigators vor der Version 6 laden.



Abbildung 4: In die Statuszeile wird der angegebene Text geschrieben.

Wenn Sie das Beispiel in den Internet Explorer laden, reagiert dieser mit einer Fehlermeldung.



Abbildung 5: Der Internet Explorer verweigert die Unterstützung.

Frappierend ist jedoch, dass auch neuere Varianten des Navigators keinerlei Lust zeigen, auf die alte Ereignisbehandlungsstruktur zu reagieren. Zwar wird kein Fehler angezeigt, aber Ignoranz hilft dem Besucher auch nicht. Dieses Ignorieren erfolgt auch in anderen Browsern wie Mozilla, Opera oder dem Konqueror (ohne Fehlermeldung).

Das Microsoft-Ereignismodell zur globalen Ereignisbehandlung

Das Microsoft-Objektmodell erlaubt die zentrale Behandlung von Ereignissen mittels der folgenden Syntax:

```
<script for=[Objekt] event=[Ereignis] language="JavaScript">
    ...[Konkrete Anweisungen]...
</script>
```

Listing 54: Der gesonderte <script>-Container zur globalen Ereignisbehandlung im Microsoft-Modell

Als [Ereignis] geben Sie einfach die im vorherigen Abschnitt erklärten Eventhandler an (allerdings auf jeden Fall vollkommen kleingeschrieben!). Mehr ist nicht notwendig, um alle Ereignisse des spezifizierten Typs von dem angegebenen Objekt zentral zu verarbeiten.

Beispiel:

```
<script for=document event=onclick language="JavaScript">
    ...
</script>
```

Listing 55: Reaktion auf alle Klicks in der Webseite

Innerhalb des Skripts können Sie dann auf alle Eigenschaften des event-Objekts zugreifen, die der Internet Explorer kennt. Das gesamte Prozedere ist unkompliziert, solange kein Browser die Seite lädt, der damit nicht klarkommt. Schauen wir uns ein Beispiel an, das onmouseover in allen Elementen einer Webseite überwacht:

```
<html>
<script for=document event=onmouseover language="JavaScript">
    status= "X-Position: " + event.clientX + ", Y-Position: " + event.clientY;
</script>
<body>
<h1>Globale Ereignisbehandlung nach dem Microsoft-Modell</h1>
</body>
</html>
```

Listing 56: Überwachung von Mausbewegungen

Wenn Sie das Beispiel in den Internet Explorer laden und mit der Maus über die Überschrift fahren, sehen Sie in der Statuszeile die jeweiligen Koordinatenangaben.

Wenn Sie das Beispiel jedoch in einen anderen Browser laden, kommt es entweder zu Fehlermeldungen, oder aber die Anweisungen werden ignoriert.

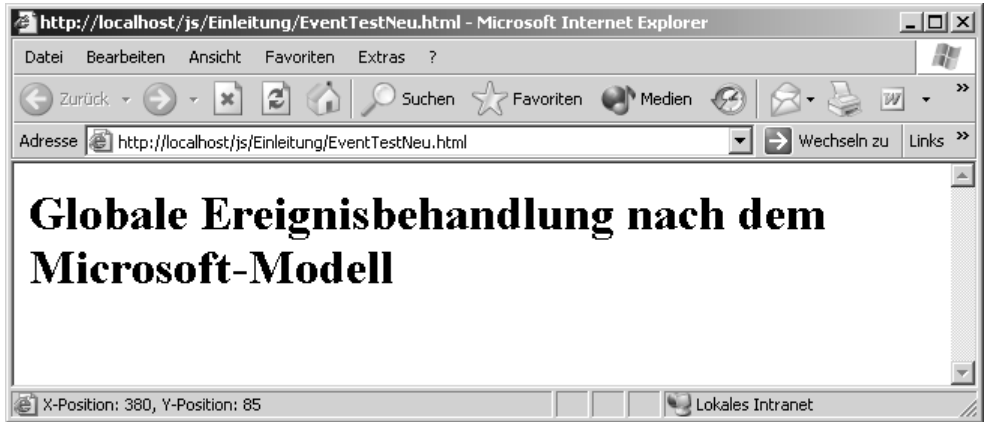


Abbildung 6: In der Statuszeile werden die Koordinaten des Mauszeigers angezeigt.



Abbildung 7: Der alte Navigator reagiert mit Fehlermeldungen.

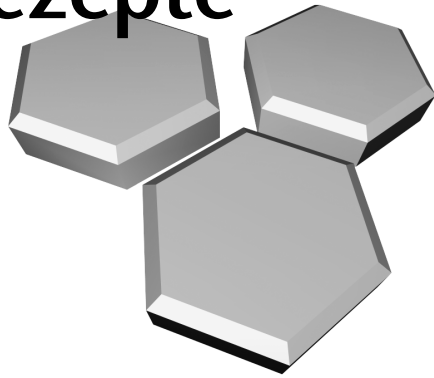
Browser, die beim Microsoft-Modell mit Fehlermeldungen reagieren, sind alte Netscape-Versionen oder auch alte Opera-Browser. Der Opera-Browser kommt zwar ab der Version 6 gut mit dem event-Objekt zurecht, wird aber bei der speziellen Erweiterung von dem `<script>`-Container Schwierigkeiten haben. Einmal versteht er ihn als normalen Container. Das bedeutet, alle darin enthaltenen Anweisungen werden beim Laden der Webseite ausgeführt. Zum anderen werden die Ereignisse selbst beim Auftreten nicht ausgeführt. Neue Netscape- und Mozilla-Browser melden zwar keine Fehler, üben sich jedoch grundsätzlich in vornehmer Ignoranz oder führen ebenso Anweisungen beim Laden der Seite aus.

Hinweis

Grundsätzlich handeln Sie sich mit globaler Ereignisbehandlung unter Umständen viele Probleme ein. Sie sollten sorgfältig abwägen, ob eine globale Ereignisbehandlung wirklich den Aufwand wert ist. Wenn man globale Ereignisbehandlung für verschiedene Browserwelten sicher machen möchte, kommt man um eine Trennung der Browser nicht herum.

TEIL II

Rezepte



Grundlagen

In diesem Kapitel sollen Rezepte besprochen werden, die zu den wesentlichen Grundlagen rund um JavaScript zählen.

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

1. Die Notation eines Skriptcontainers in eine 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 den 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 auf die Hochkommata bei der Wertzuweisung verzichten (`<script language=javascript>`), obwohl die strenge Auslegung der HTML-Syntax diese im Prinzip fordert. 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. Es ist sogar so, dass es ein explizites Problem mit diesem Attribut gibt, auf das im Rezept zur Angabe der JavaScript-Version eingegangen werden soll (siehe Seite 84).

Wenn Sie eine andere Skriptsprache oder -angabe verwenden wollen (etwa VBScript, JScript oder auch der ursprüngliche Name von JavaScript – LiveScript), geben Sie diese vollkommen analog an. Beachten Sie, dass die Angabe von JScript oder LiveScript künstlich Probleme in einigen Browsern erzeugen kann, die mit der Angabe von JavaScript als Sprache

zurechtkommen würden. Es gibt wenige Situationen, wo das sinnvoll ist. Hingegen ist die Angabe der JavaScript-Version unter manchen Umständen zweckmäßig (*siehe das Rezept auf Seite 84*).

Hinweis

Im Rahmen der Rezepte in dem 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.

Tipp

Sie können in einer Webseite mehrere Skriptcontainer verwenden (sogar mit verschiedenen Skriptsprachen). Ebenso können Sie auf das Attribut `language` ganz verzichten. In dem Fall wird der Default-Skriptinterpreter des Browsers aufgerufen. Dies ist in der Regel der JavaScript-Interpreter, aber darauf verlassen kann 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 in den Browser abgearbeitet wird. Wenn der Browser eine auszuführende Anweisung lädt, wird sie vollzogen. 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 wird, wo es benötigt wird und es beim Laden automatisch ausgeführt werden soll. In JavaScript ist es aber 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, wo 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, einen `<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 er 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 steht 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. Dies ist zwar nicht zwingend, aber sicherer für den Fall, dass ein Browser die Seite lädt, der keine Skripte interpretieren kann.

Hinweis

Beachten Sie, dass hier nicht von Browsern die Rede ist, bei denen der Anwender JavaScript deaktiviert hat. Diese verstehen den `<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. Das betrifft im Grunde nur Browser, die vor 1995 entwickelt wurden. Die Bedeutung von dieser Vorsichtsmaßnahme ist also nicht sonderlich hoch.

Wenn Sie bei einem Skriptcontainer auf den HTML-Kommentar verzichten und ein Browser die Seite lädt, der keine Skripte interpretieren kann, ist dies gleichbedeutend damit, dass er den `<script>`-Tag nicht kennt. Er wird ihn nach dem *Prinzip der Fehlertoleranz*¹ einfach ignorieren. Das Problem ist dann aber, dass nachfolgender Text – in unserem Fall sind die Skriptanweisungen für einen HTML-Interpreter nicht Text – ohne Formatierung auf dem Bildschirm dargestellt wird. Ein Anwender bekommt also unter Umständen die Skriptanweisungen einfach als Text im Browser angezeigt, und das ist nicht im Sinne des Erfinders.

Achtung

Die Anwendung des HTML-Kommentars im Inneren des `<script>`-Containers beinhaltet einige Fallen. Der Netscape Navigator wird in einigen Versionen Probleme mit dem Ende-Zeichen des HTML-Kommentars haben. Er versteht es als JavaScript-Anweisung (`--` ist in JavaScript der Dekrementoperator, und den versucht der Browser auszuführen). Man versteckt 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 der 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 Skript-Container mit HTML-Kommentaren sieht also vollständig so aus:

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

Listing 1: Ein typischer Container zum Einbinden von einem JavaScript (ohne Versionsangabe)

1. Das Prinzip der Fehlertoleranz zählt zu den wichtigsten Eckdaten von HTML bzw. der gesamten Funktionalität des WWW. Dieses Prinzip veranlasst Browser unter anderem dazu, bei der Interpretation von Webseiten so fehlertolerant wie irgend möglich zu sein.

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 Implementation selbst enthalten. Natürlich wird die Wartung und Anpassung erheblich erleichtert, und auch sonst gibt es weitere Benefits. 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 der `<script>`-Tag verwendet, der dann aber dem Attribut `src` die externe Datei angibt. Die Syntax ist also so:

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

Listing 2: Referenz auf eine externe JavaScript-Datei

In Anführungszeichen wird hinter dem Attribut `src` der Name bzw. 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 einen vollständigen URL, der 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="meineSkripte.js">
</script>
```

Listing 3: Angabe einer relativ zur Webseite im gleichen Verzeichnis befindlichen JavaScript-Datei

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

Listing 4: Angabe einer relativ zur Webseite im parallelen Verzeichnis notierten JavaScript-Datei

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

Listing 5: Angabe von einem 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. Diese werden dann als Text verstanden.

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 Schrittfolgen aufzurufen.

Achtung

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. In dem Konfliktfall wird immer die zuletzt definierte Funktion oder Variable (im Sinn der Abarbeitung der Webseite von oben nach unten) verwendet.

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 anderer Form auch bei Style-Sheets gibt. Dieses System hat aber einige erhebliche Nachteile. Der wichtigste ist wohl, dass es nicht bei allen Browsern funktioniert (zumindest nicht älteren Datums). Außerdem ist es 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`, gefolgt von einem Doppelpunkt, als Attribut angeben. Die Syntax sieht also so aus:

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

Listing 6: Schema einer Inline-Referenz

Beispiel:

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

Listing 7: 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.

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

Wenn Sie ein Skript in einem `<script>`-Container notieren, ist es 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 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 8: Alternative Informationen können im `<noscript>`-Bereich notiert werden.

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

JavaScript gibt es mittlerweile in mehreren Versionen, und natürlich sind gewisse Techniken explizit erst sukzessive in bestimmten Versionen von JavaScript eingeführt worden. Wenn Sie solche neueren 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 nicht. Bei der Einbindung eines JavaScripts in eine Webseite (siehe das Rezept Seite 79) 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 9: 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 scheinbare 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 neuen und/oder für sie unbekannten Anweisungen auszuführen. 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 zu der Angabe `JavaScript` ist) Probleme macht. Er 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 der Grund ist, dass der Internet Explorer bei der Angabe von dem MIME-Typ die Versionsangabe vollkommen ignoriert. Das gilt auch für andere Browser! Ein Browser wird also auch Container mit Versionsangaben ausführen, die er unter Umständen nicht unterstützt. Mit anderen Worten: Die gleichzeitige Angabe von dem MIME-Typ und einer Versionsangabe für JavaScript sollte unterbleiben.

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

Beispiel (*versionsangabe.html*):

```
01 <html>
02   <body>
03     <script language='JavaScript'>
04       <!--
05         document.write("1. Das schreibt jeder Browser, der JavaScript kann - ➡
language='JavaScript'.<hr>");
06       //-->
07     </script>
08     <script language='JavaScript1.0'>
09       <!--
10         document.write("2. Das sollte identisch zu oben sein - ➡
language='JavaScript1.0'.<hr>");
11       //-->
12     </script>
```

Listing 10: Eine Webseite mit mehreren Skriptcontainern für mehrere JavaScript-Versionen

```

13     <script language='JavaScript1.0' type='text/javascript'>
14     <!--
15         document.write("3. Auch das sollte identisch zu der ersten Angabe sein - ␣
language='JavaScript1.0' type='text/javascript'.<hr>");
16     //-->
17     </script>
18     <script language='JavaScript1.1'>
19     <!--
20         document.write("4. Das können nur Browser, die JavaScript ab Version 1.1 ␣
verstehen - language='JavaScript1.1'.<hr>");
21     //-->
22     </script>
23     <script language='JavaScript1.2'>
24     <!--
25         document.write("5. Das können nur Browser, die JavaScript ab Version 1.2 ␣
verstehen - language='JavaScript1.2'.<hr>");
26     //-->
27     </script>
28     <script language='JavaScript1.3'>
29     <!--
30         document.write("6. Das können nur Browser, die JavaScript ab Version 1.3 ␣
verstehen - language='JavaScript1.3'.<hr>");
31     //-->
32     </script>
33     <script language='JavaScript1.4'>
34     <!--
35         document.write("7. Das können nur Browser, die JavaScript ab Version 1.4 ␣
verstehen - language='JavaScript1.4'.<hr>");
36     //-->
37     </script>
38     <script language='JavaScript1.5'>
39     <!--
40         document.write("8. Das können nur Browser, die JavaScript ab Version 1.5 ␣
verstehen - language='JavaScript1.5'.<hr>");
41     //-->
42     </script>
43     <script language='JavaScript1.5' type='text/javascript'>
44     <!--
45         document.write("9. Das können nur Browser, die JavaScript ab Version 1.5 ␣
verstehen. Aber über type='text/javascript' werden einige Browser motiviert, sich zu ␣
übernehmen.<hr>");
46     //-->
47     </script>
48 </body>
49 </html>

```

Listing 10: 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.

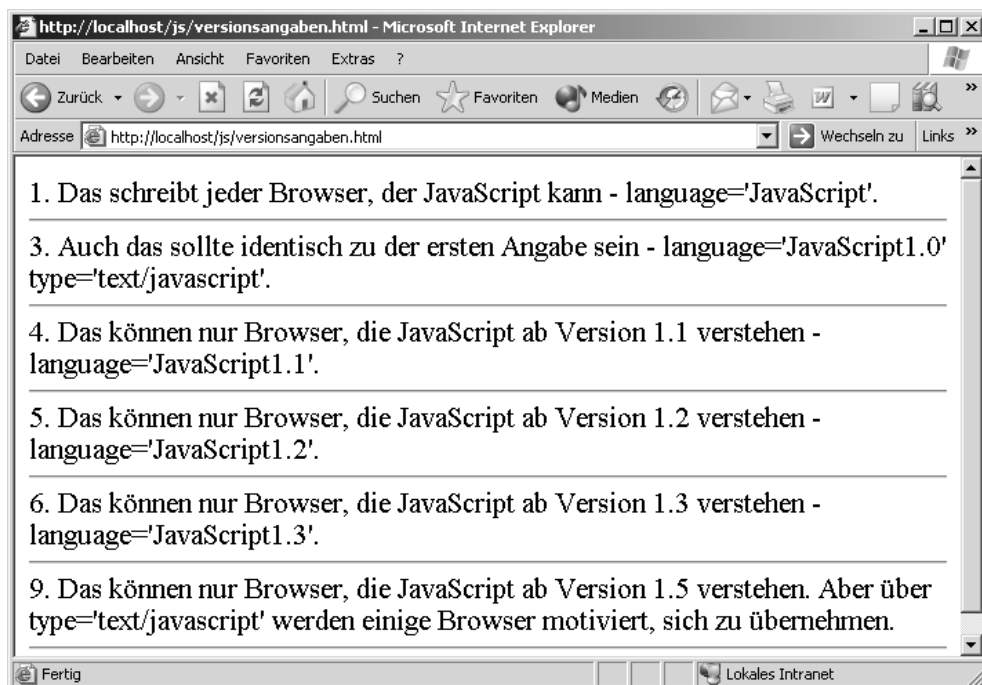


Abbildung 1: Der Internet Explorer 6 traut sich nur JavaScript 1.3 und kann scheinbar kein JavaScript 1.0.

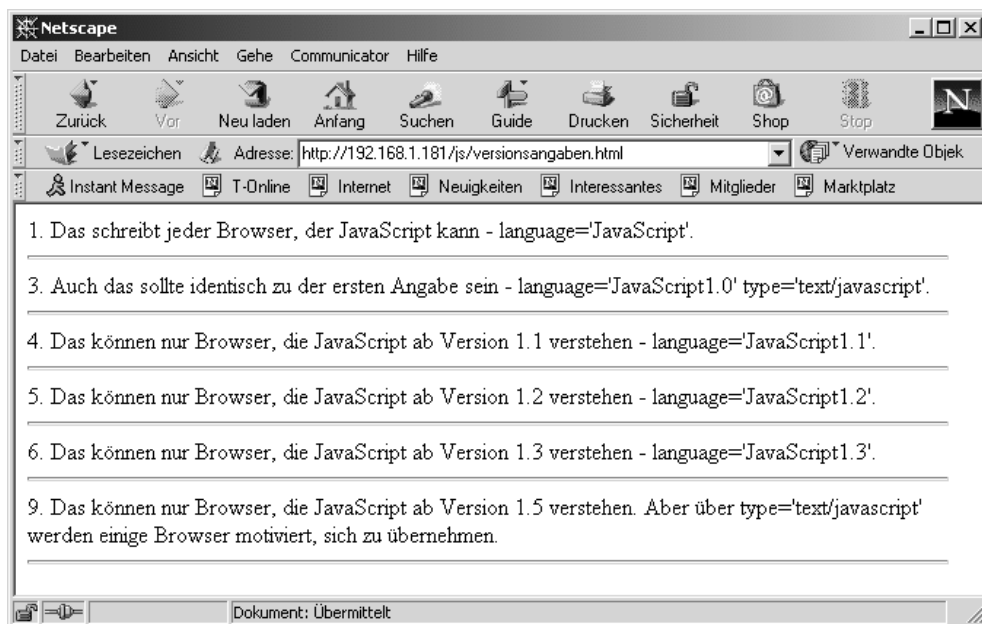


Abbildung 2: Der Navigator 4.7 kann nur JavaScript 1.3.

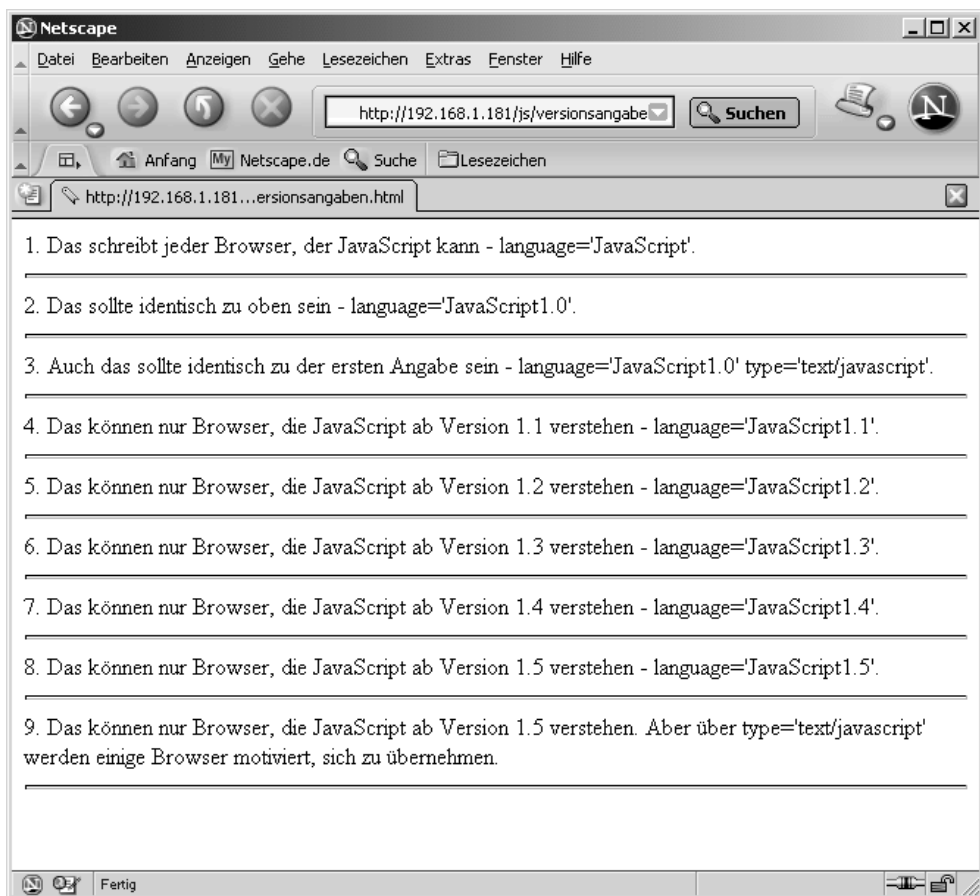


Abbildung 3: Der Navigator 7.1 kann auch JavaScript 1.5 und alle anderen Notationen.

3 Wie kann ich sicherstellen, dass ein Browser nur bekannte JavaScript-Anweisungen ausführt?

Das Rezept mag im ersten Moment überflüssig erscheinen, denn im Rezept »Wie kann ich eine JavaScript-Version bei der Einbindung angeben?« auf Seite 84 können Sie sehen, dass Sie beim `<script>`-Container explizit eine Version für JavaScript 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 javascript1.3 gekennzeichnet sind (siehe Abbildung 8 auf Seite 87). Dabei unterstützt der Internet Explorer 6 den Standard 1.5 nahezu vollständig. Ebenso wird Opera 7.2 nur Container bis zur Angabe javascript1.4 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 Browserversionen offiziell mit welcher Version von JavaScript mindestens zurechtkommen²:

Browser	JavaScript-Version					
	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	x	x	x	x	x	x
Mozilla 1.2	x	x	x	x	x	x
Mozilla 1.4	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	
Microsoft Internet Explorer 2.x						
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		

Tabelle 1: Die offizielle Kompatibilität wichtiger Browserversionen mit JavaScript-Standards

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 Browservariante und die Browserversion abfragen. Siehe dazu das Rezept »Wie kann ich eine Browserweiche erstellen?« auf Seite 112. Universell können Sie also folgende Schablone einsetzen (*versionsschablone.js*):

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

```
<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 Browserweiche für die Browser auf, die mit diesen Anweisungen
  klarkommen, offiziell jedoch nur JavaScript 1.3 unterstützen */
  //-->
</script>
```

Listing 11: Eine Schablone zum qualifizierten Sicherstellen der Unterstützung bestimmter Anweisungen

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

JavaScript ist mit Abstand die verbreitetste Programmier Technik in modernen Webclients. Dennoch sollten Sie sich nicht darauf verlassen, dass sie bei einem Besucher unterstützt wird. Ein Webprojekt sollte niemals JavaScript so verwenden, dass es ohne die Unterstützung vollkommen unbrauchbar ist³. 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 Werbepopups 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 (also in der

3. Es sei denn, Sie können auf die Besucher ohne JavaScript-Unterstützung verzichten.

4. Das sollte in der Regel genügen.

Regel beim Laden der Seite *index.html*, die bei den meisten Webprojekten als Einstiegsseite verwendet wird) 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 können, 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 einem anderen URL aus. Funktioniert der, muss auch JavaScript aktiviert sein. Wenn Sie den Anwender nun komfortabel führen wollen, erstellen Sie eine Einstiegsseite in Ihr Webprojekt, das im Wesentlichen eine solche Weiterleitung enthält und ihn für den Fall vom deaktivierten JavaScript *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 der 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">
06 <!--
07   location.href="mitJS.html";
08 //-->
09 </script>
10 <body>
11 <a href="ohneJS.html">Wenn die automatische Weiterleitung nicht funktioniert,
klicken Sie bitte hier, um zum Webprojekt ohne JavaScript-Unterstützung zu kommen</
a><br>
12 <a href="mitJS.html">Wenn die automatische Weiterleitung nicht funktioniert,
klicken Sie bitte hier, um zum Webprojekt mit JavaScript-Unterstützung zu kommen</a>
13 </body>
14 </html>

```

Listing 12: Trennung nach JavaScript-Unterstützung oder nicht

5. Was mir etwas paranoid erscheint :-).

Wenn man diese Webseite lädt, erfolgt eine automatische Weiterleitung per JavaScript mit dem `location`-Objekt (`location.href` – Zeile 7). Nun wird diese Anweisung bei deaktiviertem JavaScript nicht ausgeführt. Im Header finden Sie in Zeile 3 einen HTML-`<meta>`-Tag, der 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.

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

Direkt können Sie die unterstützte JavaScript-Version nicht abfragen, aber indirekt. Im Rezept »Wie kann ich eine JavaScript-Version bei der Einbindung angeben?« auf Seite 84 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 von einem Browser zu testen. Das funktioniert zumindest bis zur Version 1.3 sehr zuverlässig und mit Einschränkungen auch für neuere Versionen⁶.

Beispiel (*versionsunterstuetzung.html*):

```

01 <script language="JavaScript1.5">
02 <!--
03   location.href="index1_5.html";
04 //-->
05 </script>
06 <script language="JavaScript1.4"> // im Grunde überflüssig
07 <!--
08   location.href="index1_4.html";
09 //-->
10 </script>
11 <script language="JavaScript1.3">
12 <!--
13   location.href="index1_3.html";
14 //-->
15 </script>
16 <script language="JavaScript1.2">
17 <!--
18   location.href="index1_2.html";
19 //-->
20 </script>
21 <script language="JavaScript1.1">
22 <!--
23   location.href="index1_1.html";

```

Listing 13: Ein indirekter Test, welche Version von JavaScript unterstützt wird

6. Darüber hinaus sind keine allgemeinen Aussagen möglich (siehe auch das Rezept »Wie kann ich sicherstellen, dass ein Browser nur bekannte JavaScript-Anweisungen ausführt?« auf Seite 84).

```
24 !-->
25 </script>
26 <script language="JavaScript">
27 <!--
28   location.href="index1_0.html";
29 !-->
30 </script>
```

Listing 13: Ein indirekter Test, welche Version von JavaScript unterstützt wird (Forts.)

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.

6 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 Browser werden mit dem 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 Browserkennung?

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

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

Listing 14: 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()`).

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 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);
```

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 $n - 1$ Wert für `appName` explizit angeben. Vor allem muss man berücksichtigen, dass sich die Browseridentifikation bei manchen Browsern manuell vom Anwender verändern lässt.

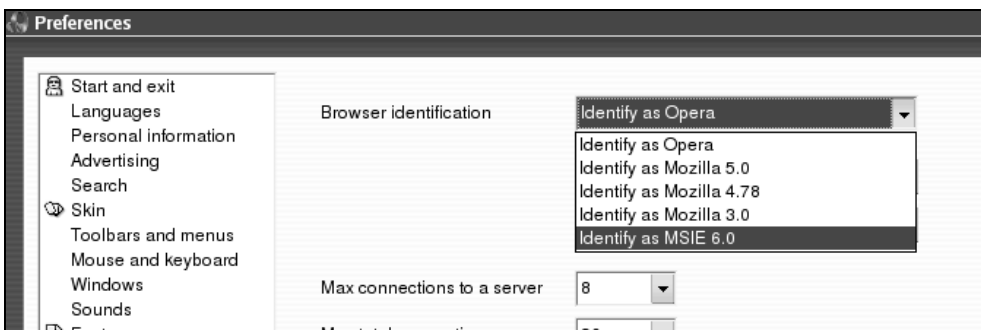


Abbildung 4: In vielen Browsern kann der Anwender den Wert von `navigator.appName` ändern.

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 Abprüfung auf Gleichheit mit »Netscape« durch Skripte `false` liefern kann, obwohl der Browser ursprünglich diesen Wert in der Eigenschaft gesetzt hatte. Man muss also grundsätzlich mit einer Änderung von `navigator.appName` rechnen.

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

Praxisorientierter 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 (*nmc.html*):

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

Listing 15: Trennen der Browser über `navigator.appName` und dem 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. Das 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 Rezept 3 von diesem Abschnitt.

Wie erfolgt die indirekte Identifizierung unter Ausnutzung eines Mangels?

Es gibt die Möglichkeit zum Trennen vom Internet Explorer (samt kompatiblen Browsern) und einigen Versionen des Navigators (samt kompatiblen Browsern), die ein ähnliches Prinzip wie in dem Fall von deaktiviertem JavaScript 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) in der Regel das Objekt `all`.

Für die Alternative gibt es die direkte Identifizierung einiger Versionen des Navigators über das Objekt `layers`. Diese Technik hat man früher sehr oft verwendet.

Hinweis

Die Abfrage auf beide Objekte sind aber auch Paradebeispiele, dass die Ausnutzung eines Mangels oder einer ganz bestimmten Eigenheit in einem bestimmten Browser, welche 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 Browserhersteller nicht auf Dauer weigern können, einen von diesem verbreiteten, proprietären Standard zu implementieren. In dem Fall werden Identifika-

tionen 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 dem 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 über `all` unbrauchbar.

Auf der anderen Seite kann es auch passieren, dass irgendwann ein Browserhersteller alle seine Prinzipien aufgibt und zu seinen eigenen Ideen inkompatibel wird. Wenn er darüber nicht Pleite geht, 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 das Objekt nicht mehr! Folge – sie werden von dem nachfolgenden Beispiel weder zu der einen noch 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 16: 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. Dann wird weitergeleitet, wenn `layers`-Unterstützung vorhanden ist. Sonst erfolgt überhaupt keine Weiterleitung.

Was ist die sinnvollste Identifizierungstechnik?

Alle drei Techniken zu Identifizierung eines Browsers sind mangelbehaftet. Sie müssen entscheiden, mit welchem Mangel Sie am besten leben können. Offensichtlich ist es sinnvoller, Rezept 2 denn Rezept 1 zu verwenden. Aber ob Sie sich auf die Unterstützung von einem spezifischen Objekt verlassen oder lieber die Selbstidentifikation verwenden, ist Geschmacksache. Ich tendiere massiv zur Verwendung von `navigator.appName`, denn der typische Anwender verändert diese Einstellung im Browser nicht. Das machen vielleicht wenige Poweruser, und die können – falls eine Seite nicht korrekt angezeigt wird – den Browser wieder umkonfigurieren. Aber wie gesagt – Anwender von Browsern wie Opera stellen gelegentlich Internet Explorer als `navigator.appName` ein und lassen sich dann überhaupt nicht explizit erkennen⁷.

7. Auf solchen Einstellungen beruhen zum Teil auch die Statistiken, die dem Internet Explorer einen Anteil von über 90% zugestehen. Viele Browser mit veränderter `navigator.appName`-Eigenschaft werden einfach als Internet Explorer gezählt.

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).

7 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.

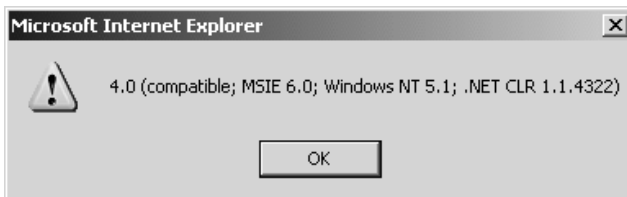


Abbildung 5: Das liefert der Internet Explorer 6.0 bei der Abfrage von `navigator.appVersion`.

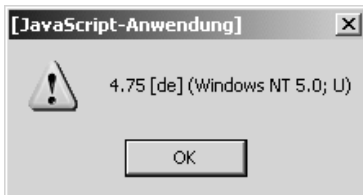


Abbildung 6: Der Netscape Navigator 4.7

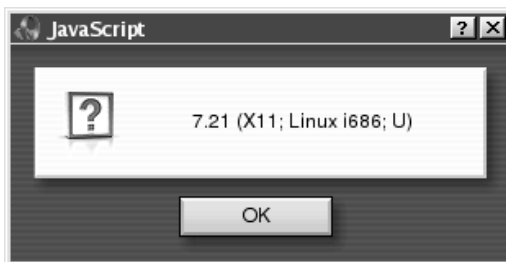


Abbildung 7: Opera 7.2 unter Linux

Um die Version wirklich abfragen zu können, müssen Sie `navigator.appVersion` nach der passenden Information parsen. Dazu macht man sich bei einer Lösung die Tatsache zunutze, dass `navigator.appVersion` ein String ist, den man mit geeigneten String-Funktionen durchsuchen kann. Aber da die Versionsinformation am Beginn von dem String steht und es zwei Toplevel-Funktionen von JavaScript gibt, die einen String in eine Ganzzahl bzw. Gleitkommazahl umwandeln können, genügt meist deren Verwendung. 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 den Zahlen auch Text enthalten. Ab dem ersten nicht-nummerischen 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` zurückgegeben. 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 Nachkomma-Anteil abgeschnitten.

Bei der Auswertung der Version müssen Sie beachten, dass die Versionsnummer nicht immer identisch mit der nach außen dokumentierten Versionsnummer ist. So gibt der Internet Explorer 6.0 etwa den Wert 4 aus, der Navigator 4.7 jedoch auch den Wert 4.7. Der Navigator 7.2 liefert jedoch den Wert 5, Mozilla 4, Opera 7.21 den Wert 7.21 und der Konqueror 3.1.x den Wert 5.

```
if (parseInt(navigator.appVersion)<5){  
    ...  
}
```

Listing 17: Test auf Browserversion

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 18: Expliziter Test auf Netscape Navigator bis zur Version 4.7

Im Kapitel über 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?« finden Sie eine weitere Möglichkeit, wie Sie die Versionsnummer eines Webbrowsers mit Hilfe solcher regulären Ausdrücke ermitteln können.

8 Wie kann ich die Spracheinstellung eines 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 daraus die Sprachversion von einem Browser 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 in einer Browserversion ändern kann. Der Navigator liefert für `navigator.language` etwa in der Version 4.7 den Wert `de`, während die Version 7.2 `de-DE` als Wert enthält.



Abbildung 8: Der Wert von `navigator.language` beim Navigator 4.7

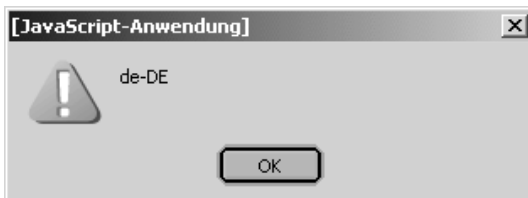


Abbildung 9: Der Wert von `navigator.language` beim Navigator 7.2

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 19: 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.

9 Wie kann ich die Bildschirmauflösung eines Besuchers ermitteln?

Ein großes Highlight von clientseitiger 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 wird⁸. 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. Das Objekt `screen` stellt Ihnen mit den Eigenschaften `width` und `height` Zugang zur Bildschirmauflösung eines Besuchers bereit. Des Weiteren können Sie auch `availHeight` (Höhe des verfügbaren Bildschirms in Pixel, d.h. minus Elementen, 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 ist⁹. Etwa 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;
```

Listing 20: Trennung nach Bildschirmauflösung

8. 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.
9. Es gibt die Auflösungen 1600 x 1200 und 1600 x 1024 oder 1280 x 1024 und 1280 x 960. Der Wert von `screen.width` ist also alleine nicht eindeutig.

```

    case 768:
        ...
        break;
    case 864:
        ...
        break;
    case 960:
        ...
        break;
    case 1024:
        ...
        break;
    case 1050:
        ...
        break;
    case 1000:
        ...
        break;
    case 1024:
        ...
        break;
    case 1200:
        ...
        break;
    default:
        ...
}

```

Listing 20: 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. Sie sollten in dem 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 (*bildschirmaufloesung.html*):

```

01 <html>
02 <body>
03 <script language="JavaScript">
04 <!--
05 switch(screen.height) {
06   case 480:
07     location.href="s480.html";
08     break;
09   case 600:

```

Listing 21: Automatische Weiterleitung je nach Auflösung

Grund-
lagen

Core

Formu-
lare

Aus-
nahme

Arrays

Strings

Datum

Platt-
form

DHTML

Events

```
10     location.href="s600.html";
11     break;
12     case 768:
13         location.href="s768.html";
14         break;
15     case 864:
16         location.href="s864.html";
17         break;
18     case 960:
19         location.href="s960.html";
20         break;
21     case 1024:
22         location.href="s1024.html";
23         break;
24     case 1050:
25         location.href="s1050.html";
26         break;
27     case 1000:
28         location.href="s1000.html";
29         break;
30     case 1024:
31         location.href="s1024.html";
32         break;
33     case 1200:
34         location.href="s1200.html";
35         break;
36     default:
37         location.href="sDefault.html";
38 }
39 //-->
40 </script>
41 </body>
42 </html>
```

Listing 21: Automatische Weiterleitung je nach Auflösung (Forts.)

10 Wie kann ich die eingestellte Anzahl an Farben bei einem Besucher 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 eingestellte Anzahl an Farben bei einem Besucher kann abgefragt werden. Wie auch bei der Bildschirmauflösung ist es eine Spezialität der clientseitigen Programmierung, Informationen zum eingestellten Grafikmodus eines Besuchers auswerten zu können¹⁰. Mit dem `screen`-Objekt und den Eigenschaften `colorDepth` (Bit-Tiefe

10. 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.

der Farbpalette) haben Sie Zugang zu Farbeinstellungen auf dem Clientrechner. Die Eigenschaft beinhaltet den Wert 4 bei 16 Farben, 8 bei 256 Farben, 15 bei 32768 Farben, 16 bei 65536 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 22: 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 `location.href` einsetzen. Sie sollten in dem 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       <!--
05       switch(screen.colorDepth) {
06         case 4:
07           location.href="f16.html";
08           break;
09         case 8:
10           location.href="f256.html";
11           break;
12         case 15:
```

Listing 23: Automatische Weiterleitung je nach Farbtiefe

```
13     location.href="f32768.html";
14     break;
15 case 16:
16     location.href="f65536.html";
17     break;
18 case 24:
19     location.href="f16_7M.html";
20     break;
21 default:
22     location.href="fDefault.html";
23 }
24 //-->
25 </script>
26 </body>
27 </html>
```

Listing 23: Automatische Weiterleitung je nach Farbtiefe (Forts.)

11 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)¹¹ 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).

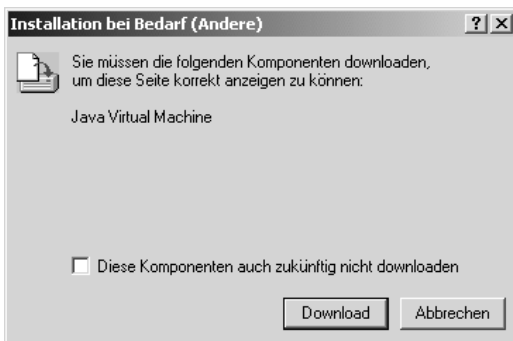


Abbildung 10: Der Internet Explorer wird in einigen Versionen ohne Unterstützung für Java ausgeliefert.

Was nutzt einem Ersteller einer Webseite die beste Technik, wenn zahlreiche Besucher aufgrund fehlender Unterstützung in Ihrem Browser diese nicht nutzen können oder zeitaufwändig Java über das Internet nachinstallieren müssen? Bevor Sie in einer Webseite bei einem

11. 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.

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 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 24: 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 den 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       <!--
05       if (navigator.javaEnabled()){
06         document.write(
07           "<applet code='MeinApplet.class' width=450 height=550></applet>");
08       }
09       else{
10         document.write("Sie verfügen leider über keine Java-Unterstützung");
11       }
12       //-->
13     </script>
14   </body>
15 </html>
```

Listing 25: Die dynamische Einbindung von einem Java-Applet

12 Wie kann ich bestimmen, welche Plug-Ins ein Browser unterstützt?

Ein Browser kann quasi natürlich 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 manche weitere Multimediaformate

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

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 Grundausstattung nicht verarbeiten könnte. Viele Browser werden mittlerweile bereits mit einer Reihe an Plug-Ins ausgeliefert. Verwendet eine Webseite nun eine bestimmte Dateart 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 auch¹² 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 (eine wichtige Ausnahme ist der Internet Explorer) alle installierten Plug-Ins in einem Browsers abfragen können. Ältere Browser wie der Navigator 4.7 unterstützen die Eigenschaft nicht. 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 bereit:

- ▶ `name`: Der Name von dem Plug-In.
- ▶ `filename`: Der Name der Plug-In-Datei auf dem Clientsystem.
- ▶ `description`: 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.
- ▶ `length`: 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.

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 width=600>
04       <script language="JavaScript">
05         <!--
06         i=0;
07         for(i=0;i< navigator.plugins.length;i++)
08           document.write(
09             "<tr><td>Name: </td><td>" + navigator.plugins[i].name +
```

Listing 26: Ausgabe aller Plug-Ins in einem Browser

```

10      "</td><td>Datei: </td><td>" + navigator.plugins[i].filename +
11      "</td><td>Beschreibung: </td><td>" +
12      navigator.plugins[i].description+
13      "</td></tr>");
14  //-->
15  </script>
16  <table>
17  </body>
18  </html>

```

Listing 26: Ausgabe aller Plug-Ins in einem Browser (Forts.)



Abbildung 11: Die Plug-Ins im Navigator 7.1 unter Windows



Abbildung 12: Die installierten Plug-Ins im Opera unter Linux

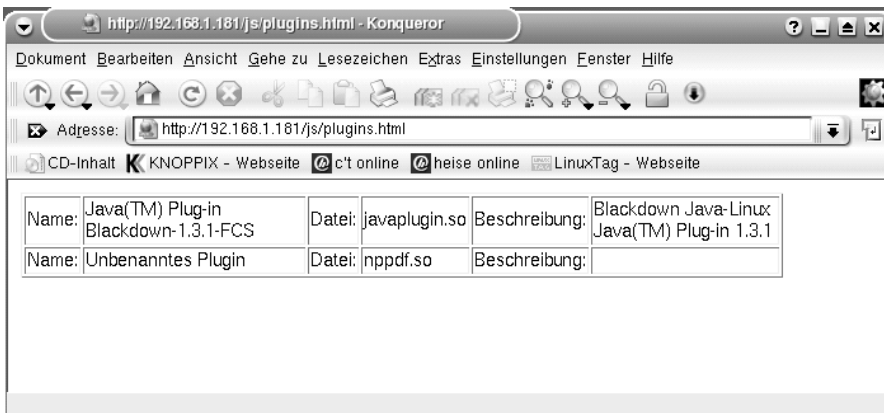


Abbildung 13: Der Konqueror

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 von dem 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 27: 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 müssen, 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 verwenden¹³.

Beispiel:

```
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 28: Explizite Prüfung auf ein Shockwave-Plug-In und anschließende Verwendung, falls es vorhanden ist

13 Wie kann ich bestimmen, welche MIME-Typen ein Browser unterstützt?

Sie können bei fast allen neueren Browsern (eine wichtige Ausnahme ist der Internet Explorer) mit JavaScript bestimmen, ob der Client eines Besuchers in der Lage ist, mit einem bestimmten *MIME-Typ* (Multipart Internet Mail Extension) umzugehen. Ältere Browser wie der Navigator 4.7 unterstützen die Technik nicht. Das `navigator`-Objekt besitzt dazu die Eigenschaft `mimeTypes`. Dabei handelt es sich um ein Objekt (oder genauer Array), in dem alle unterstützten MIME-Typen zu finden sind. Ein MIME-Typ kann durch den Browser entweder direkt oder ein zusätzliches Plug-In unterstützt werden (deshalb kann man auch über

13. Alternativ würde in einigen Fällen auch ein Zugriff über die objektorientierte DOT-Notation funktionieren (was wir aber nicht verfolgen).

den MIME-Typ indirekt 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:

- ▶ **type:** Der Name des MIME-Typs als String (zum Beispiel "video/x-ms-wm" oder "audio/x-wav").
- ▶ **description:** Eine Beschreibung des MIME-Typs.
- ▶ **enabledPlugin:** Eine Referenz zu dem `plugins`-Objekt, über das der MIME-Typ behandelt wird. Über `enabledPlugin` haben Sie Zugriff auf die gleichen Eigenschaften wie über ein `plugins`-Objekt von `navigator` (siehe das Rezept »Wie kann ich bestimmen, welche Plug-Ins ein Browser unterstützt?« auf Seite 105).
- ▶ **suffixes:** Ein String, der alle möglichen Dateinamenerweiterungen für den MIME-Typ auflistet. Die gültigen Erweiterungen sind in der Regel drei Buchstaben lang und werden durch Kommata getrennt, wenn es mehr als eine gültige Dateierweiterung gibt.
- ▶ **length:** Die Anzahl der Elemente in dem Array.

Zum Durchsuchen des Arrays bietet sich die `for`-Schleife an. Das nachfolgende Beispiel zeigt Ihnen in einer dynamisch generierten Tabelle alle MIME-Typen, die in einem Browser installiert sind, samt Name, Beschreibung, Dateierweiterungen sowie den Namen des zugehörigen Plug-Ins an.

Beispiel (*mimetypen.html*):

```

01 <html>
02   <body>
03     <table border width=600>
04       <script language="JavaScript">
05         <!--
06           i=0;
07           for(i=0;i< navigator.mimeTypes.length;i++)
08             document.write(
09               "<tr><td>Typ: </td><td>" + navigator.mimeTypes[i].type +
10               "</td><td>Dateierweiterungen: </td><td>" +
11               navigator.mimeTypes[i].suffixes +
12               "</td><td>Beschreibung: </td><td>" + navigator.mimeTypes[i].description +
13               "</td><td>Plug-In: </td><td>" + navigator.mimeTypes[i].enabledPlugin.name
14               + "</td></tr>");
15         //-->
16       </script>
17     <table>
18   </body>
19 </html>

```

Listing 29: Ausgabe aller MIME-Typen in einem Browser



Events



Sie können auch über die Namen eines MIME-Typs auf Elemente des `mimeType`-Arrays zugreifen. Etwa so:

```
navigator.mimeType["application/x-mplayer2"].type
navigator.mimeType["image/jpeg"].suffixes
```

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.mimeType["Name des MIME-Typs "]) {
    ...
}
```

Listing 30: Prüfung auf die Unterstützung eines spezifischen MIME-Typs

Beispiel:

```
if (navigator.mimeType["application/x-mplayer2"]) {
    ...;
}
else {
    document.writeln("Der MIME-Typ application/x-mplayer2 wird bei Ihnen leider nicht
unterstützt.");
}
```

Listing 31: Explizite Prüfung auf einen MIME-Typ für ein Windows Media Player-Plug-In

14 Wie kann ich eine Browserweiche erstellen?

Ich nehme an, viele Leser kennen Webseiten mit der Angabe, dass diese für einen bestimmten Browser optimiert sind. Diese Aussage sagt in vielen Fällen zwischen den Zeilen aus, dass der Ersteller der Webseite sein Handwerk nicht versteht und dafür um Milde bittet ;-). 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 ob 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 haben¹⁴.

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 90*)?
- ▶ Ob Java unterstützt wird (*siehe Rezept »Wie kann ich testen, ob bei einem Browser Java unterstützt wird?« auf Seite 104*)?
- ▶ Ob ein bestimmtes Plug-In vorhanden ist (*siehe Rezept »Wie kann ich bestimmen, welche Plug-Ins ein Browser unterstützt?« auf Seite 105*)?
- ▶ Ob ein bestimmter MIME-Typ unterstützt wird (*siehe Rezept »Wie kann ich bestimmen, welche MIME-Typen ein Browser unterstützt?« auf Seite 109*)?
- ▶ Welche Bildschirmauflösung beim Anwender eingestellt ist (*siehe Rezept »Wie kann ich die Bildschirmauflösung eines Besuchers ermitteln?« auf Seite 100*)?
- ▶ Welche Farbtiefe beim Anwender eingestellt ist (*siehe Rezept »Wie kann ich die eingestellte Anzahl an Farben bei einem Besucher ermitteln?« auf Seite 102*)?
- ▶ Welchen Browser ein Anwender verwendet (*siehe Rezept »Wie kann ich den Browser eines Anwenders abfragen« auf Seite 93*)?
- ▶ Welche Version eines Browsers ein Anwender verwendet (*siehe Rezept »Wie kann ich die Version eines bekannten Browsers bei einem Anwender abfragen?« auf Seite 93*)?

Sämtliche relevante 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 vom Konzept zwei 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.

14. 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.

Browserweiche 1

Das erste Rezept benutzt nur eine einzige numerische Variable (`browser`) zur Identifizierung von dem Browser sowie dem Betriebssystem und (im Fall Navigator) der Browserversion. Durch einen Trick entsteht aber eine *eindeutige* Kennung, so dass sich für die Auswahl von dem konkreten Fall die `switch`-Fallunterscheidung ideal anwenden lässt. Teil 1 von dem Trick basiert darauf, je nach Bedingung die Variable `browser` um eine eindeutige Potenz von dem Wert 2 hochzuzählen. 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 ($= 2 \text{ hoch } 0$), Windows den Wert 2 ($= 2 \text{ hoch } 1$) und Linux den Wert 4 ($= 2 \text{ hoch } 2$) zu. Wollen Sie nun noch den Internet Explorer und alle anderen Browser trennen, bekommt der Internet Explorer beispielsweise den Wert 8 ($= 2 \text{ hoch } 3$) zugeordnet und alle anderen Browser den Wert 16 ($= 2 \text{ hoch } 4$). 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 handeln und nicht der Internet Explorer sein.
- ▶ Der Wert 10 kann nur als Addition von 2 und 8 entstehen. Folglich muss es sich um einen Mac OS-Browser handeln und der Internet Explorer sein.

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 ($2 \text{ hoch } 6$ ist 64, und damit stehen die Potenzen von $2 \text{ hoch } 0$ bis $2 \text{ hoch } 6$ zur Verfügung). Reservieren Sie nur alle Zweierpotenzen unter 10, stehen Ihnen maximal vier eindeutige Werte zur Verfügung ($2 \text{ hoch } 3$ ist 8), und das kann unter Umständen zu wenig sein.

Für die Browservariante verwenden Sie die nachfolgenden Zweierpotenzen bis 10000. Das beginnt dann mit $2 \text{ hoch } 7 = 128$ und geht bis $2 \text{ hoch } 13 = 8192$. Damit können Sie ebenfalls eindeutig sieben Browser 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 vorschiebt).

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   <script language="JavaScript">
03     <!--
04     // Die Kennvariable
05     var browser = 0;
06     // Bedingung 1
07     // Identifikation des Betriebssystems - Bereich von 0 bis 99
08     if (navigator.appVersion.indexOf('Mac') != -1) browser +=1; // Mac OS
09     else if (navigator.appVersion.indexOf('Win') != -1) browser +=2; // Windows
10     else if ((navigator.appVersion.indexOf('Linux') != -1)
11             || (navigator.appVersion.indexOf('X11') != -1)) browser +=4; // Linux
12     else browser +=8; // anderes Betriebssystem
13     // Bedingung 2
14     // Identifikation des Browsers - Bereich von 100 bis 9.999
15     if(navigator.appName.indexOf('Netscape')!=-1) browser +=128;
16     else if(navigator.appName.indexOf('Internet Explorer')!=-1) browser +=256;
17     else if(navigator.appName.indexOf('Opera')!=-1) browser +=512;
18     else if(navigator.appName.indexOf('Konqueror')!=-1) browser +=1024;
19     else browser +=2048;
20     // Bedingung 3
21     // Identifikation der Version für den Netscape-Browser
22     // Der Bereich nimmt nur eine Dezimalstelle von 10.000 bis 99.999 ein,
23     // da nur ein Fall notwendig ist
24     if ((navigator.appName.indexOf('Netscape')!=-1) &&
25         (parseInt(navigator.appVersion)<5)) browser +=16384;
26     // Die eigentliche Browserweiche
27     switch(browser) {
28       case 16513: document.write("Mac OS , Netscape, Version <= 4.7");
29         break;
30       case 16514: document.write("Windows, Netscape, Version <= 4.7");
31         break;
32       case 16516: document.write("Linux, Netscape, Version <= 4.7");
33         break;
34       case 16520: document.write(
35         "Anderes Betriebssystem, Netscape, Version <= 4.7");
36         break;
37       case 129: document.write("Mac OS , Netscape Version > 4.7 oder Mozilla");
38         break;
39       case 130: document.write("Windows, Netscape Version > 4.7 oder Mozilla");
40         break;
41       case 132: document.write("Linux, Netscape Version > 4.7 oder Mozilla");
42         break;
43       case 136: document.write(
44         "Anderes Betriebssystem, Netscape Version > 4.7 oder Mozilla");
45         break;
46       case 257: document.write("Mac OS , Internet Explorer");
47         break;
48       case 258: document.write("Windows, Internet Explorer");
49         break;

```

Listing 32: Eine sehr flexibel erweiterbare Browserweiche

Grund-
lagen

Core

Formu-
lareAus-
nahme

Arrays

Strings

Datum

Platt-
form

DHTML

Events

```

50 case 272: document.write("Anderes Betriebssystem, Internet Explorer");
51     break;
52 case 513: document.write("Mac OS ,Opera");
53     break;
54 case 514: document.write("Windows,Opera");
55     break;
56 case 516: document.write("Linux,Opera");
57     break;
58 case 520: document.write("Anderes Betriebssystem,Opera");
59     break;
60 case 1028: document.write("Linux,Konqueror");
61     break;
62 case 1032: document.write("Anderes Betriebssystem,Konqueror");
63     break;
64 case 2049: document.write("Mac OS , anderer Browser");
65     break;
66 case 2050: document.write("Windows, anderer Browser");
67     break;
68 case 2052: document.write("Linux, anderer Browser");
69     break;
70 case 2056: document.write("Anderes Betriebssystem, anderer Browser");
71     break;
72 default: document.write("Keine bekannte Plattform/Browser/Version");
73 }
74 //-->
75 </script>
76 <body>
77 </body>
78 </html>

```

Listing 32: Eine sehr flexibel erweiterbare Browserweiche (Forts.)

Beachten Sie, dass das Rezept nicht alle Kombinationen berücksichtigt. 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 können 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 eine reine Fleißaufgabe ist, von der Sie hier einfach profitieren können¹⁵, ist es durch seine logische

15. Was man nicht alles für seine Leser macht ;-)).

Struktur gut für viele Bedingungen zu verwenden und vermeidet potenzielle Fehler, die in Rezept 1 in komplexen Situationen bei der Summierung der abzutestenden 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

Aufgrund der Länge des Rezepts von über 1900 Quellcodezeilen werden wir es hier nur auszugsweise abdrucken. Sie finden das vollständige Rezept auf der Buch-CD.

Beispiel (*browserweiche2.html*):

```
01 <html>
02   <script language="JavaScript">
03     <!--
04     // Erste Identifikation : Betriebssystem
05     if (navigator.appVersion.indexOf('Mac') != -1) { // Mac OS
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("Mac OS , Netscape, Version <= 4.7, 16
19                   Farben, Auflösung 480X600");
20                   break; //Farbtiefe 16 Farben
21                 case 8: // 256 Farben
22                   document.write("Mac OS , Netscape, Version <= 4.7, 256
23                   Farben, Auflösung 480X600");
24                   break; //Farbtiefe 256 Farben
25                 case 16: // 65536 Farben
26                   document.write("Mac OS , Netscape, Version <= 4.7, 65536
27                   Farben, Auflösung 480X600");
28                   break; //Farbtiefe 65536 Farben
29                 default: // 16777216 + mehr Farben
30                   document.write("Mac OS , Netscape, Version <= 4.7, 16777216
31                   + mehr Farben, Auflösung 480X600");
32             }
33           }
34         }
35       }
36     }
37     //
38   }
39   </script>
40 </html>
```

Listing 33: Eine sehr umfangreich aufgeschlüsselte Browserweiche

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

```

27             break; //Farbtiefe 16777216 + mehr Farben
28         }
29         break; // Auflösung 480X600
30     case 600:
31         // Identifikation der Farbtiefe
32         switch(screen.colorDepth) {
33             case 4: // 16 Farben
34                 document.write("Mac OS , Netscape, Version <= 4.7, ↵
16 Farben, Auflösung 600X800");
35                 break; //Farbtiefe 16 Farben
36             case 8: // 256 Farben
37                 document.write("Mac OS , Netscape, Version <= ↵
4.7, 256 Farben, Auflösung 600X800");
38                 break; //Farbtiefe 256 Farben
39             case 16: // 65536 Farben
40                 document.write("Mac OS , Netscape, Version <= 4.7, ↵
65536 Farben, Auflösung 600X800");
41                 break; //Farbtiefe 65536 Farben
42             default: // 16777216 + mehr Farben
43                 document.write("Mac OS , Netscape, Version <= 4.7, ↵
16777216 + mehr Farben, Auflösung 600X800");
44                 break; //Farbtiefe 16777216 + mehr Farben
45         }
46         break; // Auflösung 600X800
47     case 768:
48         // Identifikation der Farbtiefe
49         switch(screen.colorDepth) {
50             case 4: // 16 Farben
51                 document.write("Mac OS , Netscape, Version <= 4.7, ↵
16 Farben, Auflösung 768X1024");
52                 break; //Farbtiefe 16 Farben
53             case 8: // 256 Farben
54                 document.write("Mac OS , Netscape, Version <= ↵
4.7, 256 Farben, Auflösung 768X1024");
55                 break; //Farbtiefe 256 Farben
56             case 16: // 65536 Farben
57                 document.write("Mac OS , Netscape, Version <= 4.7, ↵
65536 Farben, Auflösung 768X1024");
58                 break; //Farbtiefe 65536 Farben
59             default: // 16777216 + mehr Farben
60                 document.write("Mac OS , Netscape, Version <= 4.7, ↵
16777216 + mehr Farben, Auflösung 768X1024");
61                 break; //Farbtiefe 16777216 + mehr Farben
62         }
63         break; // Auflösung 768X1024
64     case 1024:
65         // Identifikation der Farbtiefe
66         switch(screen.colorDepth) {
67             case 4: // 16 Farben
68                 document.write("Mac OS , Netscape, Version <= 4.7, ↵
16 Farben, Auflösung 1024X1200");

```

Listing 33: Eine sehr umfangreich aufgeschlüsselte Browserweiche (Forts.)

```

69             break; //Farbtiefe 16 Farben
70             case 8: // 256 Farben
71                 document.write("Mac OS , Netscape, Version <= 4.7, 256 Farben, Auflösung 1024X1200");
72             break; //Farbtiefe 256 Farben
73             case 16: // 65536 Farben
74                 document.write("Mac OS , Netscape, Version <= 4.7, 65536 Farben, Auflösung 1024X1200");
75             break; //Farbtiefe 65536 Farben
76             default: // 16777216 + mehr Farben
77                 document.write("Mac OS , Netscape, Version <= 4.7, 16777216 + mehr Farben, Auflösung 1024X1200");
78             break; //Farbtiefe 16777216 + mehr Farben
79         }
80         break; // Auflösung 1024X1200
81     default:
82         // Identifikation der Farbtiefe
83         switch(screen.colorDepth) {
84             case 4: // 16 Farben
85                 document.write("Mac OS , Netscape, Version <= 4.7, 16 Farben, Defaultauflösung");
86             break; //Farbtiefe 16 Farben
87             case 8: // 256 Farben
88                 document.write("Mac OS , Netscape, Version <= 4.7, 256 Farben, Defaultauflösung");
89             break; //Farbtiefe 256 Farben
90             case 16: // 65536 Farben
91                 document.write("Mac OS , Netscape, Version <= 4.7, 65536 Farben, Defaultauflösung");
92             break; //Farbtiefe 65536 Farben
93             default: // 16777216 + mehr Farben
94                 document.write("Mac OS , Netscape, Version <= 4.7, 16777216 + mehr Farben, Defaultauflösung");
95             break; //Farbtiefe 16777216 + mehr Farben
96         }
97     }
98 } // Ende Navigator <= 4.7
...
...
...
1740 } // Ende Opera
1741 else if(navigator.appName.indexOf('Konqueror')!=-1) { // Konqueror
1742     // Identifikation der Auflösung
1743     // Hier sind vier plus Defaultwert als Auflösungen realisiert.
1744     switch(screen.height) {
1745         case 480:
1746             // Identifikation der Farbtiefe
1747             switch(screen.colorDepth) {
1748                 case 4: // 16 Farben
1749                     document.write("Linux, Konqueror, 16 Farben, Auflösung 480X600");

```

Listing 33: Eine sehr umfangreich aufgeschlüsselte Browserweiche (Forts.)

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

HTML

Events

```

1750             break; //Farbtiefe 16 Farben
1751             case 8: // 256 Farben
1752                 document.write("Linux, Konqueror, 256 Farben, ➡
Auflösung 480X600");
1753                 break; //Farbtiefe 256 Farben
1754             case 16: // 65536 Farben
1755                 document.write("Linux, Konqueror, 65536 Farben, ➡
Auflösung 480X600");
1756                 break; //Farbtiefe 65536 Farben
1757             default: // 16777216 + mehr Farben
1758                 document.write("Linux, Konqueror, 16777216 + ➡
mehr Farben, Auflösung 480X600");
1759                 break; //Farbtiefe 16777216 + mehr Farben
1760             }
1761             break; // Auflösung 480X600
1762         case 600:
1763             // Identifikation der Farbtiefe
1764             switch(screen.colorDepth) {
1765                 case 4: // 16 Farben
1766                     document.write("Linux, Konqueror, 16 Farben, ➡
Auflösung 600X800");
1767                     break; //Farbtiefe 16 Farben
1768                 case 8: // 256 Farben
1769                     document.write("Linux, Konqueror, 256 Farben, ➡
Auflösung 600X800");
1770                     break; //Farbtiefe 256 Farben
1771                 case 16: // 65536 Farben
1772                     document.write("Linux, Konqueror, 65536 Farben, ➡
Auflösung 600X800");
1773                     break; //Farbtiefe 65536 Farben
1774                 default: // 16777216 + mehr Farben
1775                     document.write("Linux, Konqueror, 16777216 + ➡
mehr Farben, Auflösung 600X800");
1776                     break; //Farbtiefe 16777216 + mehr Farben
1777             }
1778             break; // Auflösung 600X800
1779         case 768:
1780             // Identifikation der Farbtiefe
1781             switch(screen.colorDepth) {
1782                 case 4: // 16 Farben
1783                     document.write("Linux, Konqueror, 16 Farben, ➡
Auflösung 768X1024");
1784                     break; //Farbtiefe 16 Farben
1785                 case 8: // 256 Farben
1786                     document.write("Linux, Konqueror, 256 Farben, ➡
Auflösung 768X1024");
1787                     break; //Farbtiefe 256 Farben
1788                 case 16: // 65536 Farben
1789                     document.write("Linux, Konqueror, 65536 Farben, ➡
Auflösung 768X1024");
1790                     break; //Farbtiefe 65536 Farben

```

Listing 33: Eine sehr umfangreich aufgeschlüsselte Browserweiche (Forts.)

```

1791             default: // 16777216 + mehr Farben
1792                 document.write("Linux, Konqueror, 16777216 + 
mehr Farben, Auflösung 768X1024");
1793                 break; //Farbtiefe 16777216 + mehr Farben
1794             }
1795             break; // Auflösung 768X1024
1796         case 1024:
1797             // Identifikation der Farbtiefe
1798             switch(screen.colorDepth) {
1799                 case 4: // 16 Farben
1800                     document.write("Linux, Konqueror, 16 Farben, 
Auflösung 1024X1200");
1801                     break; //Farbtiefe 16 Farben
1802                 case 8: // 256 Farben
1803                     document.write("Linux, Konqueror, 256 Farben, 
Auflösung 1024X1200");
1804                     break; //Farbtiefe 256 Farben
1805                 case 16: // 65536 Farben
1806                     document.write("Linux, Konqueror, 65536 Farben, 
Auflösung 1024X1200");
1807                     break; //Farbtiefe 65536 Farben
1808                 default: // 16777216 + mehr Farben
1809                     document.write("Linux, Konqueror, 16777216 + 
mehr Farben, Auflösung 1024X1200");
1810                     break; //Farbtiefe 16777216 + mehr Farben
1811             }
1812             break; // Auflösung 1024X1200
1813         default:
1814             // Identifikation der Farbtiefe
1815             switch(screen.colorDepth) {
1816                 case 4: // 16 Farben
1817                     document.write("Linux, Konqueror, 16 Farben, 
Defaultauflösung");
1818                     break; //Farbtiefe 16 Farben
1819                 case 8: // 256 Farben
1820                     document.write("Linux, Konqueror, 256 Farben, 
Defaultauflösung");
1821                     break; //Farbtiefe 256 Farben
1822                 case 16: // 65536 Farben
1823                     document.write("Linux, Konqueror, 65536 Farben, 
Defaultauflösung");
1824                     break; //Farbtiefe 65536 Farben
1825                 default: // 16777216 + mehr Farben
1826                     document.write("Linux, Konqueror, 16777216 + 
mehr Farben, Defaultauflösung");
1827                     break; //Farbtiefe 16777216 + mehr Farben
1828             }
1829         }
1830     } // Ende Konqueror
1831     else {
1832         // Identifikation der Auflösung

```

Listing 33: Eine sehr umfangreich aufgeschlüsselte Browserweiche (Forts.)

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

HTML

Events

```

1833         // Hier sind vier plus Defaultwert als Auflösungen realisiert.
1834         switch(screen.height) {
1835             case 480:
1836                 // Identifikation der Farbtiefe
1837                 switch(screen.colorDepth) {
1838                     case 4: // 16 Farben
1839                         document.write("Anderes Betriebssystem, Anderer
Browser, 16 Farben, Auflösung 480X600");
1840                         break; //Farbtiefe 16 Farben
1841                     case 8: // 256 Farben
1842                         document.write("Anderes Betriebssystem, Anderer
Browser, 256 Farben, Auflösung 480X600");
1843                         break; //Farbtiefe 256 Farben
1844                     case 16: // 65536 Farben
1845                         document.write("Anderes Betriebssystem, Anderer
Browser, 65536 Farben, Auflösung 480X600");
1846                         break; //Farbtiefe 65536 Farben
1847                     default: // 16777216 + mehr Farben
1848                         document.write("Anderes Betriebssystem, Anderer
Browser, 16777216 + mehr Farben, Auflösung 480X600");
1849                         break; //Farbtiefe 16777216 + mehr Farben
1850                 }
1851                 break; // Auflösung 480X600
1852             case 600:
1853                 // Identifikation der Farbtiefe
1854                 switch(screen.colorDepth) {
1855                     case 4: // 16 Farben
1856                         document.write("Anderes Betriebssystem, Anderer
Browser, 16 Farben, Auflösung 600X800");
1857                         break; //Farbtiefe 16 Farben
1858                     case 8: // 256 Farben
1859                         document.write("Anderes Betriebssystem, Anderer
Browser, 256 Farben, Auflösung 600X800");
1860                         break; //Farbtiefe 256 Farben
1861                     case 16: // 65536 Farben
1862                         document.write("Anderes Betriebssystem, Anderer
Browser, 65536 Farben, Auflösung 600X800");
1863                         break; //Farbtiefe 65536 Farben
1864                     default: // 16777216 + mehr Farben
1865                         document.write("Anderes Betriebssystem, Anderer
Browser, 16777216 + mehr Farben, Auflösung 600X800");
1866                         break; //Farbtiefe 16777216 + mehr Farben
1867                 }
1868                 break; // Auflösung 600X800
1869             case 768:
1870                 // Identifikation der Farbtiefe
1871                 switch(screen.colorDepth) {
1872                     case 4: // 16 Farben
1873                         document.write("Anderes Betriebssystem, Anderer
Browser, 16 Farben, Auflösung 768X1024");
1874                         break; //Farbtiefe 16 Farben

```

Listing 33: Eine sehr umfangreich aufgeschlüsselte Browserweiche (Forts.)

```

1875             case 8: // 256 Farben
1876                 document.write("Anderes Betriebssystem, Anderer
Browser, 256 Farben, Auflösung 768X1024");
1877                 break; //Farbtiefe 256 Farben
1878             case 16: // 65536 Farben
1879                 document.write("Anderes Betriebssystem, Anderer
Browser, 65536 Farben, Auflösung 768X1024");
1880                 break; //Farbtiefe 65536 Farben
1881             default: // 16777216 + mehr Farben
1882                 document.write("Anderes Betriebssystem, Anderer
Browser, 16777216 + mehr Farben, Auflösung 768X1024");
1883                 break; //Farbtiefe 16777216 + mehr Farben
1884         }
1885         break; // Auflösung 768X1024
1886     case 1024:
1887         // Identifikation der Farbtiefe
1888         switch(screen.colorDepth) {
1889             case 4: // 16 Farben
1890                 document.write("Anderes Betriebssystem, Anderer
Browser, 16 Farben, Auflösung 1024X1200");
1891                 break; //Farbtiefe 16 Farben
1892             case 8: // 256 Farben
1893                 document.write("Anderes Betriebssystem, Anderer
Browser, 256 Farben, Auflösung 1024X1200");
1894                 break; //Farbtiefe 256 Farben
1895             case 16: // 65536 Farben
1896                 document.write("Anderes Betriebssystem, Anderer
Browser, 65536 Farben, Auflösung 1024X1200");
1897                 break; //Farbtiefe 65536 Farben
1898             default: // 16777216 + mehr Farben
1899                 document.write("Anderes Betriebssystem, Anderer
Browser, 16777216 + mehr Farben, Auflösung 1024X1200");
1900                 break; //Farbtiefe 16777216 + mehr Farben
1901             }
1902         break; // Auflösung 1024X1200
1903     default:
1904         // Identifikation der Farbtiefe
1905         switch(screen.colorDepth) {
1906             case 4: // 16 Farben
1907                 document.write("Anderes Betriebssystem, Anderer
Browser, 16 Farben, Defaultauflösung");
1908                 break; //Farbtiefe 16 Farben
1909             case 8: // 256 Farben
1910                 document.write("Anderes Betriebssystem, Anderer
Browser, 256 Farben, Defaultauflösung");
1911                 break; //Farbtiefe 256 Farben
1912             case 16: // 65536 Farben
1913                 document.write("Anderes Betriebssystem, Anderer
Browser, 65536 Farben, Defaultauflösung");
1914                 break; //Farbtiefe 65536 Farben
1915             default: // 16777216 + mehr Farben

```

Listing 33: Eine sehr umfangreich aufgeschlüsselte Browserweiche (Forts.)

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

```
1916                                     document.write("Anderes Betriebssystem, Anderer ↵
Browser, 16777216 + mehr Farben, Defaultauflösung");
1917                                     break; //Farbtiefe 16777216 + mehr Farben
1918                                     }
1919     }
1920 } // Ende anderer Browser
1921 } // Ende anderes Betriebssystem
1922 //-->
1923 </script>
1924 <body>
1925 </body>
1926 </html>
```

Listing 33: Eine sehr umfangreich aufgeschlüsselte Browserweiche (Forts.)

In diesem Kapitel werden Rezepte vorgestellt, die sich mit grundlegenden Fragen bezüglich der Syntax von JavaScript sowie dem grundsätzlichen Umgang mit Objekten beschäftigen.

15 Wie kann ich einen Test auf Unendlichkeit durchführen?

In diversen Situationen kann bei Berechnungen der Fall auftreten, dass deren Ergebnis nicht mehr definiert ist, etwa bei der Division durch die Zahl 0 oder einem Ergebnis, das den Wertebereich der numerischen Daten in JavaScript übersteigt. Dann liefert JavaScript den definierten Wert `Infinity`, auf den Sie aber nicht direkt prüfen können¹. 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 34: Test auf Unendlichkeit

Wenn das Argument `NaN`, 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 – ein unabdingbares Feature. Allerdings kann es natürlich durchaus vorkommen, dass ein Programmierer bei der Programmierung nicht alle Situationen erkennt, die `Infinity` auslösen.

Beispiel (*finite.html*):

```
01 <html>
02 <script language="JavaScript">
03 a = Math.round(Math.random()*3);
04 document.write(a+"<hr>");
05 b = Math.round(Math.random()*10);
06 document.write(b+"<hr>");
07 if(!isFinite(b/a)){
```

Listing 35: Einsatz von `isFinite()`

1. So etwas geht nicht: `if(a == Infinity) ...`

```
08 document.write("Der Divisor ist 0");  
09 }  
10 else {  
11 document.write("Das Ergebnis der Division von " + b + " geteilt durch " + a + "  
ist " + b/a);  
12 }  
13 </script>  
14 </html>
```

Listing 35: Einsatz von `isFinite()` (Forts.)

In den Zeilen 3 und 5 werden zufällig zwei Zahlen berechnet, die in der Folge durcheinander geteilt werden sollen.

Tip

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 mathematisch unendlich, und das resultiert in JavaScript im wohldefinierten Wert `Infinity`. In Zeile 7 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).



Abbildung 16: Die Division konnte durchgeführt werden.

Zeigt `isFinite(b/a)` jedoch, dass die Evaluierung von dem Ausdruck keinen endlichen Wert ergibt, wird eine Fehlermeldung ausgegeben.

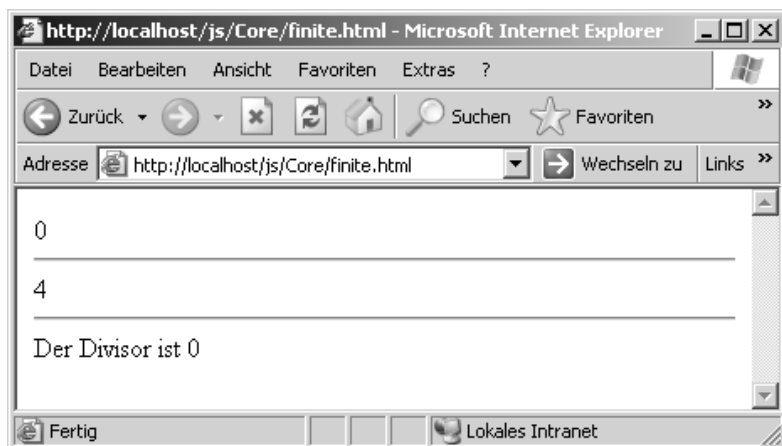


Abbildung 17: Der Divisor ist 0.

16 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`). Der Wert `NaN` kann aber nicht direkt in einem Vergleich verwendet³, aber über die Funktion `isNaN([Ausdruck])` getestet werden. Diese Funktion prüft den übergebenen Wert, ob er ein erlaubter numerischer Wert ist (Rückgabe `true`) oder nicht (`false`). Allgemein setzt man die Funktion meist in Verbindung mit einer `if`-Abfrage ein. Etwa so:

```
if(isNaN(a)) ...
```

Listing 36: 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 logisch falsch ein. Die Funktion `isFinite()` liefert in den meisten Fällen logisch gesehen `true`, wenn ein zu testender Ausdruck den gewünschten Wert (endlich) hat, während `isNaN()` logisch gesehen `true` ergibt, wenn der zu testende Ausdruck den meistens gewünschten Wert (eine Zahl) *nicht* ergibt (wobei das natürlich explizit von der Art der Verwendung abhängt).

2. Not a Number.

3. So etwas geht nicht: `if(a == NaN) ...`

Beispiel (*isnan.html*):

```
01 <html>
02 <script language="JavaScript">
03 a = prompt("Geben Sie eine Zahl ein");
04 if(isNaN(a)){
05     document.write("Ihre Eingabe war keine Zahl");
06 }
07 else {
08     document.write("Das Ergebnis der Multiplikation von 5 mal " + a + " ist " +
09 5*a);
10 }
11 </script>
12 </html>
```

Listing 37: Die Verwendung von *isNaN()*

Tipp

Die Funktion `isNaN()` kann nicht nur auf zusammengesetzte Ausdrücke, sondern auch auf Literale und einzelne Variablen (wie in dem Beispiel) angewendet werden.

In Zeile 3 wird vom Anwender die Eingabe einer Zahl verlangt. Gibt dieser eine Zahl ein, liefert die Überprüfung in Zeile 4 den Wert `false` (die Evaluierung ergibt *nicht* NaN), und die Ausgabe in Zeile 8 erfolgt.

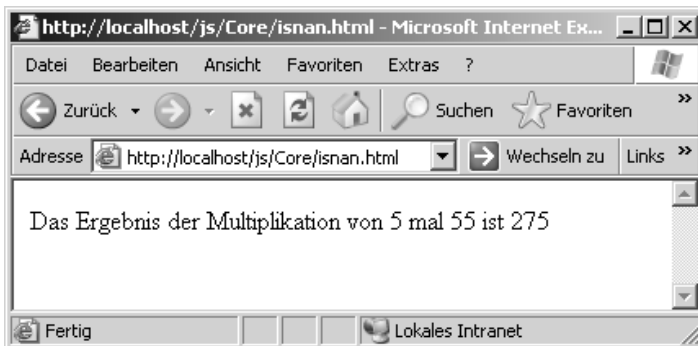


Abbildung 18: Das Ergebnis der Multiplikation der Benutzereingabe mit einem konstanten Wert

Nimmt ein Anwender jedoch eine nichtnumerische Eingabe vor, liefert `isNaN()` den Wert `true`, und damit wird der `if`-Block ausgeführt.

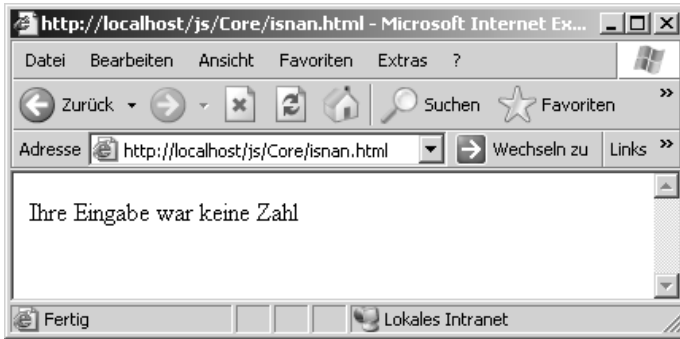


Abbildung 19: Der Anwender hat keine Zahl eingegeben.

17 Festlegung des Variablen- oder eines Literaltyps bzw. des Funktionsrückgabetyps

Ein Literal hat von Natur aus einen Datentyp. Der Wert selbst repräsentiert diesen, und Sie können und müssen sich dazu keine weiteren Gedanken machen. 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 38: Die Wertzuweisung legt den Datentyp fest.

Vollkommen analog ergibt sich der Datentyp eines Rückgabewerts alleine daraus, welchen Wert eine Funktion zurückgibt.

Beispiele:

```
function test(){
    return 2;
}
```

Listing 39: Der Datentyp des Rückgabewerts wird einfach durch den Wert festgelegt, den die Funktion zurückgibt.

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

So bequem dieses automatische Verhalten auch ist – es gibt auch keine andere Möglichkeit, 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 131).

18 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 Rückgabebetyp einer Funktion jedoch nur indirekt durch Wertzuweisung fest (nicht über ein Schlüsselwort, wie es in den meisten anderen Sprachen üblich ist⁴). 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 beziehungsweise den Rückgabewert einer Funktion festlegen?« auf Seite 129). Oft will oder muss man jedoch zur Laufzeit explizit herausbekommen, von welchem Typ eine Variable, ein Literal⁵ oder der Rückgabewert einer Funktion ist. Sie können mit Hilfe des Toplevel-Operators `typeof` 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
- ▶ `object`

als Ergebnis angegeben werden. Das nachfolgende Beispiel zeigt, wie der Datentyp eines Ausdrucks angezeigt werden kann.

4. Etwa wie in Java mit `int a;` oder `double b;`.
5. Wobei das bei einem Literal in JavaScript trivial ist.

Beispiel (*typeof1.html*):

```

01 <html>
02 <script language="JavaScript">
03 var a = 42;
04 var b = true;
05 var c = new Date();
06 var d;
07 var e = "Flo";
08 var f = 1.234567;
09 function meinFktion() {
10 return 1;
11 }
12 document.write("<table border>");
13 document.write("<tr><td>Wert von a</td><td>" + a + "</td><td>Datentyp</td><td>"
14 + typeof a + "</td></tr>");
15 document.write("<tr><td>Wert von b</td><td>" + b + "</td><td>Datentyp</td><td>"
16 + typeof b + "</td></tr>");
17 document.write("<tr><td>Wert von c</td><td>" + c + "</td><td>Datentyp</td><td>"
18 + typeof c + "</td></tr>");
19 document.write("<tr><td>Wert von d</td><td>" + d + "</td><td>Datentyp</td><td>"
20 + typeof d + "</td></tr>");
21 document.write("<tr><td>Wert von e</td><td>" + e + "</td><td>Datentyp</td><td>"
22 + typeof e + "</td></tr>");
23 document.write("<tr><td>Wert von f</td><td>" + f + "</td><td>Datentyp</td><td>"
24 + typeof f + "</td></tr>");
25 document.write("<tr><td>Wert von dem Funktionsaufruf meinFktion()</td><td>" +
meinFktion()
26 + "</td><td>Datentyp</td><td>" + typeof meinFktion() + "</td></tr>");
27 document.write("<tr><td>Wert von meinFktion</td><td>" + meinFktion
28 + "</td><td>Datentyp</td><td>" + typeof meinFktion + "</td></tr>");
29 document.write("</table>");
30 </script>
31 </html>

```

Listing 40: Die Ausgabe verschiedener Datentypen mit *typeof*

In dem Listing wird dynamisch eine Tabelle geschrieben und mit *typeof* der Datentyp verschiedener Variablen und Rückgabewerte jeweils ausgegeben. Beachten Sie in dem Beispiel insbesondere die letzten beiden Ausgaben (Zeile 25 und 27). Dort wird einmal die Funktion mit Klammern (Zeile 25) und einmal ohne Klammern verwendet (Zeile 27). Der Datentyp mit Klammern ist derjenige des Rückgabewerts der Funktion, der Datentyp des Aufrufs ohne Klammern ist *function* (eine Funktionsreferenz).

19 Wie kann ich den Typ einer Variablen bzw. 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 keiner lose typisierten Sprache können Sie nicht verhindern, dass eine simple Wertzuweisung

The screenshot shows a Netscape browser window with the address bar set to `http://localhost/js/Core/typeof1.html`. The main content area displays a table with the following data:

Wert von a	42	Datentyp	number.
Wert von b	true	Datentyp	boolean.
Wert von c	Thu May 27 2004 12:20:44 GMT+0200	Datentyp	object.
Wert von d	undefined	Datentyp	undefined.
Wert von e	Flo	Datentyp	string
Wert von f	1.234567	Datentyp	number.
Wert von dem Funktionsaufruf meinFktion()	1	Datentyp	number.
Wert von meinFktion	function meinFktion() { return 1; }	Datentyp	function.

Abbildung 20: Wert und Datentyp gegenübergestellt

den Typ einer Variablen anpasst (siehe Rezept »Wie kann ich den Typ einer Variablen, eines Literals beziehungsweise den Rückgabewert einer Funktion festlegen?« auf Seite 130).

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 Wert⁶. Die nachfolgenden Rezepte decken deshalb nicht alle denkbaren Situationen ab, aber sie helfen Ihnen, die wichtigsten Fälle unter Kontrolle zu halten. Die Rezepte bauen mit Hilfe des Operators `typeof` sowie der Funktionen `parseFloat()` und `parseInt()` Schutzmechanismen auf, die offensichtliche Fehlzusweisungen 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 nichtnumerischen 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 wandelt nur eine übergebene Zeichenkette in eine Ganzzahl um und gibt diese als Ergebnis zurück. Auch bei einer Gleitkommazahl wird der Nachkomma-Anteil abgeschnitten. Der Operator `typeof` ist ein unitärer Operator, der den Datentyp eines Operanden als String zurückliefert (siehe Rezept

6. Es sei denn, in dem String waren nur Zahlen.

»Wie kann ich den Typ einer Variablen, eines Literals beziehungsweise den Rückgabewert einer Funktion bestimmen?« auf Seite 129).

Beachten Sie bei den nachfolgenden Rezepten, dass wir hier mit einer Abart 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 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 offen gelegt. 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 einen Variablenwert etwa 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 via geeigneter 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 geliefert 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.

20 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 41: Konvertieren in einen String

Der Trick basiert einfach darauf, dass für den Rückgabewert ein Leerstring mit dem eigentlichen Wert verbunden wird. Der Trick funktioniert mit sämtlichen Datentypen in JavaScript (inklusive object).

Beispiel (*stringfestlegung.html*):

```
01 <html>  
02 <script language="JavaScript">  
03 function setString(a) {  
04     return ""+a;  
05 }  
06 b = 43;  
07 document.write("Datentyp von b: " + typeof b + "<br>");  
08 b = setString(b);  
09 document.write("Datentyp von b: " + typeof b + "<br>");  
10 b = new Array();  
11 document.write("Datentyp von b: " + typeof b + "<br>");  
12 b = setString(b);  
13 document.write("Datentyp von b: " + typeof b + "<br>");  
14 </script>  
15 </html>
```

Listing 42: 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.

21 Wie kann ich eine Festlegung auf number realisieren?

Mit der nachfolgenden Funktion stellen Sie sicher, dass ein Übergabewert zu einem `number`-Datentyp wird:

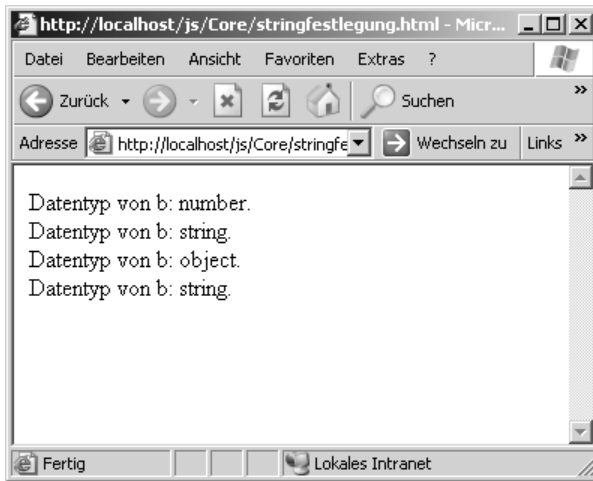


Abbildung 21: Aus number und object wird string.

```
function setNumber(a) {
    return parseFloat(a);
}
```

Listing 43: Konvertieren zu einem number-Datentyp

Der Übergabewert wird nach dem Auftreten eines numerischen Wertes geparkt 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.

Beispiel (*numberfestlegung.html*):

```
01 <html>
02 <script language="JavaScript">
03 function setNumber(a) {
04     return parseFloat(a);
05 }
06 function setNumber(a) {
07     if(!isNaN(parseFloat(a))) return parseFloat(a);
08     else return 0;
09 }
10 b = new Date();
11 document.write("Datentyp von b: " + typeof b + ". Der Wert von b: " + b + "<br>")
12 b = setNumber(b);
13 document.write("Datentyp von b: " + typeof b + ". Der Wert von b: " + b + "<br>")
14 c = "Babelfish";
```

Listing 44: Festlegung eines number-Datentyps

```

15 document.write("Datentyp von c: " + typeof c + ". Der Wert von c: " + c + "<br>")
16 c = setNumber(c);
17 document.write("Datentyp von c: " + typeof c + ". Der Wert von c: " + c + "<br>")
18 d = "3.14 und etwas mehr";
19 document.write("Datentyp von d: " + typeof d + ". Der Wert von d: " + d + "<br>")
20 d = setNumber(d);
21 document.write("Datentyp von d: " + typeof d + ". Der Wert von d: " + d + "<br>")
22 e = new Array();
23 document.write("Datentyp von e[1]: " + typeof e[1] + ". Der Wert von e[1]: " +
e[1] + "<br>")
24 e[1] = setNumber(e[1]);
25 document.write("Datentyp von e[1]: " + typeof e[1] + ". Der Wert von e[1]: " +
e[1] + "<br>")
26 </script>
27 </html>

```

Listing 44: Festlegung eines number-Datentyps (Forts.)

In dem Beispiel werden verschiedene Datentypen über die besprochene Funktion `setNumber()` jeweils zu einem String gecastet.

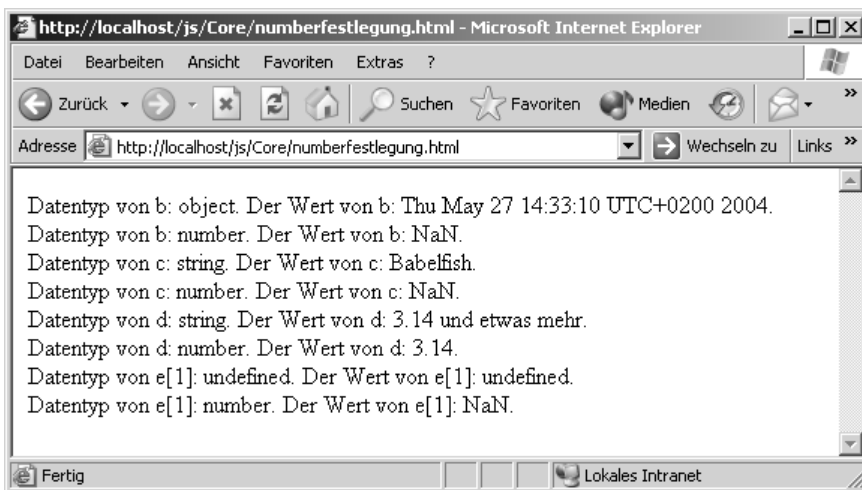


Abbildung 22: In jedem Fall gibt es einen number-Datentyp.

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 45: Konvertieren zu einem 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.

Tipp

Statt `parseFloat()` können Sie auch die Toplevel-Funktion `Number([Objekt/Ausdruck])` verwenden, um den Inhalt eines als Parameter übergebenen Objektes 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. Implementiert ist die Funktion bereits seit JavaScript 1.0, aber erst seit JavaScript 1.1 ist die Funktion für alle Objekte einsetzbar. `Number()` führt zwar auch Operationen aus, aber nur, wenn diese als Ergebnis eindeutig eine Zahl ergeben, es also eindeutig identifizierbare arithmetische Ausdrücke sind.

22 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 Sinn⁷, 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 46: Konvertieren zu einem boolean-Datentyp

Beispiel (*booleanfestlegung.html*):

```
01 <html>
02 <script language="JavaScript">
03 function setBoolean(a) {
04   if((parseInt(a)==0) || (a==true)) return true;
```

Listing 47: Festlegung eines boolean-Datentyps

- 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).

```
05 else return false;
06 }
07 b = new Date();
08 document.write("Datentyp von b: " + typeof b + ". Der Wert von b: " + b + "<br>")
09 b = setBoolean(b);
10 document.write("Datentyp von b: " + typeof b + ". Der Wert von b: " + b + "<br>")
11 c = true;
12 document.write("Datentyp von c: " + typeof c + ". Der Wert von c: " + c + "<br>")
13 c = setBoolean(c);
14 document.write("Datentyp von c: " + typeof c + ". Der Wert von c: " + c + "<br>")
15 d = "0 und etwas mehr";
16 document.write("Datentyp von d: " + typeof d + ". Der Wert von d: " + d + "<br>")
17 d = setBoolean(d);
18 document.write("Datentyp von d: " + typeof d + ". Der Wert von d: " + d + "<br>")
19 e = new Array();
20 document.write("Datentyp von e[1]: " + typeof e[1] + ". Der Wert von e[1]: " +
e[1] + "<br>")
21 e[1] = setBoolean(e[1]);
22 document.write("Datentyp von e[1]: " + typeof e[1] + ". Der Wert von e[1]: " +
e[1] + "<br>")
23 </script>
24 </html>
```

Listing 47: Festlegung eines boolean-Datentyps (Forts.)

In dem Beispiel werden verschiedene Datentypen über die besprochene Funktion `setBoolean()` jeweils zu einem String gecastet.

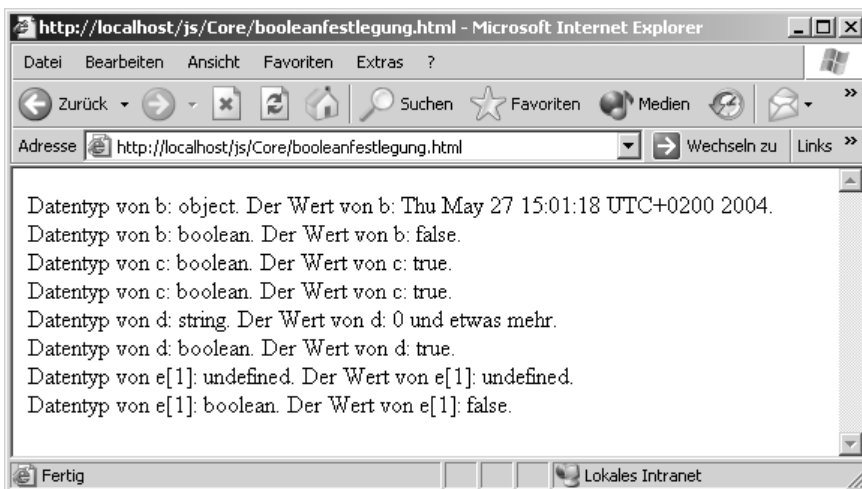


Abbildung 23: In jedem Fall entsteht ein boolean-Datentyp.

23 Wie kann ich einen Wertebereich 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 Wertebereich⁸ beziehungsweise eine untere oder obere Grenze⁹ festlegen. Ein vernünftiger Weg führt über einen indirekten Zugriff via 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 131). 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 <script language="JavaScript">
03 var a;
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 </html>
```

Listing 48: Festlegen eines numerischen Wertebereichs für eine Variable

Das Beispiel legt einen numerischen Wertebereich fest. In den Zeile 4 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 Wert für die Variable `a` zufällig generiert, der auch außerhalb des erlaubten Wertebereichs liegen kann.

-
8. 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.
 9. Eine Variable darf etwa nicht negativ sein.

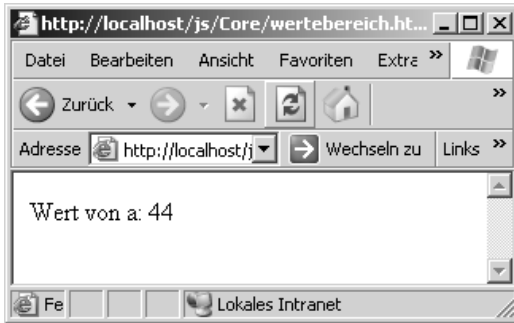


Abbildung 24: Der Wert ist innerhalb des Wertebereichs.

Liegt der zufällig generierte Wert außerhalb des gewünschten Wertebereichs, wird ein Defaultwert (hier der Wert 0) zugewiesen.

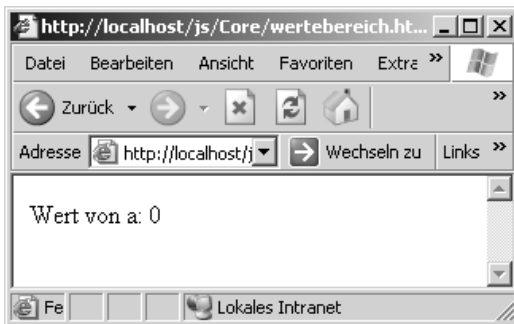


Abbildung 25: Ein Defaultwert wird zugewiesen.

Tip

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).

24 Wie kann ich testen, ob eine Variable definiert wurde?

In vielen Fällen ist es sinnvoll zu wissen, ob eine Variable bereits definiert wurde, denn 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.

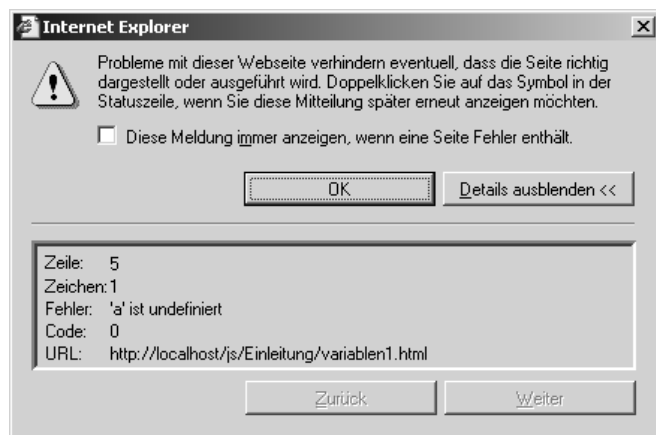


Abbildung 26: Der Interpreter hat erkannt, dass eine Variable undefiniert ist, und stoppt das Skript.

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, ist 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 49: Schema eines Tests, ob eine Variable definiert ist

Tipp

In der Regel ist es sinnvoll, einer noch undefinierten Variablen einfach einen Defaultwert zuzuweisen.

Beispiel:

```
if(typeof a=="undefined") a=0;
```

Listing 50: 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 51: 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 aufgrund eines Schreibfehlers oder ähnlicher Flüchtigkeitsfehler vor. Die Funktion ist aber nützlich für alle Fälle, in denen eine Variable existiert, aber halt undefiniert ist.

Beispiel (*variablendef1.html*):

```
01 <html>  
02 <script language="JavaScript">  
03 function getVarDef(a) {  
04     if(typeof a=="undefined") return false;  
05     else return true;  
06 }  
07 b = 1;  
08 var c;  
09 d= "";  
10 e = new Array();  
11 document.write("Variable b definiert: " + getVarDef(b) + "<br>")  
12 document.write("Variable c definiert: " + getVarDef(c) + "<br>")  
13 document.write("Variable d definiert: " + getVarDef(d) + "<br>")  
14 document.write("Variable e[1] definiert: " + getVarDef(e[1]) + "<br>")  
15 </script>  
16 </html>
```

Listing 52: Test, ob verschiedene Variablen definiert sind

In dem Beispiel wird die oben beschriebene Funktion zum Test auf Definition verwendet.

25 Wie kann ich eine Variable als lokal festlegen?

Lokale Variablen und globale Variablen besitzen verschiedene *Gültigkeitsbereiche* (engl. *scope*). Globale Variablen haben als Gültigkeitsbereich das gesamte Skript. 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.

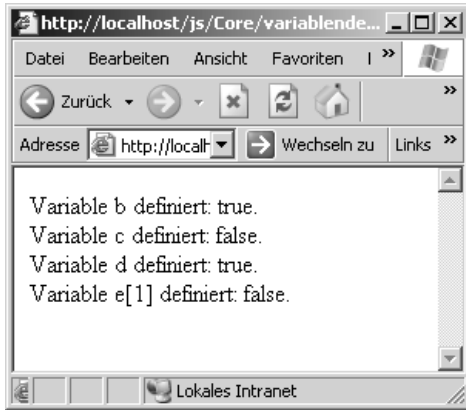


Abbildung 27: *b* und *d* sind definiert, *c* und *e[1]* haben einen undefinierten Zustand.

Tipp

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 53: Eine Funktion, die testet, ob eine Variable definiert wurde, und, falls nicht, einen Defaultwert zurückgibt, der dann der Variablen zugewiesen werden kann.

Beispiel:

```
function test(){
    var a = 5;
    ...
}
```

Listing 54: *Eine lokale Variable*

Dazu gibt es den Fall der schleifenlokalen Variablen, die nur im Rahmen einer Schleife definiert sind. Diese erfolgt schematisch so:

```
for(var i = 0; i < 5; i++) ...
```

Listing 55: *Einsatz einer schleifenlokalen Variablen*

Beispiel (*variablenlok.html*):

```
01 <html>
02 <script language="JavaScript">
03 i = 42;
04 function test(){
05 document.write("Wert der globalen Variablen i in der Methode test(): " + i + "  

06 ".<br>");
07 for(var i = 0;i< 5;i++)
08 document.write("Wert der schleifenlokalen Variablen i: " + i + ".<br>");
09 }
10 document.write("Wert der globalen Variablen i: " + i + ".<br>");
11 test();
12 document.write("Wert der globalen Variablen i: " + i + ".<br>");
13 </script>
14 </html>
```

*Listing 56: Eine globale Variable *i* wird von der schleifenlokalen Variablen nicht verändert, aber temporär verdeckt.*

In dem Beispiel wird in Zeile 6 eine schleifenlokale Variable verwendet.

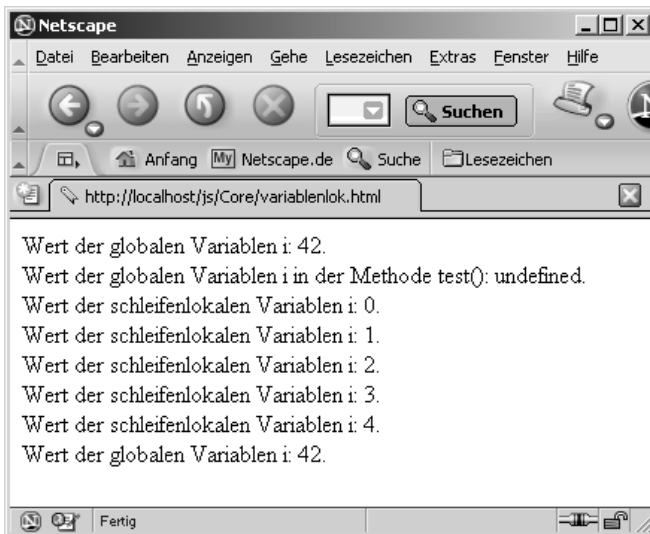


Abbildung 28: In der Funktion ist die globale Variable temporär auf undefined gesetzt.

26 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. 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.

27 Wie übergebe ich einer Funktion Werte?

Die Übergabe von Werten an eine Funktion kann auf drei Arten erfolgen:

1. Über globale Variablen
2. Über eine vorgegebene Parameterliste
3. 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

Wenn es sich vermeiden lässt, sollte man keine globalen Variablen einsetzen.

Wenn Sie in der Funktionsdeklaration Parameter spezifizieren, deklarieren sie mit den angegebenen Namen lokal in der Funktion verfügbare Variablen. Mehrere Parameter werden einfach durch Kommata getrennt. Wie in JavaScript üblich, können Sie dabei keinen Datentyp angeben.

Beispiel:

```
function test(i, j, k){  
    ...  
}
```

Listing 57: Die Deklaration von einer Funktion mit drei Parametern

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

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 58: 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 angegeben haben. Bei fehlenden Übergabewerten wird die nicht gefüllte lokale Variable als `undefined` betrachtet, und bei zu vielen Argumenten werden überschüssige Argumente einfach ignoriert.

Beispiel (*fktnitPara1.html*):

```
01 <html>
02 <script language="JavaScript">
03 function test(i, j, k){
04 document.write(i + "<br>");
05 document.write(j + "<br>");
06 document.write(k + "<hr>");
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 </html>
```

Listing 59: Eine Funktion mit drei Parametern, die mit verschiedener Anzahl an Argumenten aufgerufen wird

Die Funktion `test()` wird in der Zeile 3 bis 7 mit drei Übergabeparametern definiert. In Zeile 8 wird die Funktion jedoch ohne Parameter, in Zeile 9 mit einem, in Zeile 10 mit zwei und in Zeile 11 und 12 mit vier Parametern aufgerufen. Dennoch funktionieren die Aufrufe.

Offensichtlich unterstützt JavaScript Funktionen, die eine unterschiedliche Anzahl an Argumenten entgegennehmen. Dieses Verhalten kann man noch spezifischer ausnutzen, denn JavaScript stellt ein *Argumentendatenfeld* bzw. *Argumenten-Array* bereit. Dieses ist ein dynamisches Datenfeld, dessen Größe sich aufgrund der Anzahl der Argumente beim Funktionsaufruf ergibt. Der Zugriff auf das Datenfeld erfolgt schematisch so:

```
[Funktionsname].arguments[index]
```

Listing 60: Zugriff auf ein Funktionsargument über das Argumenten-Array

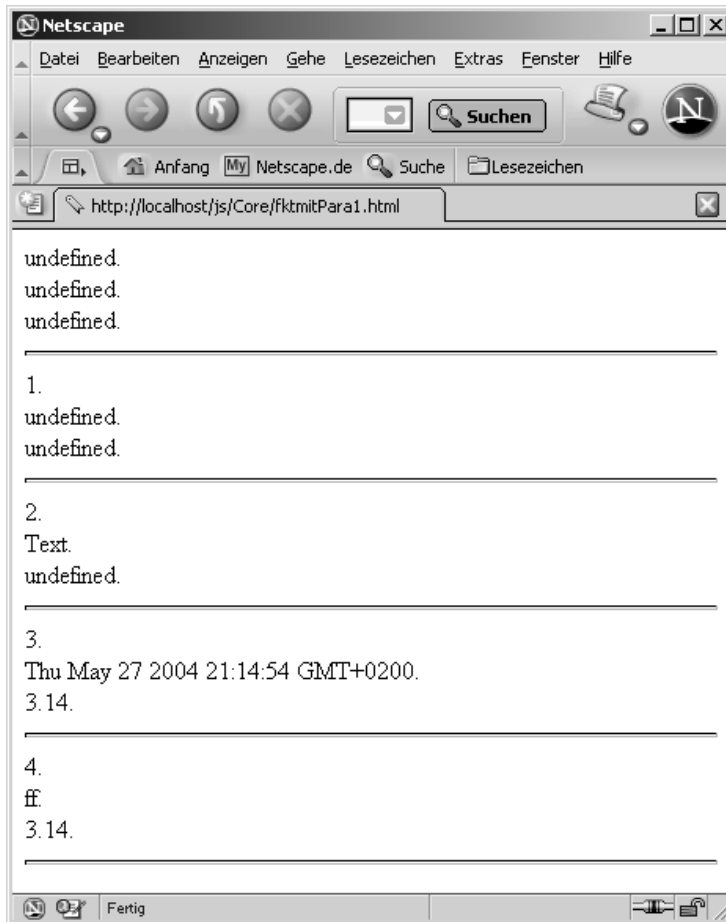


Abbildung 29: Der Aufruf mit einer unterschiedlichen Anzahl an Argumenten ist kein Problem.

Beispiel (*fktnitPara2.html*):

```

01 <html>
02 <script language="JavaScript">
03 function test(){
04   for(var i=0;i < test.arguments.length;i++)
05     document.write(test.arguments[i] + "<br>");
06 }
07 test();
08 test(1);
09 test(2,"Text");
10 test(3,new Date(), 3.14);
11 test(4,"ff", 3.14, 666);
12 </script>
13 </html>

```

Listing 61: Verschiedene Anzahlen der Argumente beim Aufruf der Funktion können über das Argumentendatenfeld verarbeitet werden.

Grund-
lagen

Core

Formu-
lare

Ausnah-
me

Arrays

Strings

Datum

Platt-
form

DHTML

Events

Auch in diesem Beispiel wird eine Funktion `test()` in der Zeile 3 bis 6 definiert, aber ohne Übergabeparameter. In Zeile 7 wird die Funktion jedoch ohne Parameter, in Zeile 8 mit einem, in Zeile 9 mit zwei, in Zeile 10 mit drei und in Zeile 12 mit vier Parametern aufgerufen. Dennoch funktionieren die Aufrufe. In der Funktion `test()` wird in Zeile 4 explizit das Argumentendatenfeld verwendet.



Abbildung 30: Je nach Anzahl der Argumente werden die verschiedenen Parameter verwendet.

28 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 62: 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 63: Das ist in JavaScript eine gültige Syntax für einen Defaultwert eines Parameters.

29 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 64: Eine Funktion mit Rückgabewert

Hinweis

Eine Funktion kann nur maximal einen Rückgabewert liefern.

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 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. Dieses Verhalten macht vor allem Sinn, wenn die Anweisung `return` nicht am Ende der Funktion steht, sondern innerhalb einer Kontrollflussstruktur, sodass unter gewissen Umständen die Funktion abgebrochen wird, in einer anderen Situation aber weiter abgearbeitet werden kann.

Beispiel (*rueckgabe.html*):

```
01 <html>  
02 <script language="JavaScript">  
03 var a;  
04 function rueck(a){  
05     if(a>50) {  
06         return new Date();  
07     }  
08     document.write("Der Wert des Parameters war zu klein, um das erste return  
auszulesen<br>");  
09     return new Date(2000,0,1);
```

Listing 65: Ein bedingtes return

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

HTML

Events

```

10 }
11 document.write("Datum: " + rueck(Math.random()*100));
12 </script>
13 </html>

```

Listing 65: Ein bedingtes return (Forts.)

In dem Beispiel wird in den Zeilen 4 bis 10 eine Funktion `rueck()` definiert, die je nach Übergabewert ein anderes `return` aufruft und entsprechend angepasste Rückgabewerte liefert. Ist der – in Zeile 11 zufällig generierte – Übergabewert größer als der Wert 50, liefert die Funktion das aktuelle Tagesdatum des Rechners. Dieses wird dann unmittelbar in der Ausgabe in Zeile 11 verwendet.

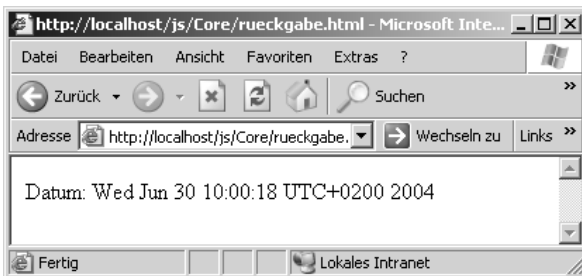


Abbildung 31: Der Rückgabewert des ersten return wird unmittelbar verwendet.

Ist der Übergabewert kleiner als 50 (genauer – oder gleich), 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 zurückgegeben¹⁰.

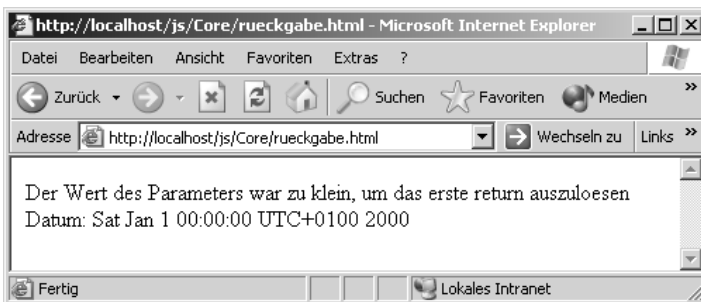


Abbildung 32: Der hartcodierte Wert wird zurückgegeben.

Tipp

Sie können `return` auch ohne Angabe eines zurückzugebenden Wertes verwenden. Dann fungiert `return` rein als Sprunganweisung.

10. Mit einem entsprechenden Konstruktor der Klasse `Date`

Achtung

Unerfreulicherweise ist es in JavaScript möglich, so genannten *unerreichbaren Code* zu schreiben. Darunter versteht man Codezeilen, die nie ausgeführt werden können (eine so genannte *unerreichbare Anweisung*). So ein Code kann etwa dadurch entstehen, dass nach einem auf jeden Fall ausgelösten `return` (oder einer anderen Sprunganweisung) weitere Anweisungen stehen¹. Solche unerreichbaren Anweisungen sind zwar eigentlich nicht schädlich in dem Sinn, dass sie etwas Falsches tun. Aber natürlich hat ein Programmierer die Anweisungen aus irgendeinem sinnvollen Grund notiert, und wenn er sich darauf verlässt, dass bestimmte Codezeilen ausgeführt werden, und sie werden einfach nicht erreicht, kann der Schaden mindestens genauso groß sein. In Programmiersprachen wie Java bemerkt der Compiler so eine Konstellation in der Regel rechtzeitig und erzeugt einen Fehler zur Kompilierzeit. In einer Skriptsprache wie JavaScript ist das naturgemäß aber nicht möglich, da Code 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 <script language="JavaScript">
03 function unerreicht(){
04   document.write("Vor dem return<br>");
05   return;
06   document.write("Nach dem return und ganz wichtig, aber leider nie erreichbar");
07 }
08 unerreicht();
09 document.write("Nach der Funktion mit der unerreichbaren Anweisung");
10 </script>
11 </html>
```

Listing 66: Eine Funktion mit einer unerreichbaren Anweisung

Die Funktion von Zeile 3 bis 7 löst in Zeile 5 ein `return` aus. Die nachfolgend notierte Anweisung in Zeile 6 wird niemals erreicht, und dennoch wird das Skript ohne Probleme ausgeführt.



Abbildung 33: Code mit unerreichbaren Anweisungen wird in JavaScript nicht moniert.

1. Unerreichbarer Code ist nicht damit zu verwechseln, dass bestimmter Code nie aufgerufen wird (etwa eine definierte Funktion, die aber nie verwendet wird).

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*. Das grundsätzliche Verfahren ist einfach, denn man notiert nur im Inneren einer Funktion den Aufruf für die Funktion selbst.

Schematisches Beispiel:

```
function test(){
  ... irgendwelche Anweisungen
  test();
}
```

Listing 67: Rekursiver Aufruf

Sie sollten bei rekursiven Aufrufen jedoch beachten, dass 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 ([Abbruchbedingung]) test();
}
```

Listing 68: 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. $5!$ ¹¹ würde als $1 * 2 * 3 * 4 * 5$ berechnet.

Beispiel (*rekursion1.html*):

```
01 <html>
02 <table>
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 }
```

Listing 69: Rekursive Berechnung der Fakultät

11. Das Ausrufezeichen markiert in der Mathematik die Fakultät. Achtung – in JavaScript hat das Ausrufezeichen eine andere Bedeutung.

```

10 z = new Array();
11 for(i=1;i<150;i++) {
12   z[i] = i;
13   document.write("<tr><td>" + z[i] + "!"</td><td> " + fakultaet(z[i]) + "</td><tr>");
14 }
15 </script>
16 </table>
17 </html>

```

Listing 69: Rekursive Berechnung der Fakultät (Forts.)

Von Zeile 4 bis 9 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 13 nur mit einem Argument, in der Folge aber mit zwei Argumenten (in Zeile 8) aufgerufen wird. Aus diesem Grund wird in Zeile 5 ein Defaultwert für den ersten Aufruf gesetzt (*siehe dazu auch das Rezept » Wie kann ich eine Funktion mit Defaultwerten für Parameter erstellen?« auf Seite 148*). Der Aufruf in Zeile 8 reduziert den ersten Übergabewert immer um den Wert 1, so dass das Abbruchkriterium in Zeile 6 nach genau so vielen Schritten erreicht ist, wie der ursprüngliche Wert des ersten Argumentes angibt. Für unser Beispiel wird ein Array angelegt, dieses mit Zahlen zwischen 1 und 150 gefüllt und dann die jeweilige Fakultät ausgegeben.

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 8). 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 <table>
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 </table>
14 </html>

```

Listing 70: Rekursion mit Rückgabewert, aber ohne rekursiven Aufruf mit `return`

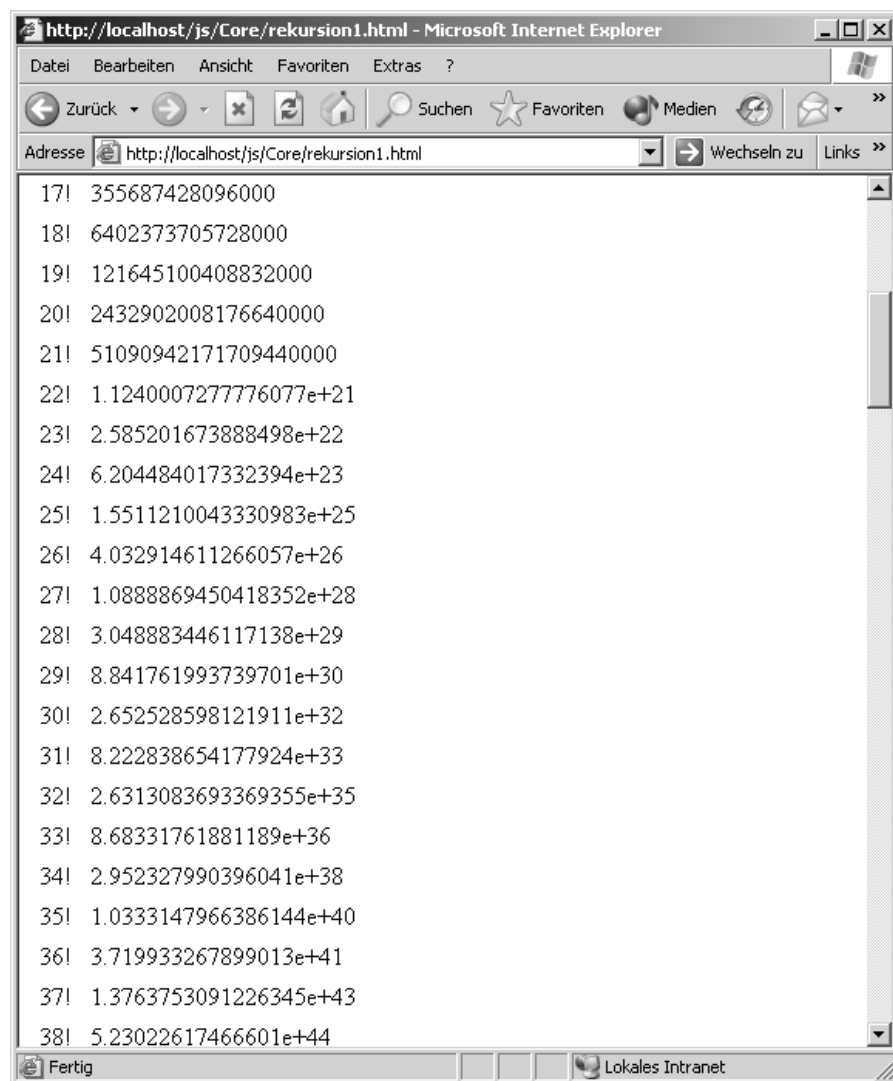


Abbildung 34: Berechnung der Fakultät

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.

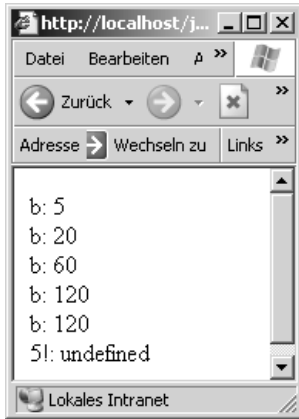


Abbildung 35: Undefinierter Rückgabewert

31 Wie erzeuge ich eine Objektinstanz?

Ein Großteil des Nutzens von JavaScript resultiert aus dem Umgang mit Objekten. Um aus einer Objektdекlaration (Klasse) explizit eine neue Objektinstanz anzulegen, verwendet man in der Regel das reservierte JavaScript-Schlüsselwort `new`, gefolgt von dem Bezeichner der Objektdекlaration und anschließend einem Klammernpaar (eventuell mit Parametern darin). Dies sieht von der Syntax her so aus:

```
new [Objektdекlaration]([Optionale Parameter]);
```

Listing 71: Erzeugen einer Objektinstanz

Der Bezeichner der Objektdекlaration samt Klammernpaar und Inhalt ist das, was als *Konstruktormethode* oder kurz *Konstruktor* bezeichnet wird, welche vom Bezeichner her immer identisch mit der Objektdекlaration 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 hatte, 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:

```
var [ObjektInstanz] = new [Objektdекlaration]([Opt Parameter]);
```

Listing 72: Erzeugen einer Objektinstanz und Zuweisung zu einer Variablen

Das ganze Zuweisungsverfahren unterscheidet sich nicht von dem Verfahren, wie wir es bei primitiven Datentypen verwenden.

Hinweis

Es kann zu einer Objektdекlaration durchaus mehr wie eine Konstruktormethode geben. Die `Date`-Klasse besitzt beispielsweise verschiedene Konstruktormethoden mit unterschiedlichen Aufgaben (siehe in dem Beispiel zum Rezept »Wie kann ich bei einer Funktion einen Wert zurückgeben?« auf Seite 149).

32 Wie erzeuge ich in JavaScript ein eigenes Objekt?

Neben der Verwendung von den vordefinierten Standardobjekten über JavaScript kann man auch eigene Objekte definieren. Da JavaScript jedoch nicht streng objektorientiert, sondern nur objektbasierend ist, muss man das auf der einen Seite aber nicht oft tun, und auf der anderen Seite geht das auch grundsätzlich nur sehr eingeschränkt. Damit Sie in JavaScript nun ein eigenes Objekt anlegen können, sind effektiv zwei Schritte nötig.

1. Eine eigene Objektdекlaration (Klasse) muss erstellt werden.
2. Mit Hilfe dieser Klasse wird dann wie bei vordefinierten Objektdекларationen eine konkrete Objektinstanz erstellt.

Um ein eigenes Objekt anzulegen, müssen Sie innerhalb JavaScript einfach eine spezielle Funktion definieren, die als Deklaration des Objekts (als seine Konstruktormethode) fungiert. Nachdem die Objektdекlaration angelegt ist, können Sie an anderen Stellen innerhalb Ihres JavaScripts Instanzen dieses Objektes erzeugen. Dies machen Sie wie bei Standardklassen, indem Sie einer Variablen mit dem reservierten JavaScript-Schlüsselwort `new` eine Objektinstanz zuweisen. Dies sieht also von der Syntax her schematisch so aus:

```
function [Funktionsname]() {  
  ... [Beschreibungen von Eigenschaften] ...  
}  
  
...  
var [ObjektInstanz] = new [Funktionsname]();
```

Listing 73: Deklarieren einer Klasse und Erzeugen einer Objektinstanz

In der Konstruktormethode ist ein JavaScript-Schlüsselwort von zentraler Bedeutung – das `this`-Schlüsselwort. Es ist ein wesentliches Element jeder objektorientierten Programmiersprache, denn darüber hat man Zugriff auf das Objekt selbst, welches eine Funktionalität aufruft. Wenn Sie in der Form `this.[variable]` in der Konstruktormethode eine Variable deklarieren, ist das eine Eigenschaft von dem daraus erzeugten Objekt.

Beispiel:

```
function MeinObjekt(){
    this.a = 42;
}
```

Listing 74: 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 in der Konstruktormethode über `this`. Dabei müssen Sie aber die Funktion dort als Funktionsreferenz ohne Klammern notieren!

Beispiel:

```
function test(){
    document.write("Heute ist folgendes Datum: " + new Date());
}
function MeinObjekt(){
    this.test = test;
}
```

Listing 75: Eine Deklaration einer Methode

Ein Objekt erzeugen Sie dann einfach so:

```
new MeinObjekt();
```

Listing 76: Erzeugen einer Objektinstanz

Das nachfolgende Beispiel soll in einem etwas umfangreicheren Fall die Erzeugung und Verwendung selbst definierter Objekte zeigen. Wir erzeugen eine Objektdекlaration `Tier`, die dann reale Tiere so weit beschreiben soll, wie es notwendig ist.

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.

Wir wollen uns auf drei Eigenschaften und eine Methode für die `Tier`-Objekte beschränken. Dabei charakterisieren diese gewissen Eigenschaften das konkrete Tier, aber in der ObjektdeklARATION kann ja naturgemäß noch nicht bekannt sein, wie das später daraus erzeugte Objekt heißen soll beziehungsweise was es konkret für ein Tier ist. Wie soll man auch in der Bauvorschrift schon wissen, welchen Variablenbezeichner der Programmierer später nimmt, der das konkrete Objekt erzeugt? Aus diesem Grund wird in der Deklaration der Name für dieses spätere Objekt durch einen Platzhalter ersetzt – *this*.

Beispiel (*objektErzeugung.html*):

```
01 <html>
02 <table>
03 <script language="JavaScript">
04 function Laut geben(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.Laut geben = Laut geben;
13 }
14 //Objekterzeugung
15 var felix = new Tier("M",4,"Hund");
16 var florian = new Tier("M",4,"Katze");
17 var andrea = new Tier("F",31,"Maus");
18 document.write("Das erste Tier ist ein " + felix.art + " und " + felix.alter + "
Jahre alt. ");
19 if(felix.geschlecht == "M") document.write("Es handelt sich um ein
Männchen.<br>");
20 else document.write("Es ist ein Weibchen.<br>");
21 felix.Laut geben("Wau Wau");
22 document.write("Das zweite Tier ist ein " + florian.art + " und " + florian.alter
+ " Jahre alt. ");
23 if(florian.geschlecht == "M") document.write("Es handelt sich um ein
Männchen.<br>");
24 else document.write("Es ist ein Weibchen.<br>");
25 florian.Laut geben("Mihiau");
26 document.write("Das dritte Tier ist eine " + andrea.art + " und " + andrea.alter +
" Jahre alt. ");
27 if(andrea.geschlecht == "M") document.write("Es handelt sich um ein
Männchen.<br>");
28 else document.write("Es ist ein Weibchen.<br>");
29 andrea.Laut geben("Piiiiieeps");
30 </script>
31 </table>
32 </html>
```

Listing 77: Selbst definierte Objekte

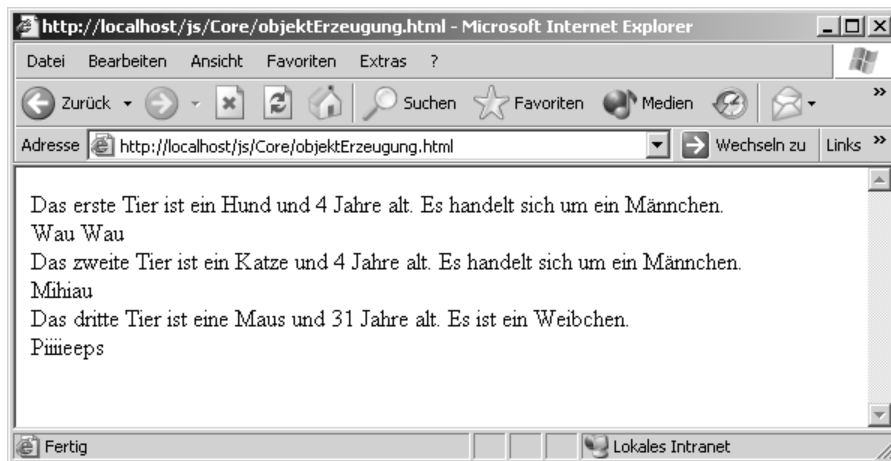


Abbildung 36: Die drei aus Tier erzeugten Objekte

In den Zeilen 4 bis 6 finden Sie eine gewöhnliche Funktionsdeklaration vor. In den Zeile 8 bis 13 wird der Konstruktor definiert. Beachten Sie Zeile 12, in der die Funktion `Laut geben()` 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 `Laut geben()` aufgerufen. Über den `this`-Stellvertreter der Objektdeklaration wurden bei der Erzeugung jedem konkreten Objekt die jeweiligen Eigenschaften zugewiesen.

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 *kaufen*¹². 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 von Funktionalität zu erzeugen. In der OOP werden nun ähnliche Objekte zu Gruppierungen zusammengefasst, die eine leichtere Klassifizierung der Objekte ermöglicht (Klassen). Die Eigenschaften und Methoden der Objekte werden in den Gruppierungen gesammelt und für eine spätere Erzeugung von realen Objekten verwendet. Zentrale Bedeutung hat dabei die hierarchische Struktur der Gruppierungen, von allgemein bis fein. Beispiel: Eine fiktive Klasse `Hund` könnte man als Verfeinerung (oder im OOP-Sprachgebrauch *Spezialisierung*) der Klasse `Saeugetier` betrachten und diese Klasse wiederum als Verfeinerung einer höheren Klasse `Tier`.

12. Der Begriff »Kaufen« ist eine offizielle Bezeichnung aus der OOP.

Gemeinsame Erscheinungsbilder sollten also in der objektorientierten Philosophie in einer möglichst hohen 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*, eine untergeordnete Klasse *Subklasse* genannt.

Super- und Subklassen stehen in der richtigen OOP nun 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 (was ja bei Vererbung genauso geschieht) werden. 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 78: Erweiterung einer Eigenschaften per Prototyping

beziehungsweise

```
[Objekttyp].prototype.[Methode] = [Methode];
```

Listing 79: Erweiterung einer Methode per Prototyping

Sobald diese Aktion durchgeführt wurde, besitzen alle zukünftig erzeugten, aber auch alle vorher bereits erzeugten Objekte die neue Eigenschaft beziehungsweise Methode. Insbesondere ist zu beachten, dass nach der `prototype`-Anweisung erzeugte Objekte für die zugefügte 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 bereits daraus ein Objekt erzeugt wurde.

Beispiel (*proto1.html*):

```

01 <html>
02 <table>
03 <script language="JavaScript">
04 function Laut geben(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.Laut geben = Laut geben;
13 }
14 //Objekterzeugung
15 var felix = new Tier("M",4,"Hund");
16 document.write("Das erste Tier ist ein " + felix.art + " und " + felix.alter + "
Jahre alt. ");
17 if(felix.geschlecht == "M") document.write("Es handelt sich um ein Männchen.<br>");
18 else document.write("Es ist ein Weibchen.<br>");
19 felix.Laut geben("Wau Wau");
20 Tier.prototype.besitzer = "Ralph";
21 document.write("Besitzer von Tier 1 ist " + felix.besitzer + "<br>");
22 var florian = new Tier("M",4,"Katze");
23 document.write("Das zweite Tier ist ein " + florian.art + " und " + florian.alter +
" Jahre alt. ");
24 if(florian.geschlecht == "M") document.write("Es handelt sich um ein Männchen.<br>");
25 else document.write("Es ist ein Weibchen.<br>");
26 florian.Laut geben("Mihiau");
27 document.write("Besitzer von Tier 2 ist " + florian.besitzer + "<br>");
28 florian.besitzer="Andrea";
29 document.write("Besitzer ist " + florian.besitzer + "<br>");
30 var pumuckel = new Tier("M",444,"Kobold","Meister Eder");
31 document.write("Das dritte Tier ist ein " + pumuckel.art + " und " +
pumuckel.alter + " Jahre alt. ");
32 if(pumuckel.geschlecht == "M") document.write("Es handelt sich um ein
Männchen.<br>");
33 else document.write("Es ist ein Weibchen.<br>");
34 pumuckel.Laut geben("Hurra, hurra");
35 document.write("Besitzer von Tier 3 ist " + pumuckel.besitzer + "<br>");
36 pumuckel.besitzer = "Meister Eder";
37 document.write("Besitzer von Tier 3 ist " + pumuckel.besitzer + "<br>");
38 document.write("Besitzer von Tier 1 ist " + felix.besitzer + "<br>");
39 </script>

```

Listing 80: Veränderung einer eigenen Klasse mit prototype

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

```
40 </table>
41 </html>
```

Listing 80: Veränderung einer eigenen Klasse mit prototype (Forts.)

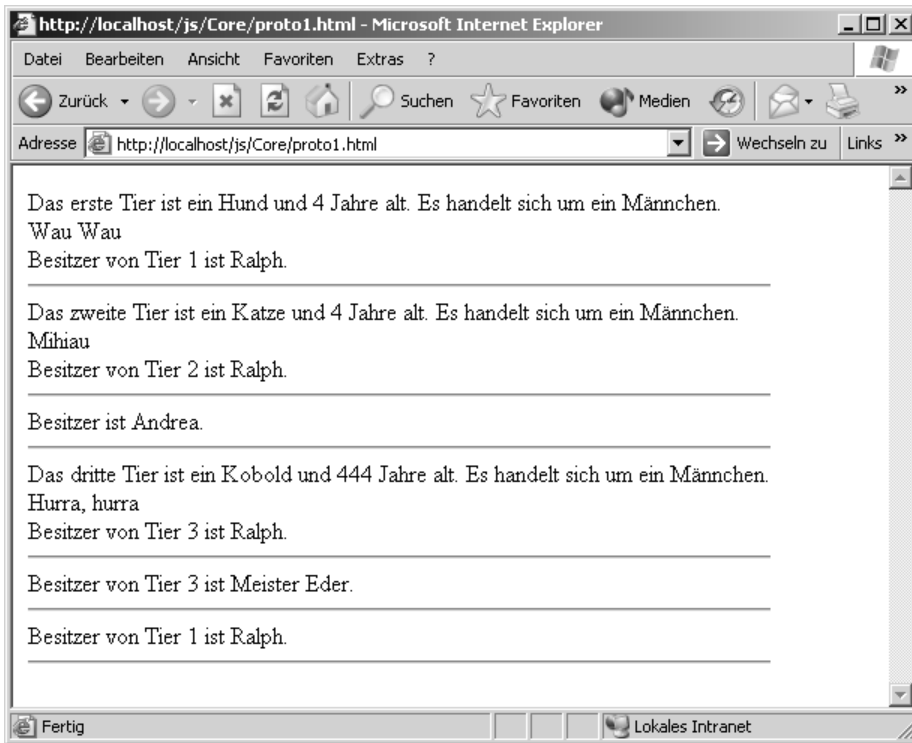


Abbildung 37: Prototyping bei einer eigenen Klasse

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 `Laut geben()`.

In Zeile 15 wird aus der Konstruktormethode ein Objekt `felix` erzeugt, das dann diese drei Eigenschaften und die Methode besitzt. Die Zeilen 16 bis 19 bestätigen das mit entsprechenden Ausgaben.

Zeile 20 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 21 wird auf die neue Eigenschaft via dem 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 22 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.

Interessant ist jetzt Zeile 30, 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 in Zeile 35 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 36), wird der Wert der Eigenschaft `besitzer` für das Objekt geändert. Der Wert der Eigenschaft `besitzer` für das erste Objekt wurde damit aber nicht geändert (was so auch zu erwarten und sinnvoll ist). Die Ausgabe von Zeile 37 demonstriert das.

Wir wollen in einem zweiten Beispiel nun eine Standardklasse von JavaScript erweitern. Dabei wollen wir die Klasse `Function` um eine Methode zur Berechnung der Fakultät erweitern. Die `Function`-Klasse bietet sich nahezu ideal für Prototyping an, denn diese 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 geht etwa so (*proto2.html*):

```
01 <html>
02 <table>
03 <script language="JavaScript">
04 function fakultaet(a) {
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 document.write(5 + "!: " + a.fak(4) + "<br>");
13 </script>
14 </table>
15 </html>
```

Listing 81: Erweiterung einer Standardklasse mit Prototyping

In Zeile 10 wird die Standardeigenschaft `prototype` der Klasse `Function` verwendet und darüber die Methode `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 Zeile 12 über dieses Objekt auf die erweiterte Methode `fak` zugegriffen.

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

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 oft Informationen 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 <script language="JavaScript">
03 document.write(String(new Date()));
04 </script>
05 </html>
```

Listing 82: 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 tückischen Fehlern führen.

Hinweis

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 in eine andere Funktion oder Methode notieren, die dann den String automatisch extrahiert. Etwa so:

```
01 <html>
02 <script language="JavaScript">
03 document.write(new Date());
04 </script>
05 </html>
```

Listing 83: 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:

```
01 <html>
02 <script language="JavaScript">
03 document.write(new Date().toString());
04 </script>
05 </html>
```

Listing 84: Ebenfalls die Extraktion des Inhaltes von einem Datumsobjekt unter Verzicht auf die Funktion `String()`, aber mit explizitem Einsatz von `toString()`

35 Wie kann ich Zeichenketten kodieren und dekodieren?

Es gibt im Rahmen von verschiedenen Webvorgängen Situationen, wo die direkte Verwendung von gewissen Sonderzeichen (dazu zählen auch das Leerzeichen und deutsche Umlaute) nicht gestattet ist, etwa allgemein in URLs¹³ oder bei CGI-Prozessen. Wollen Sie diese dennoch verwenden, müssen sie diese *maskieren* beziehungsweise *kodieren*. Jedes potenziell kritische Zeichen wird 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.

Statt so eine Kodierung manuell durchzuführen, 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. Spezielle Unicode-spezifizierte Escape-Sequenzen der Form `%uXXXX` werden nicht unterstützt.

Beispiel (*kodier.html*):

```
01 <html>
02 <script language="JavaScript">
03 a = prompt("Geben Sie was ein");
```

Listing 85: Die Kodierung einer Benutzereingabe

13. Wobei hier mittlerweile auch deutsche Umlaute möglich sind (wenngleich es zahlreiche Browser noch nicht unterstützen und es den meisten Anwender auch noch nicht vertraut ist), etwa eine Webadresse wie www.münchen.de (mit ü).

```

04 document.write("Ihre Eingabe: "+a+"<br>");
05 document.write("Die Kodierung Ihrer Eingabe: "+escape(a)+"<br>");
06 </script>
07 </html>

```

Listing 85: Die Kodierung einer Benutzereingabe (Forts.)

In Zeile 5 wird die Eingabe eines Anwenders (die erfolgt mit Zeile 3) mit `escape()` kodiert.

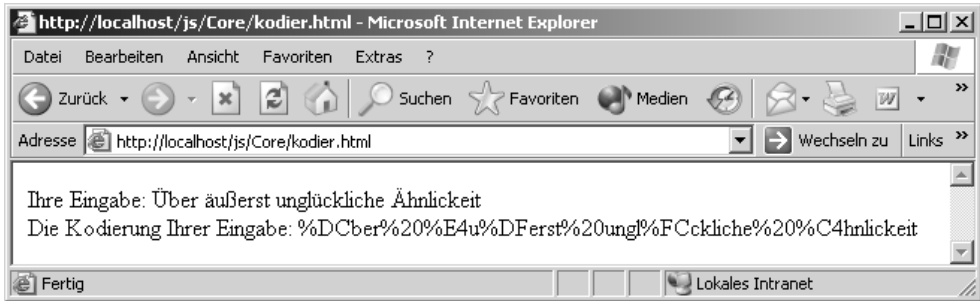


Abbildung 38: Alle potenziell kritischen Zeichen sind kodiert.

Das Gegenstück zu `escape()` ist die Funktion `unescape([Zeichenkette])`. 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. Meist wendet man diese Funktion an, um per `escape()` kodierte Strings wieder zurück zu verwandeln (etwa nachdem Sie mit HTTP von einem Client zu einem Server geschickt wurden).

Beispiel (*kodier.html*)¹⁴:

```

01 <html>
02 <script language="JavaScript">
03 a = escape(prompt("Geben Sie was ein"));
04 document.write("Das kam an: " + a + "<br>");
05 document.write("Die Dekodierung der empfangenen Werte: " + unescape(a) + "<br>");
06 </script>
07 </html>

```

Listing 86: Kodieren und wieder dekodieren

In Zeile 3 wird die Eingabe eines Anwenders entgegengenommen und unmittelbar mit `escape()` kodiert. In Zeile 4 wird das Resultat der Kodierung ausgegeben und in Zeile 5 die dekodierten Werte.

14. Hier wird nur ganz einfach die Übergabe der Werte in einem einzigen Skript simuliert.



Abbildung 39: Die kodierten Werte im String werden wieder dekodiert.

36 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 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`) das *Tabulatorzeichen*, `\v` (beziehungsweise `\u000B`) der *vertikale Tabulator*, `\\` für *Backslash* (beziehungsweise `\u005C`), `\'` (beziehungsweise `\u0027`) ein *einfaches Anführungszeichen* und `\"` (beziehungsweise `\u0022`) ein *doppeltes Anführungszeichen*.

Beispiel (*sonderzeichen.html*):

```
01 <html>
02 <script language="JavaScript">
03 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
(bbeziehungsweise \u000B) der vertikale Tabulator, \\ für Backslash (beziehungsweise
\u005C) \' (beziehungsweise \u0027) ein einfaches Anführungszeichen und \"
(bbeziehungsweise \u0022) ein doppeltes Anführungszeichen.");
04 </script>
05 </html>
```

Listing 87: Der Einsatz von Sonderzeichen in einem String

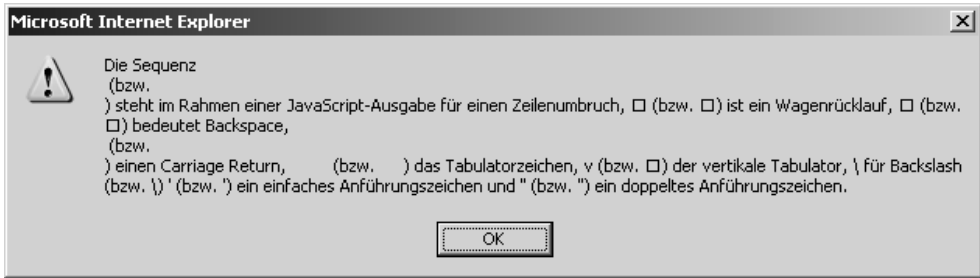


Abbildung 40: Viele Sonderzeichen in einem String, der mit `alert()` ausgegeben wird – hier teils nur begrenzt sinnvoll eingesetzt, aber die Wirkung ist gut zu sehen.

Allgemeines zu Formularen und Benutzereingaben

Sicherheit ist grundsätzlich eines der wichtigsten Themen in der Computerwelt. Vor allem bei Internet-Applikationen 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 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 da, 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 der Form `<a href="mailto:[E-Mail-Adresse]">[anzuweisender Text]</a>` 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 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 netzwerkbasierenden Anwendungen für »normale« Netze sind Webangebote jedoch vielfach extrem unsicher. Das betrifft auf der einen Seite die Administration des Web-servers (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 gehen hin zu Benutzereingaben, die ein Webangebot missbrauchen oder abstürzen lassen¹. Es gab 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 trivial. Im Gegenteil – die Planung und Realisierung eines solchen Plausibilisierungssystems kann den Aufwand der

1. Etwa den Eintrag von Skriptcode in ein Gästebuch.

Erstellung von 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 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.

37 Wie kann ich ein Webformular mit HTML aufbauen?

Dies ist hier kein HTML-Buch, aber um mit JavaScript auf Webformulare zugreifen zu können, sollten die wichtigsten² HTML-Grundlagen zum Erstellen eines Webformulars zur Verfügung stehen. Diese HTML-Techniken zum Erstellen von Formularen sind oft auch Personen nicht bekannt, die HTML sonst einigermaßen kennen. Der Grund dürfte nicht die Komplexität der HTML-Anweisungen sein (die ist nicht sonderlich hoch), sondern dass die sinnvolle Nutzung von Formularen so gut wie nie rein auf HTML beschränkt bleibt. Einsteiger, die sich weder mit clientseitiger Programmierung als auch serverseitigen Techniken auseinander setzen wollen beziehungsweise können, werden erst einmal keinen unmittelbaren Nutzen bei einer eigenen Webseite haben.

Hinweis

Das große Problem an Webformularen ist, dass mit reinem HTML keinerlei Auswertung oder sonstige Aktivität möglich ist. Ein rein auf die HTML-Welt beschränktes Formular ist schlicht und einfach nutzlos. An dieser Stelle ist man auf Ergänzungstechniken angewiesen, die – je nach Einsatzziel von dem Formular – auf dem Server oder Client ausgeführt werden. Und auf beiden Seiten kann man JavaScript einsetzen. Auf Seiten des Servers ist JavaScript zwar eher exotisch (aber dennoch konkurrenzfähig¹), ist JavaScript zur Plausibilisierung beziehungsweise Manipulation auf Seiten des Clients nahezu konkurrenzlos. Inwieweit man aber Aufgaben bereits zum Client verlagern kann, hängt massiv von der Zielsetzung des Formulars ab. Während Datenbankaktionen beziehungsweise das Speichern oder Entgegennehmen von Daten eher die Serverseite beanspruchen, können Aktionen wie das Abfangen von fehlerhaften Benutzereingaben bereits oft auf Clientseite realisiert werden.

1. Ich hatte selbst vor kurzem eine JavaScript-Schulung bei einer großen Versicherung gehalten, die JavaScript auf ihrem Webserver einsetzt.

Die Rezepte zur Erstellung der wichtigsten Elemente von einem Webformular in diesem Buchpart (die aufgrund des reinen HTML-Bezugs in einem Abschnitt aufgebaut sind) zeigen Ihnen die wichtigsten HTML-Grundlagen eines Webformulars, wobei bei den Fragestellungen, die sich nicht mit 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.

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 im Rahmen der Webseite (inklusive der Reaktionsmöglichkeiten auf Anwenderaktionen – etwa Selektion oder Deselektion) kann grundsätzlich in reinem HTML erstellt werden. Es gibt Anweisungen zur Generierung von Eingabefeldern, mehrzeiligen Textfeldern, Listen und Aufzählungen, Schaltflächen und beschreibendem Text. Darüber hinaus kann man mit HTML bereits einige Funktionalität wie die Festlegung der Anzahl der maximal erlaubten Zeichen etc. festlegen. Ansonsten gestaltet man ein Webformular optisch mit den üblichen HTML-Befehlen und reinen Textpassagen.

Tipp

Ein Webformular wird ohne zusätzliche Gestaltungselemente meist schlecht aussehen. Wenn Beschriftungen vor Eingabefeldern notiert werden, entsteht in der Regel ein indiskutabler Flatterrand.

Abbildung 41: So darf kein Formular aussehen.

Viele Webformulare nutzen deshalb zur optischen Gestaltung Tabellen. Sie können ein Formular sowohl in einen Tabellencontainer einschließen als auch umgekehrt eine oder mehrere Tabellen in einen Formularcontainer einschließen. Die einzelnen Formularelemente packt man meist in jeweils einzelne Zellen.

Abbildung 42: So oder so ähnlich kann man einen Flatterrand verhindern.

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 aber mehr Berechnungsaufwand. Dennoch kann es sinnvoll sein, wenn man die Positionierung in Verbindung mit JavaScript durchführt und die Positionen aufgrund dynamischer Vorgaben wie der Bildschirmauflösung berechnet.

38 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.

Beispiel:

```
<form>
...// Formularelemente und formatierende weitere HTML-Anweisungen
</form>
```

Listing 88: Ein Formularcontainer

Bereits in dieser einfachen Form genügt der Container allen Anforderungen an einen Formularcontainer. Nur kann man mit so einem Formular in der Regel nicht das machen, was ein Formular auszeichnet – es lassen sich keine Daten zu einem Server schicken (zumindest nicht auf einfache Weise)³. Deshalb ergänzen den äußeren Container in der Regel zwei bis drei, durch ein Leerzeichen getrennte, Parameterangaben.

Der action-Parameter

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 der URL zu einem auswertenden Programm beziehungsweise Skript. Zu dieser Spezifikation des URL dient die Angabe `action=[URL]`. Der 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

Achtung

Bei den meisten Werten, die einem HTML-Parameter zugewiesen werden, kann der Einschluss des Wertes in Hochkommata unterbleiben. Die Wertzuweisung wird immer automatisch als String betrachtet, etwa so `<body bgcolor=red>`. Die Zuweisung eines URL ist eine Ausnahme (aufgrund der enthaltenen Sonderzeichen wie dem `/`). Der gesamte URL sollte auf jeden Fall bei einer Zuweisung in Anführungszeichen eingeschlossen werden.

3. 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 der `<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 aber auch Daten direkt per JavaScript verschicken.

Hinweis

Sie können sowohl absolute als auch relative URLs verwenden. Wenn Sie einen absoluten URL einsetzen, notieren Sie als erste Angabe wie gewohnt das Protokoll (nahezu immer http). Bei einem relativen URL wird das Zielskript beziehungsweise -programm relativ von der Position der HTML-Datei mit dem Webformular auf dem Server ausgesucht.

Tipp

Nicht jeder Anwender verfügt 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. 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. Der `<form>`-Tag könnte also folgendermaßen aussehen:

```
<form action="mailto:ralph.steyer@rjs.de" method="post">
```

Listing 89: 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>
07   <td>Name: </td>
08   <td><input name="name"> </td>
09 </tr>
10 <tr>
11   <td>Vorname </td>
12   <td><input name="vname"> </td>
13 </tr>
14 <tr>
15   <td colspan="2"><input type="Submit"> </td>
16 </tr>
17 </form>
18 </table>
19 </body>
20 </html>
```

Listing 90: 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. 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 43: 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.

So bequem 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 eine E-Mail beim Empfänger ankommen, die in der Grundform für den Empfänger leider nicht gut lesbar ist. Das ist sehr logisch, denn Formulardaten sollen von Natur aus maschinell¹ weiterverarbeitet werden und sind 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"` angeben². Aber selbst dann ist die Darstellung der Formulardaten nicht unbedingt komfortabel zu lesen.

Das entscheidende Problem bei dieser Lösung stellt jedoch die Notwendigkeit dar, aus der Webseite heraus ein, dem WWW-Browser zugeordnetes, Standard-E-Mail-Programm zusätzlich zu starten, 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. Da ist einmal die große Gruppe der Laien, bei denen der Webbrowser unwissentlich so eingestellt ist. Aber ebenso müssen Sie die immer größer werdende Gruppe der Wissenden beachten, die aus Sicherheitsgründen den Webbrowser entsprechend konfiguriert haben. Ein zugeordnetes Standard-E-Mail-Programm in einem

Webbrowser ist aus verschiedenen Gründen ein extremer Lapsus. So finden bei bestimmten Browser-E-Mail-Konstellationen³ Makroviren und andere Schädlinge eine extrem leichte Beute durch Wechsel der Welten. Ebenso soll der Internet Explorer in einigen Versionen persönliche Angaben wie die Standard-E-Mail-Adresse⁴ 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.

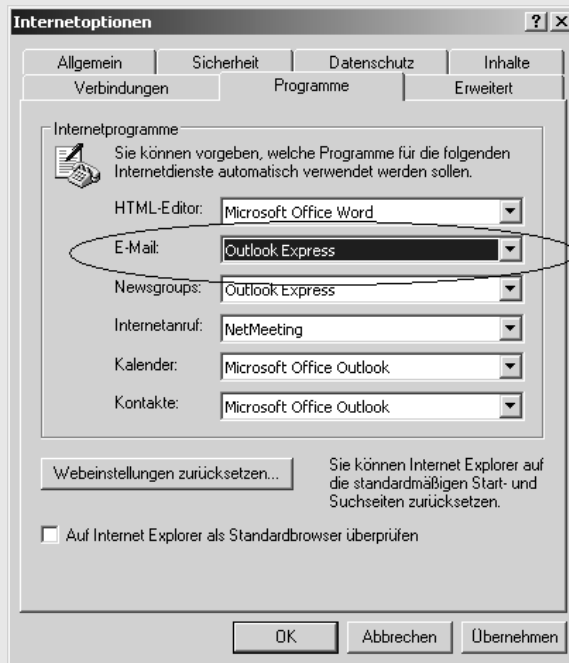


Abbildung 44: Die automatische Verbindung von einem Browser mit einem E-Mail-Programm sollte von vorsichtigen Anwendern gelöst werden – hier der Internet Explorer im Dialog Extra – Internetoptionen.

1. Also in der Regel mit serverseitigen Skripten auf einem Webserver.
2. 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.
3. Hauptsächlich in einigen Versionen des Internet Explorers in Kombination mit bestimmten Versionen von Outlook Express beziehungsweise Outlook.
4. In der oft der reale Name enthalten ist!

Sie können also beim Einsatz von `mailto` zur Übertragung von Formulardaten auf keinen Fall sicher sein, dass diese Technik auch funktioniert.

Um die gesamte Problematik zu umgehen, können Sie einen so genannten öffentlichen *Formularmailer* 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 eine zugeordnete E-Mail aufrufen müssen, sondern rein auf Basis von `http` und damit dem Webbrowser des Besuchers die Formulardaten verschicken. Erst im Internet werden die Daten auf die E-Mail-Schiene verladen und Ihnen dann in Ihr E-Mail-Postfach zugestellt. Zwar ist die Zahl der kostenlosen Formularmailer 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 ein reines Kopieren von wenigen HTML-Anweisungen notwendig.

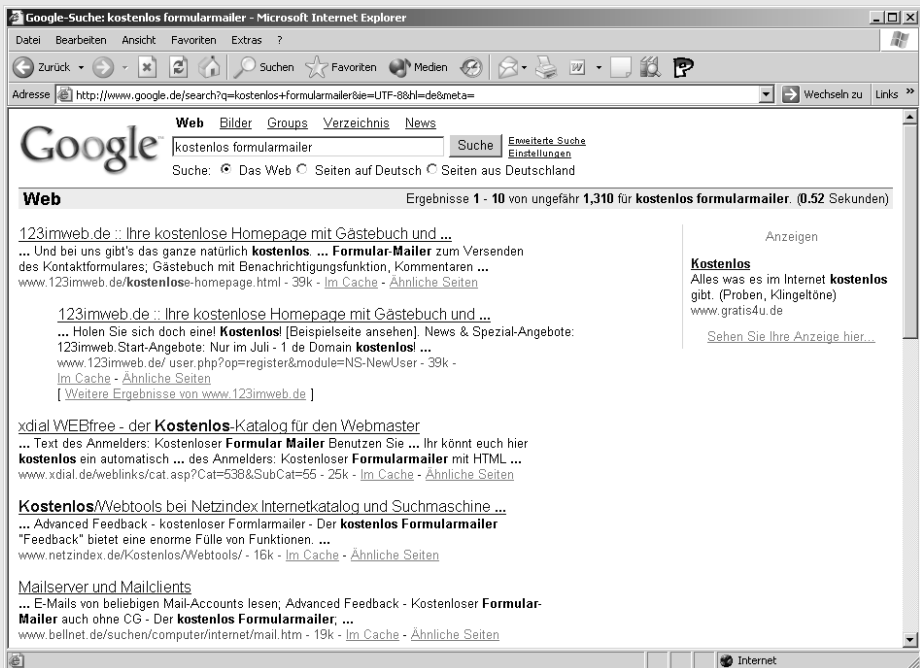


Abbildung 45: Eine Suche in Google führt Sie zu genügend Anbietern von kostenlosen Formularmailern.

Der method-Parameter

Die zweite Standardangabe bei einem Formular ist eine 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. Die Angabe `method` macht nur in Kombination mit einer `action`-Angabe Sinn. Dabei wird gewöhnlicherweise einer von den nachfolgenden zwei Werten verwendet:

- GET
- POST

Im Hintergrund läuft Folgendes ab: Wenn die Benutzereingaben in einem 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 genommen. Die so genannten *HTTP Request Header* werden dabei in der Regel als *Environment-Variablen* (Umgebungsvariablen) übertragen. Bevor die Daten an den Server gesendet werden, werden sie zunächst vom Browser zu einer einzigen Zeichenkette verpackt. Die Art und Weise der weiteren Behandlung wird durch den Parameter `method` gesteuert.

Bei der GET-Methode werden die Daten des Formulars in einer Standard-Umgebungsvariablen gespeichert und sind 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 dem im `<form>`-Tag über den `action`-Parameter angegebenen URL des Zielskripts beziehungsweise -programms und der aus den Eingabedaten erzeugten Zeichenkette ein Pseudo-URL erzeugt, wobei die Formulardaten durch ein `?` von dem ursprünglichen URL abgetrennt werden. Diesen Pseudo-URL sieht der Anwender nach dem Abschicken des Formulars im Adressfeld des Browsers. Das empfangende Programm beziehungsweise Skript wertet den Inhalt dieser Umgebungsvariablen aus, wobei dabei einige Arbeit zu leisten ist. Der Pseudo-URL bei der Übergabe von Werten an ein Skript beziehungsweise Programm beinhaltet die Zeichen der Benutzereingabe nicht unverändert. Bei der Generierung des Pseudo-URL 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 des 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 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 des URL übermittelt, den ein Webscript wieder auflösen muss. Ein solcher String ist ungefähr so aufgebaut:

```
name1=wert1&name2=wert2&name3=wert3
```

Listing 91: 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 in den Pseudo-URL bei bestimmten Sonderzeichen – darunter fallen auch deutsche Umlaute) in einzelne Buchstaben mit dem ASCII-Wert »xx« (Beispiel: %3D wird wieder zu =).

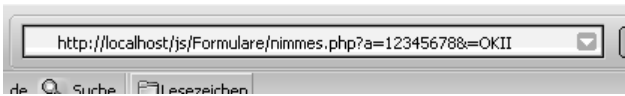


Abbildung 46: Die übertragenen Daten sind in der Adresszeile des Browser sichtbar.

Die Methode POST veranlasst das empfangende Programm beziehungsweise Skript hingegen, das Formular wie eine auf der Kommandozeilenebene erfolgte Benutzereingabe zu behandeln. Es gibt daher kein *EndOfFile*-Signal (*EOF*), und die Länge der übermittelten Daten muss vom empfangenden Programm beziehungsweise Skript aus einer weiteren Standardumgebungsvariable entnommen werden. Für den Anwender hat POST die Konsequenz, dass in der Adresszeile des Browsers keine der verschickten Eingaben des Anwenders zu sehen sind.

Achtung

Es werden bei vielen Browsern in beiden Fällen der Übertragung 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. Diese verschicken auch die Daten, die 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 aufgrund 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.

```
e/formmitgestaltung.html?name=Hugo&name=Weizenkeim&str=Milchstra%Df+42
```

Abbildung 47: Was ist nun der Vorname und was der Name?

Namen von Formularelementen sollten also in einer gesamten Webseite eindeutig sein.

Die GET-Methode hat sich im Web als Regelfall bei Verweisen auf Server etabliert und wird auch standardmäßig verwendet, wenn Sie den `method`-Parameter nicht angeben. Allerdings eignet sich die Methode GET durch die Abhängigkeit von der maximalen Länge eines URLs nicht zur Übertragung größerer Datenmengen! Die Grenze liegt ungefähr bei einem KByte. Auch bei der Übertragung von Formularen per E-Mail wird in der Regel die POST-Methode bevorzugt. Ebenso stellt die Anzeige der Benutzerdaten in der Adresszeile bei sensiblen Daten ein Sicherheitsrisiko dar⁴.

Hinweis

Suchmaschinen verwenden sehr oft die GET-Methode zur Übertragung von Suchbegriffen, denn ein Ergebnis-URL lässt sich als Lesezeichen im Browser speichern. Damit ist die Suche selbst über ein Lesezeichen aufrufbar und muss bei einer späteren Suche nicht mehr manuell eingegeben werden.



Abbildung 48: In der Adresszeile des Browsers sieht man die letzte Suchabfrage und kann sie bookmarken.

4. Sie können in einem Webformular Eingabefelder generieren, in denen die Benutzereingabe verdeckt angezeigt wird (Passwortfelder). Das nutzt jedoch nicht viel, wenn anschließend die sensiblen Daten für jeden sichtbar in der Adresszeile des Browsers zu sehen sind.

Der name-Parameter

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 HTML-Elementen in einer Webseite – bei sämtlichen enthaltenen Formularelementen ebenso.

Der target-Parameter

Über den `target`-Parameter können Sie im `<form>`-Container als URL angeben, wohin vom Zielskript beziehungsweise -programm die Antwort 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. 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 verwenden⁵, erfolgt die Ausgabe in dem geöffneten Fenster. Verwenden Sie einen Fensternamen, den es noch nicht gibt⁶, wird eine neue Fensterinstanz des Browsers geöffnet und die Ausgabe darin angezeigt.

Der enctype-Parameter

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 HTML mit dem Attribut `enctype` festlegen, wobei es – falls das Attribut nicht festgelegt ist – zur Kodierung ein Standardverfahren gibt (bei den meisten Browsern ist das `application/x-www-form-urlencoded`).

Ein schematischer Formularcontainer

Der schematische Aufbau eines äußeren (fast vollständigen – siehe dazu weiterführende HTML-Literatur) Formularcontainers sieht also so aus:

```
<form action="[URL]" method=[GET/POST] name="[Bezeichner]" target="[URL]">
...[Formularelemente]
</form>
```

Listing 92: 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-Script *meinScript.cgi* im Verzeichnis *cgi-bin* auf dem Webserver *www.rjs.de* schickt, sähe so aus:

5. Es ist also eine Browserinstanz mit diesem Namen geöffnet.

6. Es ist also noch keine Browserinstanz mit entsprechendem Namen geöffnet.

```
<form action="http://www.rjs.de/cgi-bin/meinScript.cgi" method="GET" name="a">
...
</form>
```

Listing 93: Ein Container für ein Webformular mit Angabe des Ziel-URL, der Methode und des Namens

39 Wie kann ich ein einzeliges Eingabefeld realisieren?

Ein einzeliges Eingabefeld nimmt beliebige Benutzereingaben entgegen und ist der Standardfall für die Verwendung des `<input>`-Tags. Wenn Sie diesen Tag ohne irgendwelche Parameter notieren, erhalten Sie ein solches einzeliges Eingabefeld. In der Regel notiert man aber das Attribut `type="text"`, um das einzelige 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 Hauptgrund ist aber, dass manche Browser nur den Inhalt von den Formularelementen verschicken, deren `name`-Wert gesetzt ist (siehe Seite 174). Der Bezeichner für den Feldname darf wie üblich keine Leerzeichen und keine deutschen Umlaute enthalten.

Beispiel:

```
<input type="Text" name="a">
```

Listing 94: Ein einzeliges Eingabefeld

Die maximale Anzahl an angezeigten Zeichen festlegen

Wenn Sie den `<input>`-Tag mit dem Parameter `size=[numerischer Wert]` erweitern, legen Sie die Anzahl der maximal anzeigbaren Zeichen in dem Feld fest. Wenn von einem Anwender mehr Zeichen eingegeben werden, wird der Inhalt seitwärts gescrollt. Wenn Sie diese Angabe `size` nicht machen, bekommt das Eingabefeld vom Webbrowser eine voreingestellte Größe (in der Regel 20 Zeichen) zugewiesen.

Achtung

Diese Angabe `size` beschränkt in keiner Weise die Anzahl der Zeichen, die ein Anwender eingeben darf. Die Festlegung betrifft nur die Anzeige.

Wenn Sie den `<input>`-Tag mit dem Parameter `maxlength=[numerischer Wert]` erweitern, legen Sie die Anzahl der maximal einzugebenden Zeichen fest. Sofern der Anwender mehr Zeichen eingeben will, werden diese nicht in das Eingabefeld übernommen. Wenn diese Angabe die maximal anzeigbaren Zeichen überschreitet, wird bei Bedarf automatisch seitwärts gescrollt. Wenn diese Angabe nicht gesetzt wird, kann das Eingabefeld eine beliebige Anzahl von Zeichen entgegennehmen (bis auf Plattformbeschränkungen).

Beispiel:

```
<input type="text" maxlength="10">
```

Listing 95: Ein einzeliges Listenfeld mit Festlegung der maximal erlaubten Zeichen

Hinweis

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 317). Allerdings werden Sie dann wahrscheinlich das Verhalten von dieser Beschränkung anders als das Standardverhalten bei reiner HTML-Festlegung aufbauen. Während in 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. Ein Anwender kann dann also im Grunde 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 (als auch grundsätzlich) kein JavaScript einsetzen, wenn Ihnen HTML bereits eine bestimmte Funktionalität bietet. Die potenziellen Probleme reduzieren sich erheblich, je weniger Techniken Sie beim Anwender voraussetzen.

Die 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 einzeligen Eingabefeld eine minimale Anzahl der einzugebenden Zeichen festlegen wollen und den `<input>`-Tag mit einem entsprechenden Parameter `maxlength=[numerischer Wert]` erweitern können⁷, gibt es in 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 317).

Einen Vorgabewert festlegen?

Wenn Sie den `<input>`-Tag mit dem Parameter `value="[Wert]"` erweitern, legen Sie einen Vorbelegungswert fest.

7. Siehe das Rezept »Wie kann ich rein mit HTML bei einem einzeligen Eingabefeld eine maximale Anzahl der einzugebenden Zeichen festlegen« auf Seite 177.

Beispiel:

```
<input type="text" value="10">
```

Listing 96: Ein einzeliges Listenfeld mit Vorgabewert

Hinweis

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 334). Diese Festlegung mit JavaScript ist sogar weitaus flexibler zu handhaben, als es mit 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.

40 Wie kann ich ein einzeliges Passwortfeld realisieren?

Ein Spezialfall eines einzeligen Eingabefeldes ist ein Passwortfeld. Dabei werden die Feldeingaben des Anwenders nur verdeckt angezeigt (etwa durch Sternchen).

Die Syntax ist vollkommen analog wie beim gewöhnlichen einzeligen 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 97: Ein Passwortfeld mit acht Zeichen



Abbildung 49: Ein Passwortfeld

Hinweis

Im Prinzip können Sie auch mit JavaScript ein verdecktes Eingabefeld aufbauen (bei der Eingabe jedes Zeichens wird dieses unmittelbar übernommen und im Eingabefeld stattdessen 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.

41 Wie kann ich eine einfache Formulschaltfläche realisieren?

Über den `<input>`-Tag mit der `type`-Angabe "button" können Sie eine einfache Schaltfläche generieren. Diese hat eine frei zuordenbare Funktionalität. Eine solche Schaltfläche ist also nicht (!) mit einer Standardfunktionalität von dem Formular wie dem Verschieben 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 98: Das Fragment einer Befehlsschaltfläche in Verbindung mit einem Event-Handler und dem Aufruf einer JavaScript-Funktion

Ende

Abbildung 50: Eine typische Schaltfläche in einem Webformular

Tip

In einigen Browsern funktioniert auch ein Container mit dem HTML-Tag `<button>`. Und zwar ohne (!) umgebenden Formularcontainer, etwa so:

```
<button>OK</button>.
```

Listing 99: Eine Schaltfläche ohne umgebende Formularstruktur



Abbildung 51: Der Internet Explorer als auch die meisten neueren Browser anderer Hersteller kommen mit dem `<button>`-Tag zurecht.

Allerdings gehört dieser Tag nicht zum offiziellen Sprachumfang von HTML und wird folgerichtig von streng konformen Browsern (hauptsächlich allerdings älteren) nicht unterstützt.



Abbildung 52: Das macht der Navigator 4.7, wenn Sie den `<button>`-Tag einsetzen.

42 Wie kann ich nur mit HTML eine Formulschaltfläche zum Abschicken von Formulardaten realisieren?

Über den `<input>`-Tag mit der `type`-Angabe `"submit"` erzeugen Sie eine Schaltfläche zum Abschicken der Daten eines Webformulars. Das funktioniert rein auf Basis von HTML und gänzlich ohne ergänzende Techniken wie JavaScript. Meist ist eine Schaltfläche mit der Beschriftung über `value` ergänzt. Beispiel:

```
<input type="submit" value="OK">
```

Listing 100: Ein Absendebutton

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.

Hinweis 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 wegschicken?« auf Seite 229). Diese Vorgehensweise ist 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 immer sowohl einen Submit- als auch einen Reset-Button zur Verfügung stellen.

Tipp

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.

43 Wie kann ich rein mit HTML eine Grafik zum Abschicken von Formulardaten realisieren?

In der Standardform eines Webformulars werden Formulardaten mit einer gewöhnlichen Schaltfläche versendet⁸. Aber Sie können auch eine Grafik für das Versenden von Formulardaten verwenden. 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 der `<input>`-Tag dabei eine recht ungewöhnliche Kombination der Parameter. Er wird als eine Art Mischform zwischen einem »normalen« `<input>`-Tag und einem `<img>`-Tag⁹ aufgebaut. Insbesondere muss mit dem Parameter `src` der 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 101: Eine Grafik als Absendebutton eines Formulars

Der 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 102: 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 103: Eine absolut referenzierte Grafik als Absendeelement eines Formulars

8. Siehe das Rezept »Wie kann ich nur mit HTML eine Formulschaltfläche zum Abschicken von Formulardaten realisieren?« auf Seite 185.

9. Dieser wird konventionell zum Einbinden einer Grafik in einer Webseite verwendet.

So würde ein vollständiges Beispielformular aussehen (*form9a.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 104: Ein Webformular mit einer Submit-Grafik

Tipp

Beachten Sie, dass beim Verwenden einer Grafik als Submit-Element auch die Koordinaten von der Position des Mausklicks übertragen werden. Diese Informationen können Sie natürlich für zahlreiche weitergehende Aktionen nutzen. Sie erkennen etwa genau, auf welchen Bereich einer Grafik ein Anwender geklickt hat. Für den Anwender können diese Informationen auch unsichtbar versendet werden. Das geschieht dann, wenn Sie die Formulardaten mit der POST-Methode verschicken.

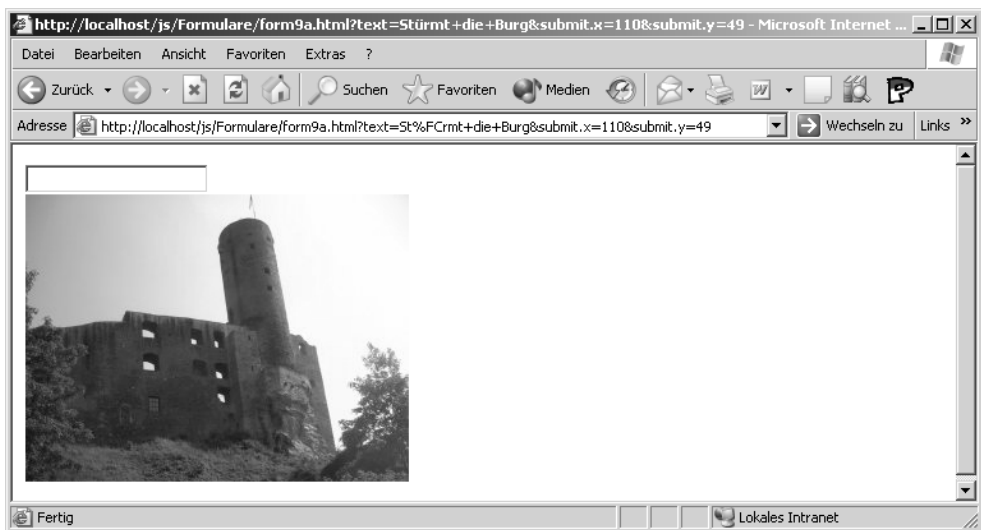


Abbildung 53: Die Formulareingabe wurde mit der Submit-Grafik verschickt – der URL in der Adresszeile des Browsers zeigt die übertragenen Daten.

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 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 105: Ein 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.

Tipp

Wenn Sie beide Größenparameter angeben, können Sie Verzerrungen erreichen, wenn Sie die Relationen nicht beibehalten.

Beispiel:

```
<input type="image" name="submit" src="PICT0025.JPG"
width=100 height=300>
```

Listing 106: Ein Absendebutton

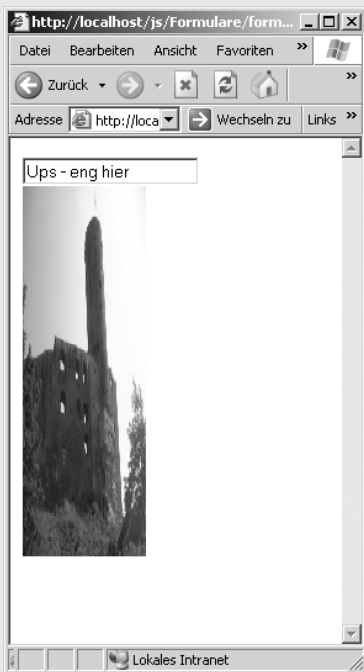


Abbildung 54: Verzerrungen der Grafik

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.

Hinweis

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 wegschicken?« auf Seite 229). Diese Vorgehensweise ist sehr sinnvoll, wenn Sie beim Verschicken von Formulardaten auf jeden Fall das Vorhandensein von JavaScript-Unterstützung erzwingen wollen. Dabei lässt sich ebenfalls eine Grafik in verschiedener Form als sensibles Element verwenden.

44 Wie kann ich nur mit HTML eine Formulare Schaltfläche zum Zurücksetzen von Formulardaten realisieren?

Über den `<input>`-Tag mit der `type`-Angabe `"reset"` erzeugen Sie eine Schaltfläche zum Zurücksetzen der Daten eines Webformulars. Meist ist eine solche Schaltfläche mit der Beschriftung über `value` ergänzt. Beispiel:

```
<input type="reset" value="Abbruch">
```

Listing 107: Ein Button zum Zurücksetzen eines Formulars

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.

Hinweis

Selbstverständlich können Sie auch mit JavaScript die Werte in einem Webformular leeren (siehe dazu das Rezept »Wie kann ich die Formulardaten ohne einen Reset-Button zurücksetzen?« auf Seite 290). 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.

Achtung

Das Zurücksetzen eines Webformulars setzt nur die Felder in dem Formular auf ihre Anfangswerte (meist leer) zurück. Damit wird nicht die Webseite verlassen.

45 Wie kann ich rein mit HTML eine Grafik zum Zurücksetzen von Formulardaten realisieren?

In der Standardform eines Webformulars werden Formulardaten mit einer gewöhnlichen Schaltfläche zurückgesetzt¹⁰. 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 müssen auf JavaScript zurückgreifen (siehe dazu das Rezept »Wie kann ich die Formulardaten ohne einen Reset-Button zurücksetzen?« auf Seite 234).

46 Wie kann ich ein Kontrollfeld realisieren?

Über den `<input>`-Tag mit der type-Angabe "checkbox" können Sie ein Kontrollfeld 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.

Achtung

Beachten Sie, dass das `value`-Attribut nicht die Beschriftung von dem Kontrollkästchen 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. Die Beschriftung eines Kontrollkästchens erfolgt unabhängig von dem Kontrollfeld mit reinem vorangestellten oder nachgestellten Text.

Die Syntax sieht schematisch so aus:

```
<input type=checkbox name="[Bezeichner]" value="[Bezeichner]"> [Beschriftung]
```

Listing 108: Ein Kontrollkästchen

Beispiel (*form1.html*):

```
<html>
<body>
<form action="" method="get">
  <input type="Checkbox" name="a" value="v">Geld<br>
  <input type="Checkbox" name="b" value="v">Glück<br>
  <input type="Checkbox" name="c" value="v">Liebe<br>
  <input type="Submit" value="Hau weg">
</form>
</body>
</html>
```

Listing 109: Ein Formular mit Kontrollkästchen

10. Siehe das Rezept »Wie kann ich nur mit HTML eine Formulschaltfläche zum Zurücksetzen von Formulardaten realisieren?« auf Seite 189.



Abbildung 55: Das Formular mit Kontrollkästchen

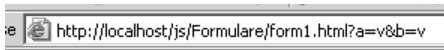


Abbildung 56: Die selektierten Einträge werden samt den Feldnamen in dem Pseudo-URL angezeigt.

47 Wie kann ich eine Gruppe mit Optionsfeldern realisieren?

Über den `<input>`-Tag mit der `type`-Angabe "radio" können Sie ein¹¹ Optionsfeld (engl. *Radiobutton*) generieren. Als Optionsfelder versteht man Formularelemente, 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 kann), müssen sie einer Gruppe zugeordnet werden. Dabei gehen Optionsfelder (und Kontrollkästchen) einen anderen Weg wie die meisten HTML-Strukturen, die einer Oberstruktur zugeordnet werden. Meist werden diese dann von einem äußeren 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, dafür aber einen unterschiedlichen Wert zugewiesen. Es gibt explizit keinen umschließenden Container. Wenn abwechselnd Optionsfelder angeklickt werden, werden immer alle anderen zu der Gruppe gehörige Optionsfelder deselektiert. Abschließend 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 besteht aus gewöhnlichem 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 110: Ein einzelnes Optionsfeld

11. Oder besser – eine Gruppe, denn ein einzelnes Optionsfeld macht keinen Sinn.

Beispiel (*form2.html*):

```
01 <html>
02 <body>
03 <form action="" method="" target="">
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 111: Verschiedene Radiobuttons in einer Gruppe

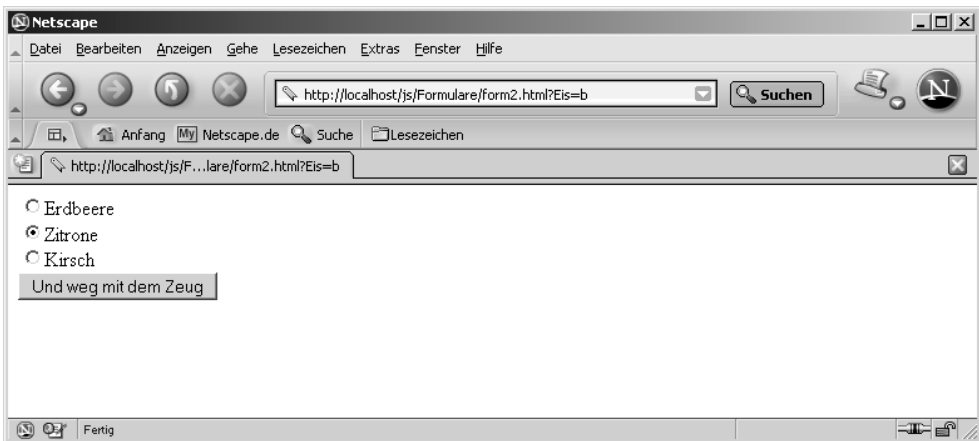


Abbildung 57: Verschiedene Radiobuttons einer Gruppe (durch den gleichen name-Wert der Gruppe zugeordnet) – beachten Sie die Adressleiste des Browsers.

Achtung

Beachten Sie, dass in einer Webseite auch alle Optionsfelder einer Gruppe deselektiert sein können. Im Beispiel ist in Zeile 5 das Attribut `checked` gesetzt. Wenn Sie dieses Attribut nicht setzen, wird beim Laden der Webseite keiner der Optionsfelder selektiert sein. Sobald allerdings ein Anwender ein Optionsfeld ausgewählt hat, wird immer genau ein Element einer Gruppe selektiert sein.

48 Wie kann ich rein mit HTML ein einzeliges Eingabefeld realisieren, in dem nur die Eingabe eines Kalenderdatums erlaubt ist?

Über den `<input>`-Tag mit der type-Angabe "date" können Sie ein einzeliges Eingabefeld generieren, in dem nach den offiziellen 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 einzelliges Eingabefeld. Mit anderen Worten – mit HTML können Sie dieses Problem nicht lösen. Sie sind auf eine Ergänzungstechnologie angewiesen, etwa JavaScript, was eine ideale Lösung darstellt. Allerdings erweist sich auch damit die Lösung als nicht ganz trivial (siehe dazu das Rezept »Wie kann ich ein einzeliges Eingabefeld realisieren, in dem nur die Eingabe eines Kalenderdatums erlaubt ist?« auf Seite 380).

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 einzelliges Eingabefeld zugreifen soll, und eine fremde Webseite übernehmen, sollten Sie diese Ausprägung von `type` einkalkulieren (siehe dazu auch das Rezept »Wie bestimme ich den Typ eines Formularelements?« auf Seite 247).

49 Wie kann ich rein mit HTML ein einzeliges Eingabefeld realisieren, in dem nur die Eingabe von Dezimalkommazahlen gestattet ist?

Über den `<input>`-Tag mit der `type`-Angabe `"float"` können Sie ein einzeliges Eingabefeld generieren, in dem nach den offiziellen 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 einzelliges Eingabefeld. Mit anderen Worten – mit 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 gewährleisten, dass nur die Eingabe von Dezimalkommazahlen gestattet ist?« auf Seite 336).

Hinweis

Sie sollten damit rechnen, dass Ersteller einer Webseite diese Ausprägung aus Unwissenheit ob der Ignoranz der Browser dennoch verwenden. Das W3C hat sie explizit vorgesehen. Wenn Sie also ein JavaScript erstellen, das auf ein entsprechendes einzelliges Eingabefeld zugreifen soll, und eine fremde Webseite übernehmen, sollten Sie diese Ausprägung von `type` einkalkulieren (siehe dazu auch das Rezept »Wie bestimme ich den Typ eines Formularelements?« auf Seite 247).

50 Wie kann ich rein mit HTML ein einzeliges Eingabefeld realisieren, in dem nur die Eingabe von ganzen Zahlen gestattet ist?

Über den `<input>`-Tag mit der `type`-Angabe `"int"` können Sie ein einzeliges Eingabefeld generieren, in dem nach den offiziellen HTML-Vorgaben nur die Eingabe von ganzen Zahlen erlaubt ist. Nur – das Attribut wird von keinem Browser berücksichtigt! Ein Besucher kann

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

trotz der Angabe `int` beliebige Daten in das Eingabefeld eintragen. Es verhält sich wie ein gewöhnliches einzeliliges Eingabefeld. Mit anderen Worten – mit 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 gewährleisten, dass nur die Eingabe von Ganzzahlen gestattet ist?« auf Seite 299*).

Hinweis

Sie sollten damit rechnen, dass Ersteller einer Webseite diese Ausprägung aus Unwissenheit ob der Ignoranz der Browser dennoch verwenden. Das W3C hat sie explizit vorgesehen. Wenn Sie also ein JavaScript erstellen, das auf ein entsprechendes einzeliliges Eingabefeld zugreifen sollen und eine fremde Webseite übernehmen, sollten Sie diese Ausprägung von `type` einkalkulieren (*siehe dazu auch das Rezept »Wie bestimme ich den Typ eines Formularelements?« auf Seite 247*).

51 Wie kann ich rein mit HTML ein einzeliliges Eingabefeld realisieren, in dem nur die Eingabe eines URL gestattet ist?

Über den `<input>`-Tag mit der `type`-Angabe `"url"` können Sie ein einzeliliges Eingabefeld generieren, in dem nach den offiziellen HTML-Vorgaben nur die Eingabe eines 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 einzeliliges Eingabefeld. Mit anderen Worten – mit HTML können Sie dieses Problem nicht lösen. Sie sind auf eine Ergänzungstechnologie angewiesen, etwa JavaScript, was eine ideale Lösung darstellt. Allerdings lässt sich das Rezept nicht ganz trivial realisieren und nur strukturell einen gültigen URL testen (*siehe dazu das Rezept »Wie kann ich ein Eingabefeld realisieren, in dem nur die Eingabe eines URL gestattet ist?« auf Seite 371*).

Hinweis

Sie sollten damit rechnen, dass Ersteller einer Webseite diese Ausprägung aus Unwissenheit ob der Ignoranz der Browser dennoch verwenden. Das W3C hat sie explizit vorgesehen. Wenn Sie also ein JavaScript erstellen, das auf ein entsprechendes einzeliliges Eingabefeld zugreifen soll, und eine fremde Webseite übernehmen, sollten Sie diese Ausprägung von `type` einkalkulieren (*siehe dazu auch das Rezept »Wie bestimme ich den Typ eines Formularelements?« auf Seite 247*).

52 Wie kann ich rein mit HTML ein einzeliliges Eingabefeld realisieren, in dem nur Werte kleiner als ein vorgegebener Grenzwert gestattet sind?

Über den `<input>`-Tag mit den optionalen Attributen `max=[numerischer Wert]` können Sie nach den offiziellen HTML-Vorgaben rein aus 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 nichtnumerisch und von beliebiger

Größe). Es verhält sich wie ein gewöhnliches einzeliges Eingabefeld. Mit anderen Worten – mit 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 kleiner als ein vorgegebener Grenzwert eingegeben werden?« auf Seite 347*).

Hinweis

Sie sollten damit rechnen, dass Ersteller einer Webseite dieses Attribut aus Unwissenheit ob der Ignoranz der Browser dennoch verwenden. Das W3C hat sie explizit vorgesehen. Wenn Sie also eine fremde Webseite übernehmen, in der ein maximaler Wert in einem Eingabefeld nur erlaubt sein soll, müssen Sie diesen Fehler korrigieren.

53 Wie kann ich rein mit HTML ein einzeliges Eingabefeld realisieren, in das nur Werte größer als ein vorgegebener Grenzwert gestattet werden?

Über den `<input>`-Tag mit dem optionalen Attributen `min=[numerischer Wert]` können Sie nach den offiziellen HTML-Vorgaben rein aus HTML heraus eine untere Grenze für erlaubte Wert bei numerischen Eingaben festlegen. 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 nichtnumerisch und von beliebiger Größe). Es verhält sich wie ein gewöhnliches einzeliges Eingabefeld. Mit anderen Worten – mit 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 353*).

Hinweis

Sie sollten damit rechnen, dass Ersteller einer Webseite dieses Attribut aus Unwissenheit ob der Ignoranz der Browser dennoch verwenden. Das W3C hat sie explizit vorgesehen. Wenn Sie also eine fremde Webseite übernehmen, in der ein maximaler Wert in einem Eingabefeld nur erlaubt sein soll, müssen Sie diesen Fehler korrigieren.

54 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 den `<input>`-Tag mit der `type`-Angabe `"file"` können Sie ein Standarddateiauswahlfenster des Betriebssystems realisieren, das die dort durch den Anwender ausgewählte Datei in das damit gleichzeitig spezifizierte Eingabefeld übernimmt. Dies kann beispielsweise für die Angabe einer Datei angewendet werden, die per Upload zu einem Server übertragen werden soll.

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

Die Angabe `type= "file"` sorgt nur für eine Referenz auf die Datei und noch für keine wirkliche Funktionalität.

Beispiel (*form3.html*):

```
01 <html>
02 <body>
03 <form action="" method="" target="">
04 Angabe der Datei:
05 <input type="file" size="60" name="uploadDatei">
06 <br>
07 <input type="Submit" value="Heb das Zeug hoch">
08 </form>
09 </body>
10 </html>
```

Listing 112: 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.

55 Wie kann ich ein verstecktes Formularfeld realisieren?

Einer der Gründe, warum HTML nicht als vollständige Programmiersprache bezeichnet werden kann, ist das Fehlen von Variablen. Aber in Form von versteckten Formularfeldern steht auch in HTML eine Art von Variablen zur Verfügung. Sie können in diesen Formularfeldern Informationen vor den Blicken des Besuchers versteckt temporär zwischenspeichern. Nur können Sie rein mit HTML auf diese Variablen nicht zugreifen, und damit sind versteckte Formularfelder im Grunde Variablen für Ergänzungstechnologien wie JavaScript (obwohl rein mit HTML generiert).

Ein verstecktes Eingabefeld zum unsichtbaren Speichern von Daten erzeugen Sie mit dem Parameter `type="hidden"`. Direkt in der Weboberfläche von einem Formular ist es unsichtbar und damit nicht direkt dem Anwender über den Browser zugänglich. Man kann es aber sehr gut verwenden, um per JavaScript Daten hineinzuschreiben oder auszulesen. Eine andere Anwendung ist das per HTML hartkodierte Speichern von Daten, die auf jeden Fall beim Absenden eines Formulars übermittelt werden müssen (und die ein Besucher der Webseite nicht sehen braucht).

Beispiel (*form4.html*):

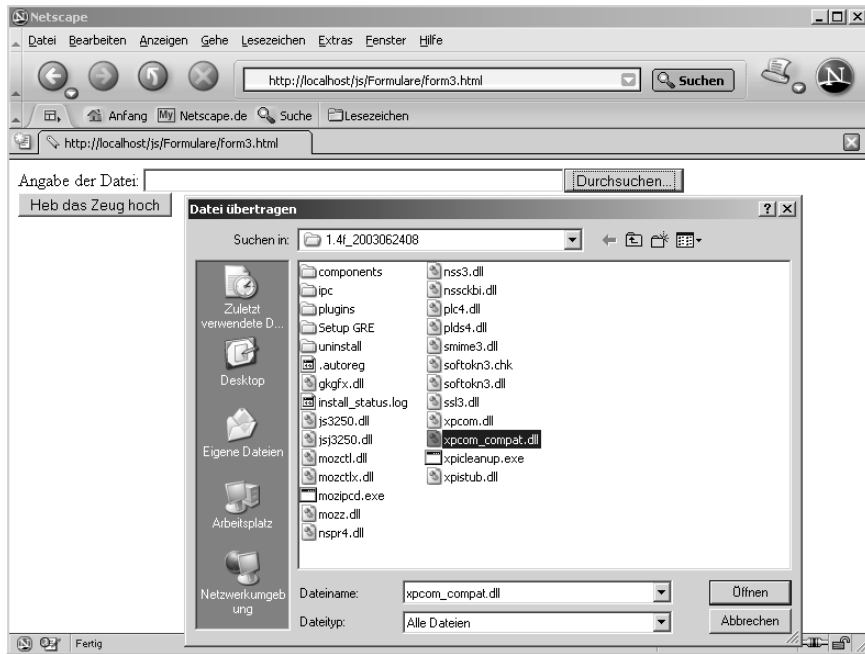


Abbildung 58: Die Dateiauswahl für den Upload – beachten Sie das einzeilige Eingabefeld und die nachfolgende Durchsuchen-Schaltfläche.

Hinweis

Beachten Sie, dass Sie die Beschriftung der Schaltfläche in diesem Fall nicht explizit angeben (können).

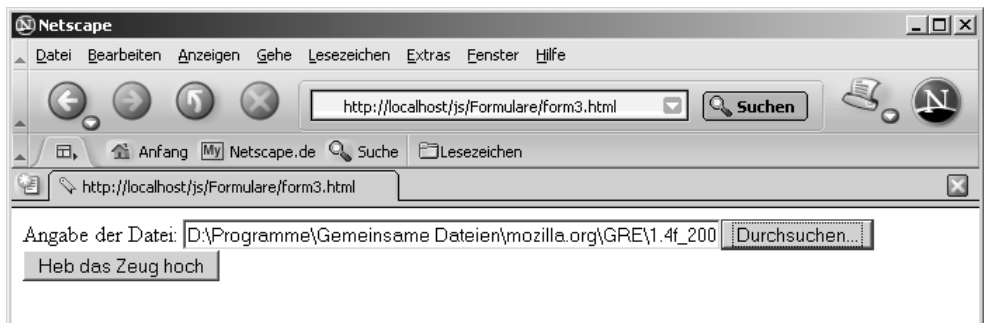


Abbildung 59: Wenn Sie eine Datei ausgewählt haben, wird diese in das einzeilige Listenfeld übernommen.

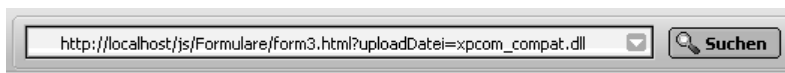


Abbildung 60: Nach dem Abschicken der Datei enthält der Pseudo-URL die ausgewählte Datei.

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

HTML

Events

Wenn Sie bei einem Dateifeld die `value`-Eigenschaft spezifizieren, wird diese nicht als Vorgabewert im Eingabefeld angezeigt.

```

01 <html>
02 <body>
03 <form action="" method="get">
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 113: Zwei versteckte Felder

In den Zeilen 4 und 5 sehen Sie die Definition von zwei versteckten Feldern.



Abbildung 61: Ein (scheinbar) leeres Formular – beachten Sie die Adresszeile des Browsers nach dem Abschicken des Formulars.

56 Wie kann ich ein mehrzeiliges Eingabefeld generieren?

Mehrzeilige Eingabefelder in einem Webformular werden mit dem `<textarea>`-Container aufgebaut. Ohne weitere Angaben bekommt das mehrzeilige Eingabefeld vom Webbrowser eine Defaultgröße zugewiesen.

Beispiel:

```

Ihr Kommentar:
<textarea></textarea>

```

Listing 114: Ein mehrzeiliges Eingabefeld mit Defaultgröße

Eine (optionale, aber in der Regel äußerst sinnvoll) Größenangabe erfolgt in Form von der Anzahl an Zeilen (`rows`=[numerischer Wert]) und Spalten (`cols`=[numerischer Wert]).

Beispiel (aus der Datei `form5.html`):

Achtung

Es ist nicht sinnvoll, sich auf die Defaultgröße für ein Texteingabefeld zu verlassen. Die unterschiedlichen Browser verhalten sich dabei sehr unterschiedlich. Insbesondere der Netscape Navigator verwendet nur eine sehr kleine Defaultgröße, mit der ein Anwender kaum etwas anfangen kann.

Ihr Kommentar:

```
<textarea name="Kommentar" cols=80 rows=6>
</textarea>
```

Listing 115: Ein Eingabefeld mit 80 Zeichen Breite und sechs Zeilen

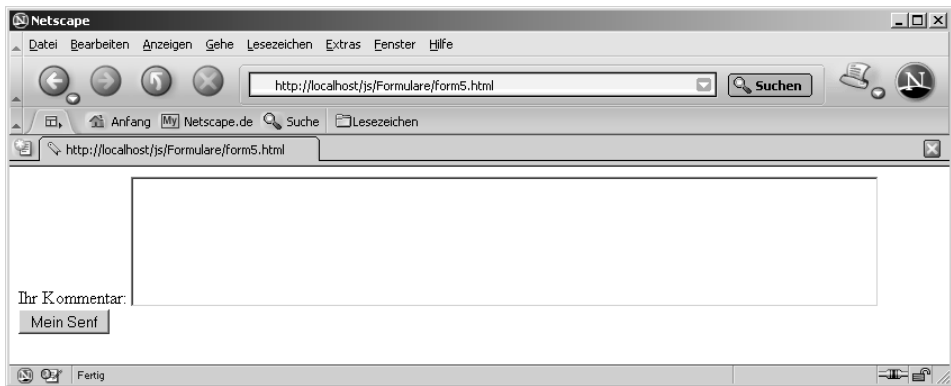


Abbildung 62: Ein mehrzeiliges Eingabefeld

Die Größenangaben bei einem mehrzeiligen Listenfeld beschränken nur die sichtbaren Zeichen, nicht die Anzahl der Zeichen, welche eingegeben werden dürfen. Sowohl die Zeilen als auch in einer Zeile eingebbaren Zeichen beschränken nur die Plattform (und natürlich der sinnvolle Umgang mit dem Eingabefeld) und die Anzahl. Wenn die Anzahl der durch einen Anwender eingegebenen Zeichen die maximal anzeigbaren Zeichen überschreitet, wird entweder automatisch seitwärts oder horizontal gescrollt. Mehrzeilige Eingabefelder werden von den meisten Browsern mit Bildlaufleisten angezeigt, die bei Überschreitung des Anzeigebeereichs durch die Menge des Textes automatisch aktiviert, andernfalls deaktiviert werden.

57 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 den 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.

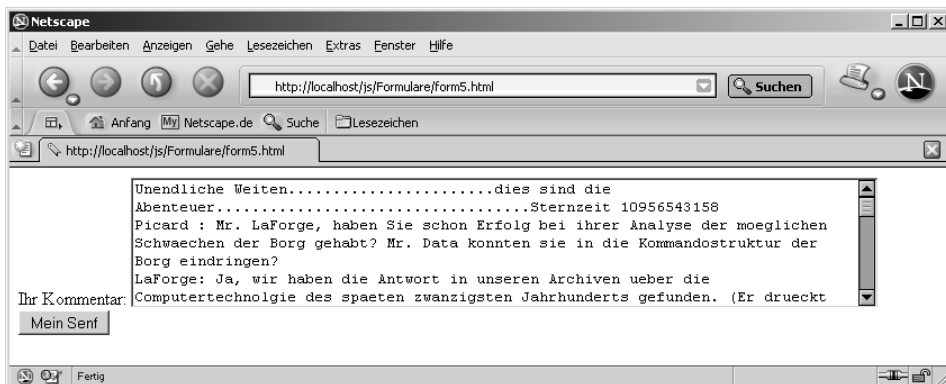


Abbildung 63: Beim Überschreiten des sichtbaren Bereichs eines mehrzeiligen Eingabefeldes generiert ein Browser Bildlaufleisten.

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 per Zeilenschaltung manuell durch den Anwender erfolgen.

Tipp

Sie können innerhalb des `<textarea>`-Containers einen Text als Vorbelegung definieren.

Beispiel:

```
Ihr Kommentar: <textarea name="Kommentar" cols=80 rows=2>
Ist das so schön bunt hier!
</textarea>
```

Listing 116: Ein vorbelegtes Eingabefeld mit 80 Zeichen Breite und zwei Zeilen

Hinweis

Beim `<option>`-Tag hat der optionale `value`-Parameter nicht die Bedeutung, die er bei den meisten anderen 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 einzeliges 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 117: Eine schematische Auswahlliste

Beispiel (*form6.html*):

```

<html>
<body>
<form action="" method="get">
<select name="eissorte">
  <option>Erdbeere</option>
  <option>Vanille</option>
  <option>Schoko</option>
  <option>Zitrone</option>
</select><br>
<input type="Submit" value="Ihre Bestellung bitte">
</form>
</body>
</html>

```

Listing 118: Ein Formular mit Auswahlliste



Abbildung 64: Eine einzeilige Auswahlliste

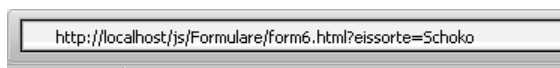


Abbildung 65: Der Wert für die Auswahlliste (über *name* bei *<select>* benannt) ist der Eintrag im *<option>*-Container.

Ein Listeneintrag kann im Prinzip über einen `<option>`-Container als auch über einen einzelnen `<option>`-Tag (ohne Ende-Tag) formuliert werden. 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.

58 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 den Tag `<option>` spezifiziert (siehe oben). Der `<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 überschreiten, 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 119: 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><hr>
10 <input type="Submit" value="Ihre Bestellung">
11 </form>
12 </body>
13 </html>
```

Listing 120: Ein Container für eine mehrzeilige Auswahlliste

In Zeile 4 wird die Größe der Auswahlliste auf den Wert 5 gesetzt. Die Anzahl der Einträge ist jedoch nur 4.

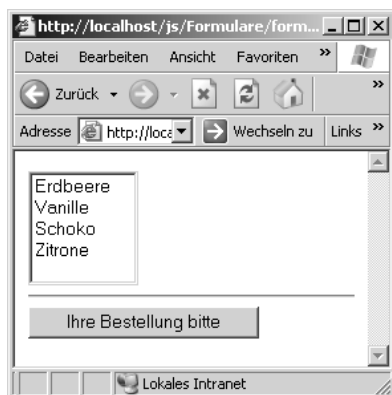


Abbildung 66: Die Anzahl der Einträge übersteigt den verfügbaren Platz nicht – keine Bildlaufleisten im Internet Explorer.



Abbildung 67: Die Anzahl der Einträge übersteigt den verfügbaren Platz nicht – dennoch Bildlaufleisten im Netscape Navigator.

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events



Abbildung 68: Mehr Einträge als Platz zur Verfügung – in allen Browsern werden dann Bildlaufleisten angezeigt.

59 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 den Tag `<option>` spezifiziert (siehe oben). In der Grundeinstellung ist dabei keine Mehrfachauswahl in einer Liste selbstverständlich. Dies hängt an den Einstellungen des Browsers, ob es standardmäßig erlaubt oder verboten ist. Eine explizite Erlaubnis von 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="[Listenname]" size=[numerischer Wert] multiple>
<option> [Listeneintrag 1]</option>
...
<option> [Listeneintrag n]</option>
</select>
```

Listing 121: 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>
```

Listing 122: Eine Auswahlliste mit explizit erlaubter Mehrfachauswahl


```

06 <option>Vanille</option>
07 <option>Schoko</option>
08 <option>Zitrone</option>
09 </select><hr>
10 <input type="Submit" value="Ihre Bestellung">
11 </form>
12 </body>
13 </html>

```

Listing 122: Eine Auswahlliste mit explizit erlaubter Mehrfachauswahl (Forts.)

In Zeile 4 wird explizit die Mehrfachauswahl erlaubt.



Abbildung 69: Mehrfachauswahl in einer Liste

60 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 den 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` (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>
...

```

Listing 123: Eine schematische Auswahlliste mit vorselektiertem Eintrag

```
<option> [Listeneintrag m]</option>
</select>
```

Listing 123: Eine schematische Auswahlliste mit vorselektiertem Eintrag (Forts.)

Beispiel:

```
<select name="eissorte" size="5" >
  <option>Erdbeere</option>
  <option>Vanille</option>
  <option>Schoko</option>
  <option selected >Zitrone</option>
</select>
```

Listing 124: Eine Auswahlliste mit vorselektiertem Eintrag

Tipp

Obwohl es nicht XHTML-konform ist, können Sie ohne irgendwelche Probleme auf den Abschluss-Tag `</option>` verzichten.

61 Wie kann ich ohne ein Webformular mit einem Besucher per JavaScript interagieren?

Sie benötigen nicht in jedem Fall ein Webformular, um mit dem Anwender eine Interaktion zu haben oder Benutzerdaten entgegenzunehmen. Das `window`-Objekt stellt Ihnen drei Standardmethoden bereit, die Ihnen mit Dialogfenstern die Interaktion mit einem Anwender gestatten.

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 Hochkomma gesetzt werden.

Beispiel (*alert.html*):

```
01 <html>
02 <body>
03 <form action="" method="get" name="f">
04   <input type="button" value="OK" onClick="alert('Hallo')">
05 </form>
06 </body>
07 </html>
```

Listing 125: 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.



Abbildung 70: Ein `alert()`-Fenster

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.

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 Hochkomma gesetzt werden.

Beispiel (`confirm.html`):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04 <!--
05 if(confirm("Wollen Sie ein Bier?")) {
06   document.write("<h1>Prost</h1>");
07 }
08 else {
09   document.write("<h1>Ihr Pech</h1>");
10 }
11 //-->
12 </script>
13 <noscript></noscript>
```

Listing 126: Eine einfache Anwendung von `confirm()`

```
14 </body>
15 </html>
```

Listing 126: Eine einfache Anwendung von `confirm()` (Forts.)

In den Zeilen 3 bis 12 des Listings befindet sich ein `<script>`-Container, dessen Anweisungen beim Laden der Webseite automatisch ausgeführt werden. Beim Laden der Webseite wird also der `confirm`-Dialog angezeigt.



Abbildung 71: Beim Laden der Webseite wird der `confirm`-Dialog angezeigt.

In der `if`-Bedingung in Zeile 5 wird der Rückgabewert der `confirm()`-Methode direkt ausgewertet und abhängig davon eine unterschiedlich gestaltete Webseite angezeigt.

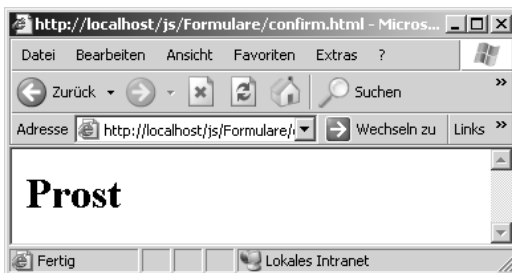


Abbildung 72: Die Seite nach Betätigen der OK-Schaltfläche



Abbildung 73: Die Seite nach Betätigen der Abbrechen-Schaltfläche

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 127: Die Syntax der `prompt()`-Methode

Die Methode fordert den Anwender in einem Dialogfenster mit einer - und einer -Schaltfläche zu einer Eingabe in einem Feld auf und gibt diesen Wert an die aufrufende Funktion zurück, wenn Sie die -Schaltfläche betätigen. Klickt ein Anwender auf , wird der definierte 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 (" " für keine Vorbelegung).

Beispiel (*prompt.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04 <!--
05   document.write("Ihre Eingabe war: " + prompt("Geben Sie was ein",""));
06 //-->
07 </script>
08 <noscript></noscript>
09 </body>
10 </html>
```

Listing 128: Eine einfache Anwendung von `prompt()`

In den Zeilen 3 bis 7 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.

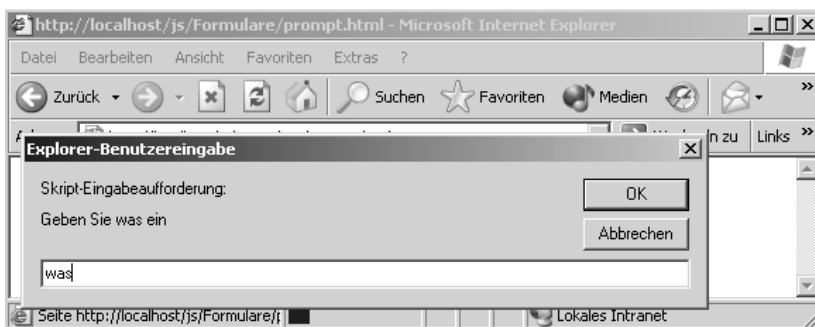


Abbildung 74: Der `prompt`-Dialog fordert einen Anwender zur freien Texteingabe auf.

Wenn der Anwender einen Text eingegeben und die -Schaltfläche betätigt hat, wird mit `document.write()` der Inhalt mit vorangestelltem Standardtext `Ihre Eingabe war:` in die Webseite geschrieben.



Abbildung 75: Eingabe eines Textes und Bestätigung

Wenn der Anwender die -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 wird nicht zurückgegeben.



Abbildung 76: 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.

62 Wie greife ich mit 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- als auch programmiersprachenübergreifende Schnittstelle bezeichnet. In diesem Konzept wird eine 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 in der DOT-Notation, indem man erst den Namen des oberen Objekts und dann den des darunter angesiedelten Objekts hinschreibt. 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 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.

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 129: Zugriff auf ein Formular in einer Webseite

Beispiel:

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

Listing 130: Zugriff auf das dritte Formular in einer Webseite

Zugriff über einen Namen

Sollte ein Formular im HTML-Tag mit einem `name`-Attribut spezifiziert sein¹², können Sie statt einem Datenfeldelement des Objektfeldes ebenso diesen Namen für den Zugriff verwenden.

12. Oder es wurde vorher mit JavaScript der Eigenschaft `name` ein Wert zugewiesen.

Beispiel:

```
<form name="adresse">
  ...// Formularelemente
</form>
```

Listing 131: Der Formularcontainer in HTML

```
window.document.adresse
```

Listing 132: Der Zugriff aus JavaScript auf das Formular

Achtung

Beachten Sie, dass beim Namen Groß- und Kleinschreibung relevant sind. Obwohl der Wert im HTML-Container gesetzt wird und Groß- und Kleinschreibung des Namens eines HTML-Elements bei Zugriffen aus HTML (etwa bei einer `target`-Angabe) irrelevant ist, spielt es beim Zugriff aus JavaScript eine Rolle.

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 Konstellation (das aktuelle Objekt muss die Webseite) auf das Formular zugreifen. *Für eine konkrete Anwendung betrachten Sie bitte die Rezepte »Wie kann ich die Formulardaten ohne einen Submit-Button wegschicken?« auf Seite 229 oder »Wie kann ich die Formulardaten ohne einen Reset-Button zurücksetzen?« auf Seite 234.*

Tipp

Unter gewissen Umständen kann ein Zugriff über die DOT-Notation auf Objekte in tieferen Ebenen der Objekthierarchie um die vorangestellte Angabe der Objekte verkürzt werden, die für einen Zugriff offensichtlich vorangestellt werden müssen. Sie können deshalb bei den meisten Zugriffen auf Bestandteile des Browserfensters 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`.

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 HTML-Tag verfügt. Ein Zugriff über den Namen ist meist besser lesbar und toleranter gegenüber Änderungen des Aufbaus der Webseite. Wenn etwa vor einem Formular ein neues Formular in der Web-

seite eingefügt wird, müssen Zugriffe über einen Index angepasst werden, während Zugriffe über einen Namen keiner Änderung bedürfen. Der Zugriff über eine Referenz auf Basis von `this` ist am universellsten einsetzbar, aber geht nicht in jeder Konstellation. Im Grunde bleibt es Ihnen aber in vielen Situationen selbst überlassen, welche Version Sie vorziehen.

Achtung

Beachten Sie, dass Sie nicht per JavaScript auf ein Webformular zugreifen können, bevor vom Browser die dazu notwendige 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 HTML-Anweisungen zum Aufbau des Webformulars fertig interpretiert hat, wird der JavaScript-Interpreter einen Fehler melden.

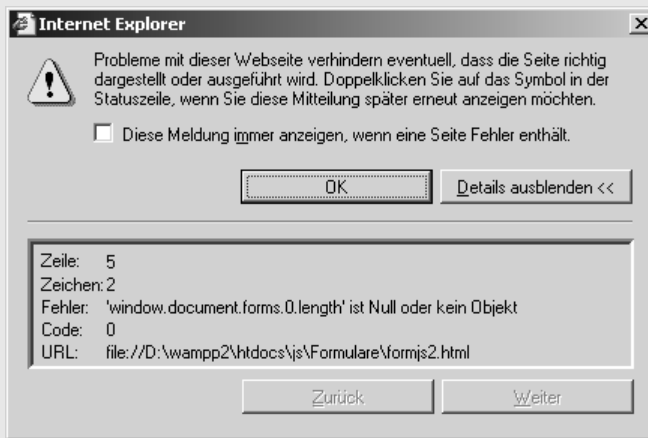


Abbildung 77: Zugriff auf ein Webformular, bevor es mit HTML erzeugt wurde

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. Ober wenn Sie einen Skriptcontainer vor dem `<body>`-Tag notieren und in dem Skript-Container JavaScript-Anweisungen ausführen, die die Existenz eines Formulars bereits voraussetzen. Indem Sie statt der direkten Notation von Aufrufen in so einem Skript-Container den Eventhandler `onLoad` verwenden, können Sie das letzte Problem zumindest umgehen. Der Eventhandler wird erst ausgeführt, wenn die Webseite bezüglich ihrer HTML-Struktur vollständig geladen wurde. Für ein Beispiel siehe Seite 238.

63 Wie bestimme ich die Anzahl der Elemente in einem Formular?

Jede Objektrepräsentation eines Webformulars verfügt über die Eigenschaft `length`. Diese Eigenschaft enthält die Anzahl der Formularelemente in dem Formular. Sie können sowohl über das Objektfeld als auch den Namen eines Formulars auf die Eigenschaft zugreifen.

Beispiel (*formjs1.html*):

```
01 <html>
02 <script language="JavaScript">
03 <body>
04 <form action="" method="get" name="a">
05 <input>
06 <input>
07 <input>
08 <br>
09 <input type="Submit" value="Ihre Bestellung">
10 </form>
11 <script language="JavaScript">
12 <!--
13   alert(window.document.forms[0].length);
14 //-->
15 </script>
16 </body>
17 </html>
```

Listing 133: Die Anzahl der Formularelemente im ersten Webformular

In der Webseite gibt es ein Formular mit vier Elementen. In Zeile 13 wird die Anzahl ausgegeben.

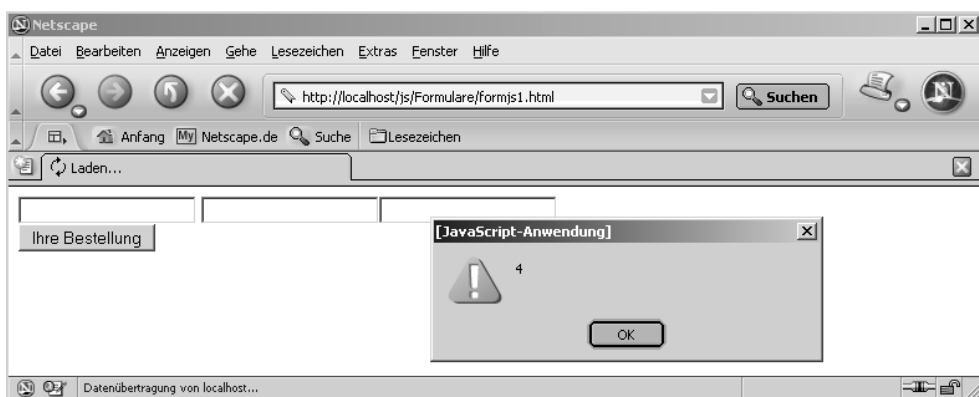


Abbildung 78: Die Anzahl der Formularelemente inklusive der Submit-Schaltfläche

Sie können bei einem Webformular die Eigenschaft `length` nur lesen. 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 134: Eine sinnlose Zuweisung

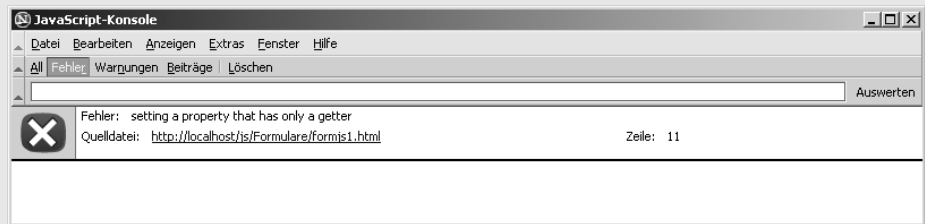


Abbildung 79: In der JavaScript-Konsole des Netscape Navigators wird der Fehler beschrieben.

64 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 HTML-Struktur des Formulars. Sie können sowohl über das Objektfeld `forms[]` als auch die Namenseigenschaft `name` eines Formulars 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)¹³.

Beispiel (*formjs2.html*):

```
01 <html>
02 <body>
03 <form action="" method="get" name="a">
04 <input>
05 <input>
06 <input>
07 <br>
08 <input type="Submit" value="Ihre Bestellung">
09 </form>
10 <hr>
11 <script language="JavaScript">
12 <!--
13 document.write("Name des Formulars, wie es per HTML gesetzt wurde: "
```

Listing 135: Abfrage und Veränderung des Namens im ersten Webformular

13. Hierbei kann ein Zugriff über den alten Namen zum Beispiel sinnvoll sein.

```
14 + window.document.forms[0].name + "<br>");  
15 window.document.forms[0].name = "b";  
16 document.write("Name des Formulars, wie es per JavaScript gesetzt wurde: "  
17 + window.document.forms[0].name + ".");  
18 //-->  
19 </script>  
20 </body>  
21 </html>
```

Listing 135: Abfrage und Veränderung des Namens im ersten Webformular (Forts.)

In Zeile 14 wird der Name des Formulars abgefragt, wie er mit HTML gesetzt wurde, und in Zeile 17 der neue Name ausgegeben, der in Zeile 15 mit JavaScript geändert wurde.

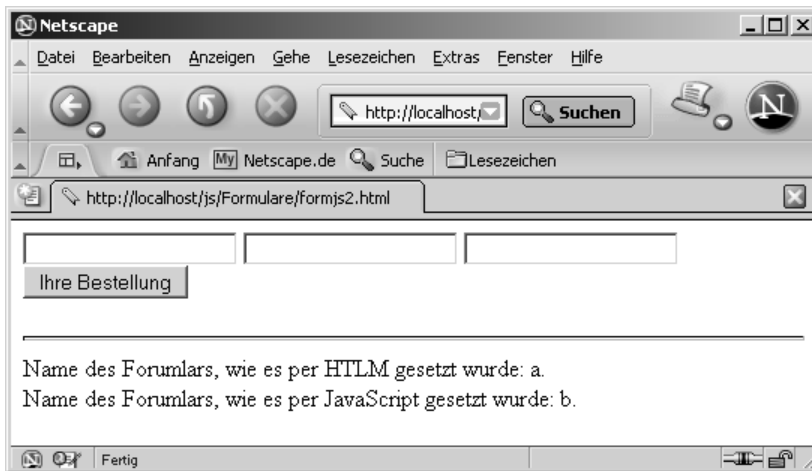


Abbildung 80: Lese- und Schreibzugriff auf den Namen des Webformulars

65 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 HTML-Struktur des Formulars. Sie können sowohl über das Objektfeld `forms[]` als auch die Namens-eigenschaft `name` eines Formulars auf die `method`-Eigenschaft zugreifen. Sie können die Versandmethode eines Formulars sowohl abfragen (Lesezugriff) als auch per JavaScript setzen (Schreibzugriff).

Beispiel (*formjs3.html*):

```

01 <html>
02 <body>
03 <form action="" method="get" name="a">
04 <input>
05 <input>
06 <input>
07 <br>
08 <input type="Submit" value="Ihre Bestellung">
09 </form>
10 <hr>
11 <script language="JavaScript">
12 <!--
13   document.write(
14     "Versandmethode des Formulars, wie sie per HTML gesetzt wurde: " +
15     window.document.a.method + "<br>");
16   window.document.forms[0].method = "post";
17   document.write(
18     "Versandmethode des Formulars, wie sie per JavaScript gesetzt wurde: "
19     + window.document.a.method + ".");
20 //-->
21 </script>
22 </body>
23 </html>

```

Listing 136: Abfrage und Veränderung der Versandmethode im ersten Webformular – hier mit Zugriff über den Formularnamen als auch das Objektfeld

Das Listing enthält ein kleines Webformular, in dem mit HTML in Zeile 3 die Versandmethode `get` gesetzt wird. In Zeile 15 wird die Versandmethode des Formulars abgefragt, wie sie mit HTML gesetzt wurde, und in Zeile 19 die neue Versandmethode ausgegeben, die in Zeile 16 mit JavaScript geändert wurde.

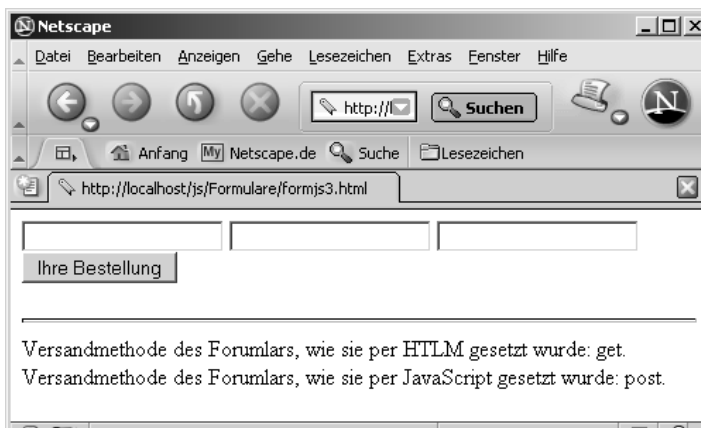


Abbildung 81: Lesen und Schreiben der Versandmethode

Ein kleines Rezept zum dynamischen Umschalten der Versandmethode

Die HTML-Parameter `method` (und damit die gleichnamige Eigenschaft der Objektrepräsentation) kann 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 137: Eine Funktion, die automatisch die voreingestellte Versandmethode eines Formulars umwandelt

In Zeile 2 fragen Sie die voreingestellte Methode des Webformulars ab, 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 138: Umwandeln der Versandmethode des ersten Formulars

Das nachfolgende Listing zeigt das Rezept in einem vollständigen Beispiel (*formversand-methodeaender.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function switchVersandMethode (formular) {
05   if(window.document.forms[formular].method.toUpperCase() == "POST") {
06     window.document.forms[formular].method = "GET";
07   }
08   else {
09     window.document.forms[formular].method = "POST";
10   }

```

Listing 139: Freie Auswahl der Versandmethode durch den Anwender

```
11 window.document.forms[formular].submit();
12 }
13 //-->
14 </script>
15 <body>
16 <form action="test.php" method="get">
17 <table >
18 <tr>
19 <td>
20 Userid
21 </td>
22 <td>
23 <input type="Text" name="user">
24 </td>
25 </tr>
26 <tr>
27 <td>
28 Passwort
29 </td>
30 <td>
31 <input type="Password" name="pw">
32 </td>
33 </tr>
34 <tr>
35 <td>
36 <script language="JavaScript">
37 <!--
38 document.write("Ihre Standardversandmethode ist <br>" +
39 window.document.forms[0].method + "<br>");
40 //-->
41 </script>
42 Zum Versenden mit dieser Methoden klicken Sie hier:
43 </td>
44 <td>
45 <input type="Submit">
46 </td>
47 </tr>
48 <tr>
49 <td>
50 <script language="JavaScript">
51 <!--
52 if(window.document.forms[0].method.toUpperCase() == "POST") {
53 document.write(
54 "Zum Versenden mit der alternativen Versandmethode GET <br>" +
55 " klicken Sie hier:");
56 }
57 else {
58 document.write(
59 "Zum Versenden mit der alternativen Versandmethode POST <br>" +
60 " klicken Sie hier:");
61 }
```

Listing 139: Freie Auswahl der Versandmethode durch den Anwender (Forts.)

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

HTML

Events

```

62  //-->
63  </script>
64  </td>
65  <td>
66  <input type="Button" onClick="switchVersandMethode(0)"
67    value="Ändern der Versandmethode">
68  </td>
69  </tr>
70  <tr>
71  </tr>
72  <tr>
73  <td>
74    Formular zurücksetzen
75  </td>
76  <td>
77    <input type="reset">
78  </td>
79  </tr>
80  </table>
81  </form>
82  </body>
83  </html>

```

Listing 139: Freie Auswahl der Versandmethode durch den Anwender (Forts.)

Von Zeile 4 bis 12 finden Sie die oben beschriebene Funktion zum Umschalten der Versandmethode. In 11 sehen Sie aber eine kleine Erweiterung (`window.document.forms[formular].submit();`). Mit dieser versenden Sie aus JavaScript heraus die Daten in dem Webformular (*siehe dazu das Rezept »Wie kann ich die Formulardaten ohne einen Submit-Button wegschicken?« auf Seite 229*). Mit der vorangestellten Funktionalität zum Umschalten der Versandmethode kann ein Anwender beim Aufruf der Funktion also immer mit der alternativen Versandmethode die Formulardaten verschicken.

In dem nachfolgenden Webformular haben Sie in einer Tabelle aufbereitet zwei Eingabefelder, deren Inhalt verschickt werden soll. Zeile 18 bis 33 – `<tr><td>Userid</td><td><input type="Text" name="user"></td></tr><tr><td>Passwort</td><td><input type="Password" name="pw"></td></tr>`. In den Zeilen 34 bis 47 ist eine weitere Tabellenzeile definiert. In der ersten Zelle darin wird mit einer JavaScript-Anweisung die eingestellte Versandmethode des Formulars abgefragt und per `document.write()` in die Zelle geschrieben und mit einem statischen Text ergänzt (`<td><script language="JavaScript"><!--document.write("Ihre Standardversandmethode ist <br>" + window.document.forms[0].method + "<br>");//--> </script>` Zum Versenden mit dieser Methoden klicken Sie hier: `</td>`). Die zweite Zelle definiert einen Submit-Button (`<td><input type="Submit"> </td>`). Wenn der Anwender darauf klickt, werden die Formulardaten mit der voreingestellten Versandmethode verschickt.

In den Zeilen 48 bis 69 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 von dem Formular, 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.

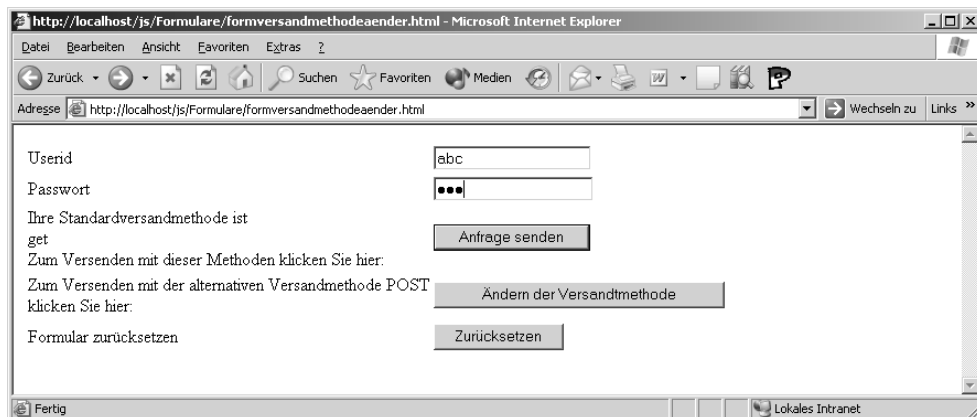


Abbildung 82: Das Formular vor dem Absenden

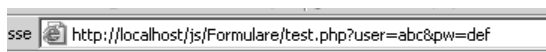


Abbildung 83: Die Daten wurden mit get verschickt.

66 Wie greife ich auf die Zieladresse eines Formulars zu?

Jede Objektrepräsentation eines Webformulars verfügt über die Eigenschaft `action`. Diese Eigenschaft enthält als String den Ziel-URL für den Versand der Daten des Formulars und korrespondiert unmittelbar mit dem gleichnamigen `action`-Parameter in der HTML-Struktur des Formulars. Sie können sowohl über das Objektfeld `forms[]` als auch die Namens-eigenschaft `name` eines Formulars auf die Eigenschaft zugreifen. Sie können den 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 <!--
11   document.write(
12     "Zieladresse des Formulars, wie sie per HTML gesetzt wurde: "
13     + window.document.a.action + "<br>");
14   window.document.forms[0].action = "http://www.webscripting.de";
15   document.write(
16     "Zieladresse des Formulars, wie sie per JavaScript gesetzt wurde: " +
17     window.document.a.action + ".");
18 //-->
19 </script>
20 </body>
21 </html>
```

Listing 140: Abfrage und Veränderung des Ziel-URLs im ersten Webformular – hier mit Zugriff über den Formularenamen als auch das Objektfeld

Das Listing enthält ein kleines Webformular, in dem mit HTML in Zeile 3 der *action*-Parameter gesetzt wird. In Zeile 13 wird dessen Wert abgefragt, wie er mit HTML gesetzt wurde, und in Zeile 17 der neue Wert ausgegeben, wie er in Zeile 14 mit JavaScript geändert wurde.

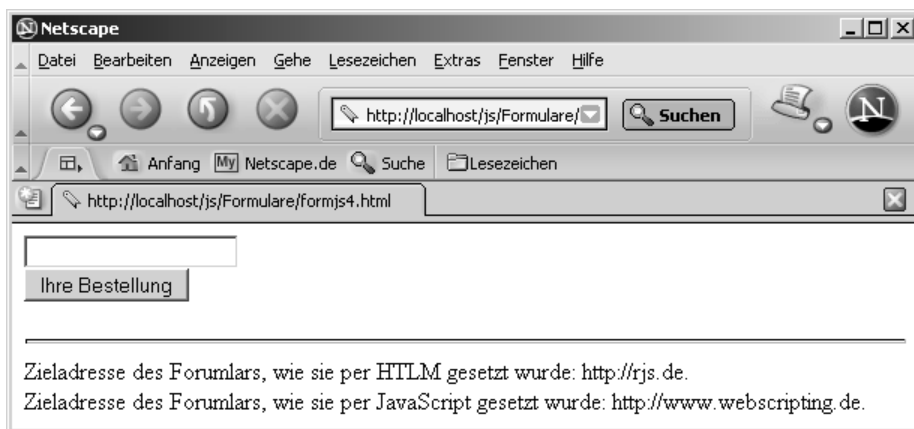


Abbildung 84: Lese- und Schreibzugriff auf *action*

67 Wie greife ich auf das Ziel der Antwort auf ein Formular zu?

Jede Objektrepräsentation eines Webformulars verfügt über die Eigenschaft `target`. Diese Eigenschaft enthält als String den Ziel-URL für die Antwort auf das Abschicken des Formulars und korrespondiert unmittelbar mit dem gleichnamigen `target`-Parameter in der HTML-Struktur des Formulars. Sie können sowohl über das Objektfeld `forms[]` als auch die Namens Eigenschaft `name` eines Formulars auf die Eigenschaft zugreifen. Sie können den Ziel-URL für den Versand der Daten eines Formulars sowohl abfragen (Lesezugriff) als auch per JavaScript setzen (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 <!--
11 document.write(
12 "Zieladresse der Antwort, wie sie per HTML gesetzt wurde: " +
13   window.document.a.target + "<br>");
14   window.document.a.target = "links";
15 document.write(
16 "Zieladresse der Antwort, wie sie per JavaScript gesetzt wurde: " +
17   window.document.a.target + ".");
18 //-->
19 </script>
20 </body>
21 </html>
```

Listing 141: Abfrage und Veränderung des Ziels der Antwort auf ein Webformular – hier mit Zugriff über den Formularnamen

Das Listing enthält ein kleines Webformular, in dem mit HTML in Zeile 3 der `target`-Parameter gesetzt wird. In Zeile 13 wird dessen Wert abgefragt, wie er mit HTML gesetzt wurde, und in Zeile 17 der neue Wert ausgegeben, wie er in Zeile 14 mit JavaScript geändert wurde.

Es gibt im Grunde nur wenige Anwendungen, das Ziel für die Antwort auf das Absenden der Formulardaten per JavaScript zu ändern. Aber es kann gelegentlich sinnvoll sein, aufgrund 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.

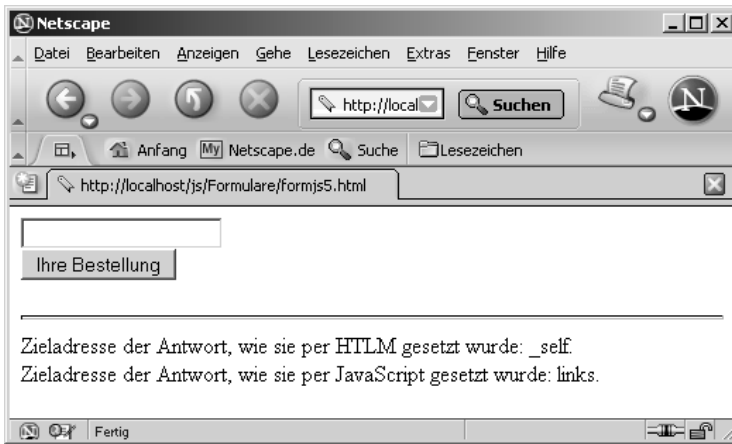


Abbildung 85: Lese- und Schreibzugriff auf target

Zuerst wollen wir einen verschachtelten Frameset erzeugen. Dieser soll aus drei Frames aufgebaut sein:

```

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 142: 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 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 jedem 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 <body>
03 <form action="formjs6.php" method="get" name="a" onSubmit="ziel()">
04 <input name="z">
05 <br>
06 <input type="Submit" value="OK">
07 </form>
08 <hr>
09 <script language="JavaScript">
10 <!--
11 function ziel() {
12     if(Math.random()*2 > 1) {
13         window.document.a.target = "links";
14     }
15     else {
16         window.document.a.target = "rechts";
17     }
18 }
19 //-->
20 </script>
21 </body>
22 </html>
```

Listing 143: Die Datei mit dem Webformular und der JavaScript-Funktion

Die Zeilen 3 bis 7 legen das Webformular mit einem Eingabefeld und einer Submit-Schaltfläche fest. Die JavaScript-Funktion `ziel()` (Zeilen 11 bis 18) setzt durch einen Zufallsmechanismus die `target`-Eigenschaft von dem Formular auf den Wert `"links"` oder `"rechts"`. Das sind die 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 3). Die Antwort des aufgerufenen PHP-Skriptes (es muss sich parallel zur HTML-Datei befinden) wird zufällig in dem einen oder anderen Frame ausgegeben.

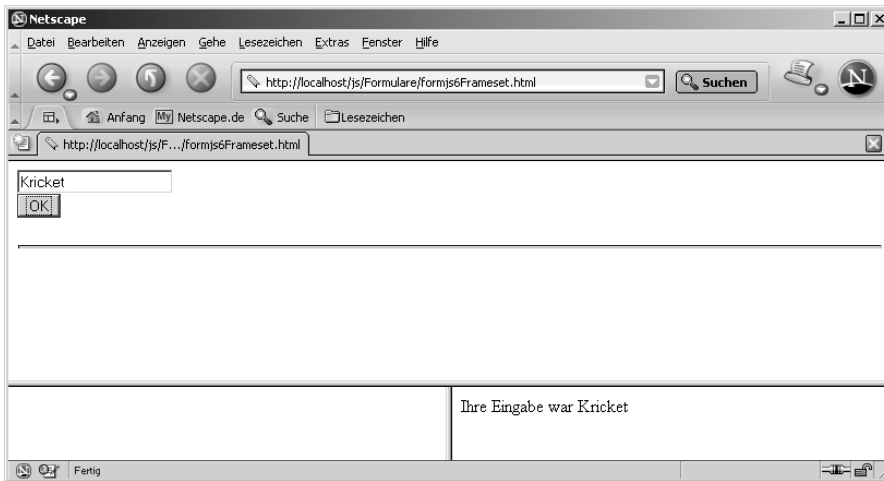


Abbildung 86: Die target-Angabe »rechts« wurde von ziel() generiert.

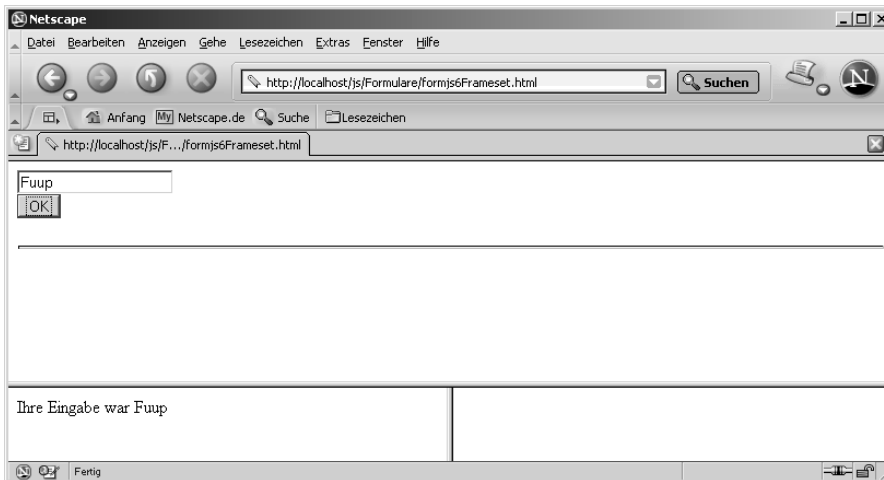


Abbildung 87: Hier erfolgt die Ausgabe mit target=»links«.

Hinweis

Obwohl es im Grunde irrelevant ist, hier zur Vollständigkeit noch das kleine PHP-Skript, das auf dem Server aufgerufen wird und die Antwort generiert:

```
<?Guten
echo "Ihre Eingabe war ".$z;
?>Guten
```

Listing 144: Das aufgerufene PHP-Skript

Sie sehen, dass darin keine Zielangabe für die Ausgabe der Antwort vom Server notiert ist.

68 Wie greife ich auf die Kodierung eines Formulars zu?

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 HTML mit dem Attribut `enctype` 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 HTML-Struktur des Formulars.

Sie können sowohl über das Objektfeld `forms[]` als auch die Namenseigenschaft `name` eines Formulars auf die Eigenschaft zugreifen. 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 action="" method="get" name="a">
04 <input name="z">
05 <br>
06 <input type="Submit" value="OK">
07 </form>
08 <hr>
09 <script language="JavaScript">
10 <!--
11   document.write(
12     "Kodierung des Formulars, wie sie per HTML gesetzt wurde: " +
13     window.document.a.encoding + "<br>");
14   window.document.a.encoding = "text/plain";
15   document.write(
16     "Kodierung des Formulars, wie sie per JavaScript gesetzt wurde: " +
17     window.document.a.encoding + ".");
18 //-->
19 </script>
20 </body>
21 </html>
```

Listing 145: Zugriff auf das Kodierungsverfahren des Formulars

Zeile 3 bis 7 definieren ein einfaches Webformular. In Zeile 13 fragen Sie die Kodierung ab, und in Zeile 14 wird eine neue Kodierung gesetzt (`window.document.a.encoding = "text/plain";`).



Abbildung 88: Die erste Ausgabe ist die Defaultkodierung, und anschließend folgt die neue Kodierung.

Achtung

Die Standardmethode zur Kodierung ist in den meisten Browsern `application/x-www-form-urlencoded`. Sie sollten allerdings beachten, dass *nicht* alle Browser die Korrespondenz zwischen dem 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 HTML `enctype` nicht explizit gesetzt wurde, wird beispielsweise beim Netscape Navigator 7.1 der Wert von `encoding` in der gleichen Situation leer sein.

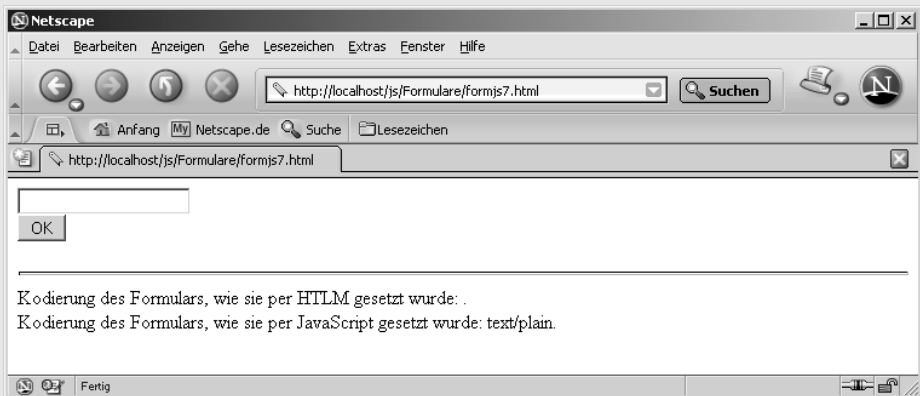


Abbildung 89: Der Wert von `encoding` ist nicht gesetzt, da `enctype` in HTML nicht explizit spezifiziert wurde.

69 Wie kann ich die Formulardaten ohne einen Submit-Button wegschicken?

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 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">
05 <input name="b">
06 <input name="b">
07 <br>
08 <input type="button" value="OK"
09   onClick="window.document.forms[0].submit()">
10 </form>
11 </body>
12 </html>
```

Listing 146: Die Anwendung der submit()-Methode – Auslösung mit onClick bei einem gewöhnlichen Button

Zeile 3 bis 10 definiert ein einfaches Webformular. In den Zeilen 8 und 9 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()">`).

Tipp

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 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 für das Beispiel `window.document.forms[0]`. Beachten Sie dazu das Beispiel *formjssubmit2.html* auf der Buch-CD.

Selbstverständlich kann ebenfalls eine Grafik als sensibles Element zum Verschicken von Formulardaten verwendet werden. Genau genommen kann jedes Webseitenelement verwendet werden, das auf einen Eventhandler wie `onClick` oder einen anderen sinnvollen Eventhandler reagieren kann. Da aber verschiedene Browser vollkommen unterschiedlich bei der Unterstützung von einzelnen Webseitenelementen im Zusammenhang mit Eventhandlern reagieren, müssen Sie hier ausführlich testen. Speziell in Hinsicht der Verwendung von Grafiken als sensibles Element zum Abschicken von Formulardaten können Sie aber einige nahezu immer funktionierende Tricks anwenden.

Tipp

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">
05 <input name="b">
06 <input name="b">
07 <br>
08 <a href="#" onClick="window.document.forms[0].submit()">
09 
10 </a>
11 </form>
12 </body>
13 </html>
```

Listing 147: Eine Grafik in einem Hyperlink als Submit-Element

Der sensitive Container erstreckt sich von Zeile 8 bis 10. Die Grafikreferenz in Zeile 9 »weiß« dabei nichts davon, dass ein Klick auf sie die Formulardaten absendet. Das obliegt ausschließlich dem umgebenden Container, bei dem `onClick` ausgelöst wird.

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.

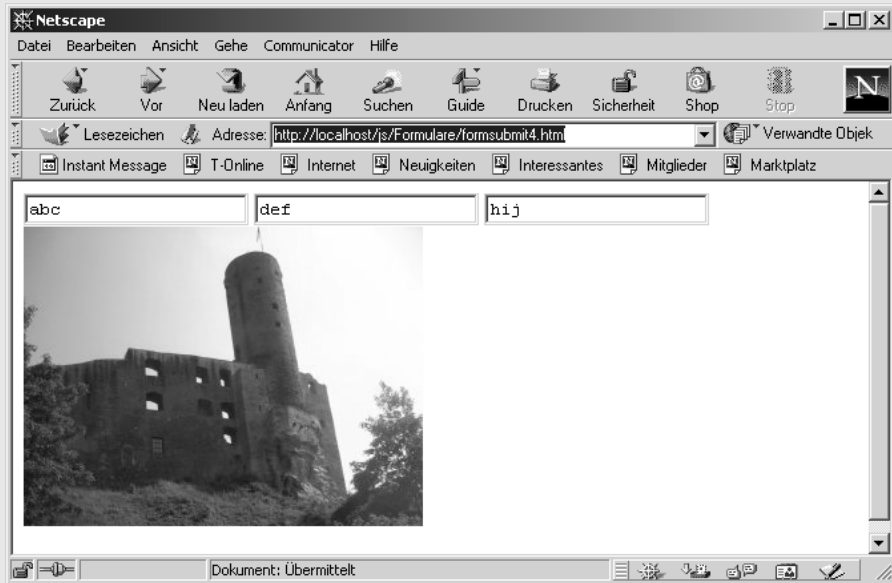


Abbildung 90: Eine Grafik in einem Hyperlink-Container fungiert als sensibles Element des Formulars.

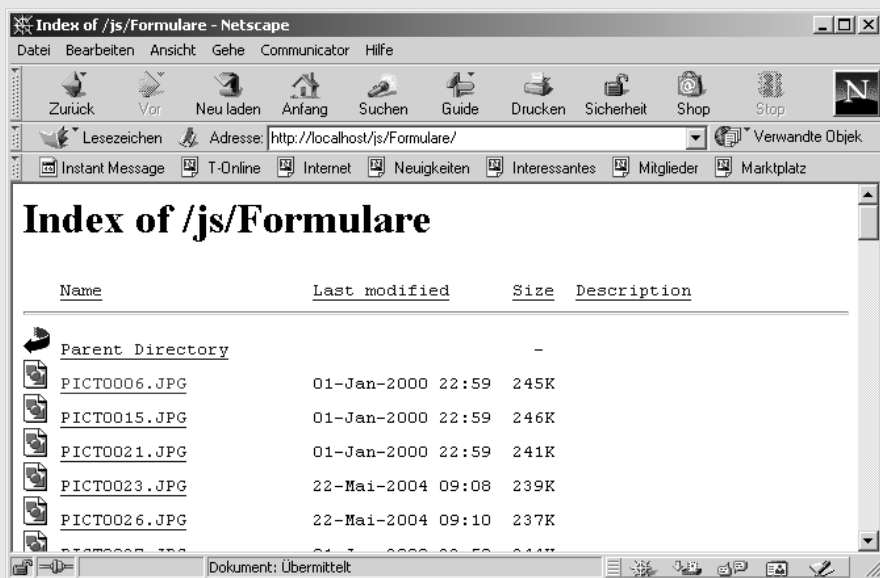


Abbildung 91: Die Daten wurde zwar gesendet, aber anschließend wird der Inhalt des Verzeichnisses angezeigt.

Andere Browser hingegen lösen dagegen ohne `href="#"` die Versendung der Formulardaten überhaupt nicht aus und tun auch sonst nichts – trotz `onClick`. Mit diesem Parameter `href="#"` sind Sie aber auf der sicheren Seite.

Tipp

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 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 HTML-Anweisung das Schlüsselwort `javascript`, gefolgt von einem Doppelpunkt, als Attribut angeben. Die Syntax sieht also so aus:

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

Listing 148: 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 149: Schema einer Inline-Referenz zum Versenden der Formulardaten bei einem Klick auf eine Grafik, die in einen Hyperlink eingeschlossen ist

Beispiel (*forms submit5.html*):

```
01 <html>
02 <body>
03 <form action="test.php" method="get">
04 <input name="a">
05 <input name="b">
06 <input name="b">
07 <br>
08 <a href="javascript:window.document.forms[0].submit()">
09 
10 </a>
11 </form>
12 </body>
13 </html>
```

Listing 150: Beispiel einer Inline-Referenz zum Aufruf bei einem Klick auf eine Grafik in einem Hyperlink-Container

Die Zeilen 3 bis 11 definieren ein einfaches Webformular. In Zeile 8 bis 10 wird eine Grafik in einen Hyperlink-Container eingeschlossen. In Zeile 8 versenden Sie die Daten des Formulars mit einer Inline-Referenz (`<a href="javascript:window.document.forms[0].submit()">`).

Tipp

Sie können sogar rein auf Basis von 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 HTML eine Grafik zum Abschicken von Formulardaten realisieren?« auf Seite 186). Sie können nun 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 sehr einfach, indem Sie `return false` notieren. Andernfalls können Sie – je nach Browser – nicht sicher sein, ob nicht die Versendung via `onSubmit` erfolgt, bevor die Versendung mit JavaScript ausgelöst wurde¹. 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 (*forms submit6.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function sende() {
05     window.status="Versenden der Daten aktiv";
06     window.document.forms[0].submit();
07     alert("Die Daten wurden verschickt.")
08     window.status="";
09 }
10 //-->
11 </script>
12 <noscript></noscript>
13 <body>
14 <form action="" method="get" name="f" onSubmit="return false">
15     <input type="text" name="text">
16     <br>
17     <input type="image" name="submit" src="links.gif" onClick="sende()">
18 </form>
19 </body>
20 </html>
```

Listing 151: Aufruf von `submit()` mit einem `<input>`-Tag

Die Zeilen 14 bis 18 definieren ein einfaches Webformular. In Zeile 17 wird mit `onClick` die Methode `sende()` aufgerufen. Diese ist in Zeile 4 bis 9 definiert. In Zeile 5 wird eine Meldung in der Statuszeile angezeigt. Zeile 6 löst das Versenden aus. In Zeile 7 bekommt der Anwender eine Erfolgsmeldung zu sehen und in Zeile 8 die Statuszeile geleert. Beachten Sie Zeile 14, in der das normale Versenden mit `onSubmit` blockiert wird.

1. Oder ein Browser noch anders reagiert – etwa die Daten zwei Mal versendet.

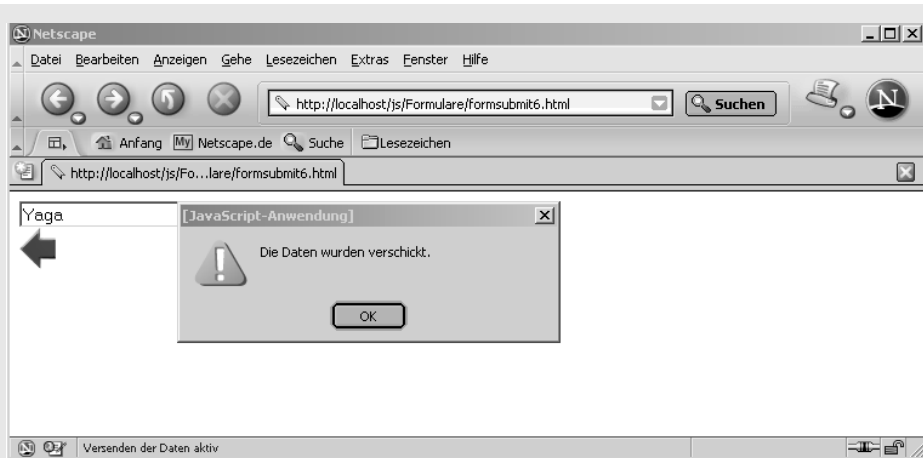


Abbildung 92: Die Daten wurden verschickt, und der Anwender sieht die Erfolgsmeldung.

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.

Tipp

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.

70 Wie kann ich die 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. 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 beschreiben lässt.

Beispiel (*formjsreset.html*):

```
01 <html>
02 <body>
03 <form action="" method="get">
04 <input name="a">
05 <input name="b">
06 <input name="b">
07 <br>
08 <input type="button" value="WischWasch" onClick="this.form.reset()">
09 </form>
10 </body>
11 </html>
```

Listing 152: Die Anwendung der reset()-Methode – Auslösung mit onClick bei einem gewöhnlichen Button

Die Zeilen 3 bis 9 definieren ein einfaches Webformular. In Zeile 8 wird auf einer Schaltfläche mit `onClick` die Methode `reset()` aufgerufen (`<input type="button" value="WischWasch" onClick="this.form.reset()">`).

Hinweis

Beachten Sie in dem Beispiel den Einsatz von `this.form` in Zeile 8 und den Tipp im Rezept »Wie kann ich die Formulardaten ohne einen Submit-Button wegschicken?«.

Achtung

Das Zurücksetzen eines Webformulars setzt nur die Felder in dem Webformular zurück. Damit wird nicht die Seite verlassen.

Hinweis

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 die Formulardaten ohne einen Submit-Button wegschicken?« auf Seite 229).

71 Wie greife ich grundsätzlich auf die Formularelemente zu?

Über JavaScript kann man über *DOM*-Modell auf die einzelnen Elemente einer Webseite zugreifen. Ein Formular in einer Webseite ist über ein Element des Objektfeldes mit Namen `forms[]` oder einen Namen zugänglich. Die einzelnen Formularelemente wie Eingabefelder sind über das dem `forms`-Objektfeld untergeordneten Objektfeld `elements` verfügbar. Dessen Indexnummern werden vom Browser automatisch generiert, wenn er ein zugehöriges Objekt

bei Abarbeitung der 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 HTML spezifiziert wurde.

Hinweis

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 251.*

Zugriff über das Objektfeld

In der DOT-Notation erfolgt der Zugriff auf ein Formularelement schematisch so:

```
window.document.forms[index1].elements[index2]
```

Listing 153: Zugriff auf ein Formularelement in einem Webformular

Beispiel:

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

Listing 154: Zugriff auf das zweite Formularelement im dritten Formular in einer Webseite

Zugriff über einen Namen

Sollte ein Formularelement im HTML-Tag ein `name`-Attribut spezifiziert haben¹⁴, können Sie statt eines Arrayeintrags des Objektfeldes diesen Namen für den Zugriff verwenden.

Beispiel:

```
<form name="adresse">
  <input name="vorname">
  ...// weitere Formularelemente
</form>
```

Listing 155: Der Formularcontainer in HTML

```
window.document.adresse.vorname
```

Listing 156: Der Zugriff aus JavaScript auf das Formularelement

14. Oder es wurde vorher mit JavaScript der Eigenschaft `name` ein Wert zugewiesen.

Beachten Sie, dass beim Namen Groß- und Kleinschreibung relevant ist. Obwohl der Wert im HTML-Container gesetzt wird und Groß- und Kleinschreibung des Namens eines HTML-Elements bei Zugriffen aus HTML (etwa bei einer `target`-Angabe) irrelevant ist, spielt es beim Zugriff aus JavaScript eine Rolle.

Der Zugriff über `this.form` oder `this`

In mehreren 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) 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 die Rezepte »Wie kann ich die Formulardaten ohne einen Submit-Button wegschicken?« auf Seite 229 oder »Wie kann ich die Formulardaten ohne einen Reset-Button zurücksetzen?« auf Seite 234. 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 241. Für eine Anwendung von dem direkten Zugriff auf ein Formularelement mit `this` siehe das Rezept »Wie erfolgt ein verkürzter Zugriff auf ein Formularelement über `this`?« und dort das Beispiel auf Seite 243. Dazu finden Sie in dem Kapitel zahlreiche weitere Beispiele, die diese Techniken verwenden.*

Welche Zugriffsform ist sinnvoll?

Für die Art des Zugriffs gelten die analogen Überlegungen wie für ein Formular selbst. Ein Zugriff über ein Objektfeld ist beim Zugriff über Schleifen effektiver. Ebenso ist es meistens die sinnvollste Möglichkeit zum Zugriff, wenn ein Formularelement über keinen `name`-Parameter im HTML-Tag verfügt. Ein Zugriff über den Namen ist meist besser lesbar und toleranter gegenüber Änderungen des Formularaufbaus selbst. 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 Referenz ist am universellsten einsetzbar, aber geht nicht in jeder Konstellation. Im Grunde bleibt es Ihnen aber in den meisten Situationen selbst überlassen, welche der Versionen Sie vorziehen.

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 157: Zugriff auf das Formular mit dem Namen adresse und dort das zehnte Element

```
window.document.forms[2].nachname
```

Listing 158: Zugriff auf das dritte Formular in der Webseite und dort das Formularfeld mit dem Namen nachname

72 Wie greife ich auf den Wert eines Formularelements zu?

Mit Ausnahme des Containers einer Auswahlliste¹⁵ besitzt jede Objektrepräsentation eines Formularelements die Eigenschaft `value`. Damit können Sie mit Ausnahme eines Auswahllistencontainers den Wert eines beliebigen Formularelementes abfragen oder setzen. Der Datentyp der Eigenschaft ist in jedem Fall ein String.

Beispiel (*formjs8.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function belegeVor() {
05   window.document.forms[0].elements[0].value="Haktar";
06   window.document.a.elements[1].value="Agrajag";
07   window.document.a.c.value="Milliway";
08   window.document.forms[0].d.value="Bistromath";
09 }
10 function macheGross() {
11   var werteVorher="";
12   var werteNachher="";
13   for(i=0;i<window.document.forms[0].length - 1;i++){
14     werteVorher = werteVorher +
15       window.document.forms[0].elements[i].value + "\n";
16     werteNachher = werteNachher +
17       window.document.forms[0].elements[i].value.toUpperCase() + "\n";
```

Listing 159: Setzen und Auslesen der Werte von Formularelementen

15. Das wäre natürlich nicht sinnvoll. Bei einer Auswahlliste interessieren die Einträge in der Liste, und die besitzen die Eigenschaft.

```

18  }
19  alert("Eingegeben haben Sie das:\n" + werteVorher +
20      "\nAbgeschickt wird das:\n" + werteNachher);
21  }
22  //-->
23  </script>
24  <body onLoad="belegeVor()">
25  <form action="" method="get" name="a" onSubmit="macheGross()">
26  <input name="a">
27  <br>
28  <input name="b">
29  <br>
30  <input name="c">
31  <br>
32  <input name="d">
33  <br>
34  <input type="Submit" value="OK">
35  </form>
36  </body>
37  </html>

```

Listing 159: Setzen und Auslesen der Werte von Formularelementen (Forts.)

In den Zeilen 4 bis 9 ist eine Funktion `belegeVor()` definiert. Diese belegt die vier einzelligen Eingabefelder in dem Formular mit Werten vor. Zur Demonstration wird der Zugriff sowohl beim Formular selbst als auch den Formularen unterschiedlich gehandhabt (über den Namen als auch das Objektfeld)¹⁶.

Hinweis

Beachten Sie in dem Beispiel, dass es in der 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 160: Zugriff auf den Wert des Formularfeldes `a` im Formular mit dem Namen `a`

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.

16. 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.

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 209.* Das dort beschriebene Problem wird mit der Verwendung des Eventhandlers umgangen.



Abbildung 93: Die Eingabefelder sind vorbelegt.

Die Funktion `makeGross()` (in den Zeile 10 bis 21) definiert zwei lokale Variablen (in den Zeilen 11 und 12). 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 von einem Eingabefeld 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 eingibt¹⁷. Am Ende der Funktion bekommt der Anwender sowohl seine Originaleingaben als auch die Daten, wie sie verschickt werden, in einem Mitteilungsfenster angezeigt (die Zeilen 19 und 20). Die Funktion `makeGross()` wird vom Eventhandler `onSubmit` ausgelöst. Damit ist gewährleistet, dass diese vor dem Verschieken der Daten abgearbeitet wird.

17. Diese bleiben einfach unverändert.

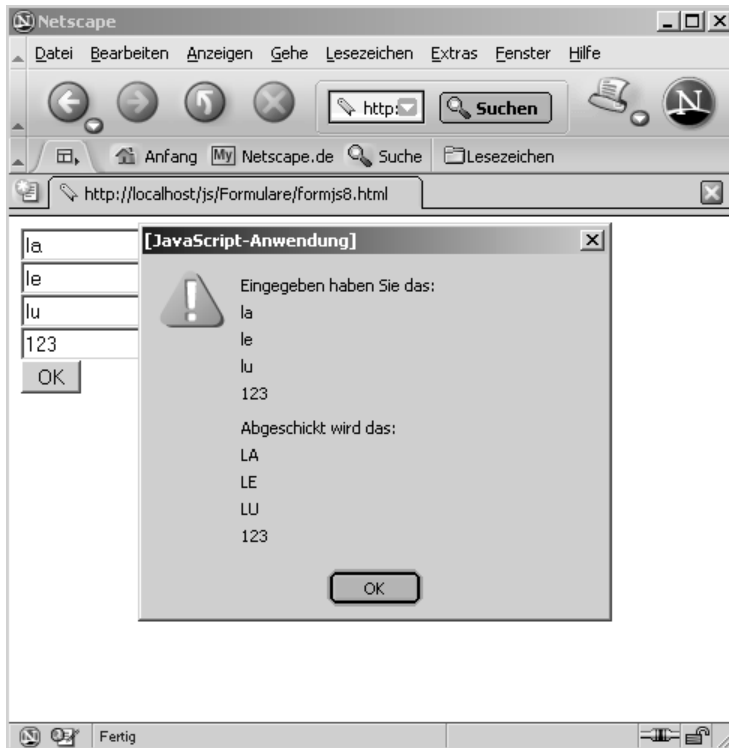


Abbildung 94: Die Originaldaten und die zu verschickenden Daten werden dem Anwender angezeigt.

Verkürzter Zugriff über `this`

Die mit `onSubmit` aufgerufene Funktion `makeGross()` 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 <!--
04 function belegeVor() {
05     window.document.forms[0].elements[0].value="Haktar";
06     window.document.a.elements[1].value="Agrajag";
07     window.document.a.c.value="Milliway";
08     window.document.forms[0].d.value="Bistromath";
09 }
10 function makeGross(frm) {
11     var werteVorher="";
12     var werteNachher="";
13     for(i=0;i<frm.length - 1;i++){
14         werteVorher = werteVorher +
```

Listing 161: Zugriff auf Formularelemente über `this`

```
15     frm.elements[i].value + "\n";
16     werteNachher = werteNachher +
17     frm.elements[i].value.toUpperCase() + "\n";
18 }
19 alert("Eingegeben haben Sie das:\n" + werteVorher +
20       "\nAbgeschickt wird das:\n" + werteNachher);
21 }
22 //-->
23 </script>
24 <body onLoad="belegeVor()">
25 <form action="" method="get" name="a" onSubmit="makeGross(this)">
26 <input name="a">
27 <br>
28 <input name="b">
29 <br>
30 <input name="c">
31 <br>
32 <input name="d">
33 <br>
34 <input type="Submit" value="OK">
35 </form>
36 </body>
37 </html>
```

Listing 161: Zugriff auf Formularelemente über *this* (Forts.)

Die in den Zeilen 10 bis 21 definierte Funktion `makeGross()` 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.

73 Wie greife ich auf den Namen eines Formularelements zu?

Jede Objektrepräsentation eines Webformularelements verfügt über die lesbare als auch schreibbare Eigenschaft `name`. Diese Eigenschaft enthält als Wert den Namen des Formularelements und korrespondiert unmittelbar mit dem `name`-Parameter in dem HTML-Container des Formularelements. Sie können sowohl über das Objektfeld `elements` als auch die Namens Eigenschaft `name` eines Formularelements sowie einen Zugriff via `this` auf die Eigenschaft zugreifen.

Hinweis

Der Zugriff auf `name` über `name` selbst ist etwa sinnvoll, 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 <!--
13   for(i=0;i< window.document.forms[0].length - 1;i++){
14     document.write("
15       Name des Formularelements, wie es in HTML gesetzt wurde: " +
16       window.document.forms[0].elements[i].name + "<br>");
17     window.document.forms[0].elements[i].name = i;
18   }
19   for(i=0;i< window.document.forms[0].length - 1;i++){
20     document.write(
21       "Name des Formularelements, wie es mit JavaScript gesetzt wurde: " +
22       window.document.forms[0].elements[i].name + "<br>");
23   }
24   //-->
25 </script>
26 </body>
27 </html>

```

Listing 162: Zugriff auf die Namen der Formularelemente

In den Zeilen 3 bis 8 wird ein Webformular definiert, und die enthaltenen Formularelemente bekommen über das `name`-Attribut Werte zugewiesen. In den Zeilen 13 bis 18 finden Sie eine `for`-Schleife¹⁸, in der zuerst der per HTML festgelegte Name ausgegeben und dann ein neuer Name zugewiesen wird. Beachten Sie, dass der neue Name über die numerische Variable `i` ermittelt wird. Der Name ist jedoch immer ein String, und die Konvertierung erfolgt automatisch im Hintergrund.

74 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.

18. Die Anzahl der Formularelemente wird über `length` berücksichtigt. Der Wert `-1` wird deshalb genommen, um den Button auszuschließen.

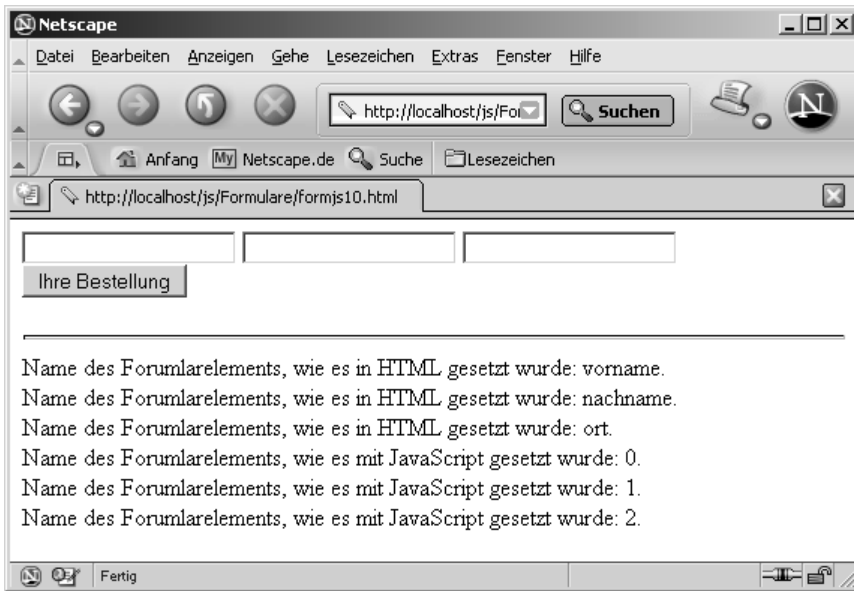


Abbildung 95: Lese- und Schreibzugriff auf die Namen der Formularelemente

Beispiel (*formjs10a.html*):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04 function statuszeile (nm) {
05     window.status =
06     "Name des Formularelements, wie es in HTML gesetzt wurde: " +
07     nm.name + ".";
08 }
09 //-->
10 </script>
11 <body>
12 <form action="" method="get">
13 <input name="vorname" onClick="statuszeile(this)">
14 <input name="nachname" onClick="statuszeile(this)">
15 <input name="ort" onClick="statuszeile(this)">
16 <br>
17 <input type="Submit" value="Ihre Bestellung">
18 </form>
19 </body>
20 </html>

```

Listing 163: Zugriff auf ein Formularfeld über *this*

In den Zeilen 12 bis 18 wird ein Webformular definiert. Die Eingabefelder in den Zeilen 13 bis 15 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 4 bis 8) wird über den Parameter (der als Objekt das aktuelle Eingabefeld enthält) der Name des jeweiligen Objektes ausgelesen (Zeile 7 – `nm.name`) und zusammen mit einem Text in der Statuszeile des Browsers angezeigt (das macht die Zuweisung zu der `status`-Eigenschaft von dem `window`-Objekt). Da der Eventhandler `onClick` verwendet wird, bekommt ein Besucher immer einen neuen Text angezeigt, wenn er in ein anderes Eingabefeld klickt.

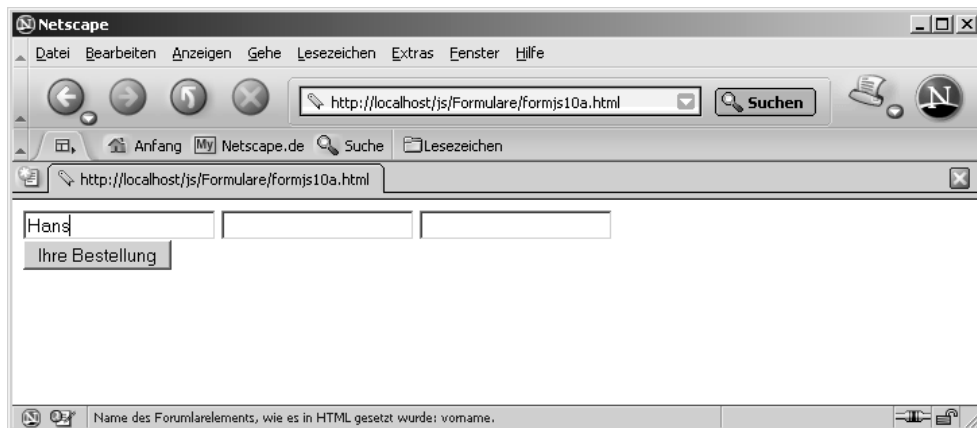


Abbildung 96: Der Anwender klickt in das erste Eingabefeld und sieht in der Statuszeile den Namen des Feldes.

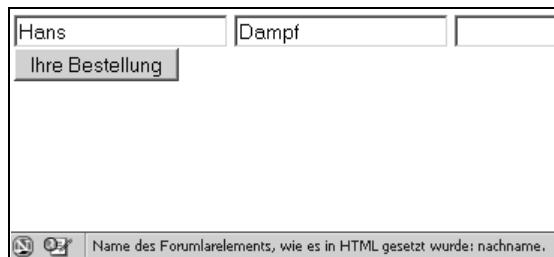


Abbildung 97: Klick in das zweite Eingabefeld – in der Statuszeile wird nachname angezeigt.

75 Wie greife ich aus einem Formularelement auf das umgehende Formular zu?

Jede Objektrepräsentation eines Formularelements in einem Webformular enthält die lesbare Eigenschaft `form`. Diese heißt zwar gleich wie das Objekt `form` und korrespondiert damit unmittelbar, aber die Eigenschaft ist in der Hierarchie dem Formularelement untergeordnet. Das Objekt `form` hingegen ist dem Formularelement übergeordnet. Die Eigenschaft `form` eines Formularelementes ist aber im Grunde nichts 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 Formularele-

ment relativ in der Hierarchieebene eine Stufe höher gehen wollen und ein Zugriff über die `form`-Eigenschaft eines Formularelementes Sinn macht. Betrachten Sie das nachfolgende Listing.

Beispiel (*formjs14.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function belegeVor(frm) {
05   frm.form.elements[0].value="Haktar";
06   frm.form.elements[1].value="Agrajag";
07   frm.form.elements[2].value="Milliway";
08   frm.form.elements[3].value="Bistromath";
09 }
10 //-->
11 </script>
12 <body>
13 <form action="" method="get" name="f">
14 <input name="a">
15 <br>
16 <input name="b">
17 <br>
18 <input name="c">
19 <br>
20 <input name="d">
21 <br>
22 <input type="button" onClick="belegeVor(this)" value="Formular vorbelegen">
23 <input type="Submit" value="OK">
24 </form>
25 </body>
26 </html>
```

Listing 164: Einsatz der `form`-Eigenschaft eines Formularelementes

In dem Beispiel wird ein Webformular mit vier Eingabefeldern per HTML definiert (die Zeilen 13 bis 24). 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 22 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 4 bis 9 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.



Abbildung 98: Das Formular wird mit einem Klick auf die entsprechende Schaltfläche vorbelegt.

76 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 Formularelementes.

Hinweis

Die Eigenschaft korrespondiert mit dem gleichnamigen `type`-Parameter in dem `<input>`-Tag, sofern es sich bei einem 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 zu dem Wert, den Sie in 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 aus Unwissenheit ob der Ignoranz der Browser dennoch verwenden (siehe dazu die Anmerkungen zu den HTML-Grundlagen von Formularen):

Formularelement	Wert für <code>type</code>
Auswahlliste	SELECT-ONE
Checkbox	CHECKBOX
Dateieingabefeld	FILE
Datumsfeld	DATE
Einfache Schaltfläche	BUTTON

Tabelle 2: Ausprägungen des `type`-Attributs

Formularelement	Wert für type
Einzeiliges Textfeld	TEXT
Ganzzahlfeld	INT
Gleitzahlfeld	FLOAT
Mehrzeiliges Textfeld	TEXTAREA
Passwortfeld	PASSWORD
Radiobutton	RADIO
Reset-Button	RESET
Submit-Button	SUBMIT
URL-Feld	URL

Tabelle 2: Ausprägungen des type-Attributs (Forts.)

Beispiel (*formjs11.html*):

```

01 <html>
02 <body>
03 <form action="" method="get">
04 Userid <input>
05 <br>
06 Passwort <input type="Password">
07 <br>
08 Kommentar <textarea name="" cols="40" rows="5"></textarea>
09 <br>
10 Ja <input type="Radio" name="a" value="v">
11 Nein <input type="Radio" name="a" value="w">
12 <br>
13 Speichern <input type="Checkbox" name="" value="v">
14 <br>
15 <select name="" size="">
16 <option>Eins</option>
17 <option>Zwei</option>
18 <option>Drei</option>
19 </select>
20 <br>
21 <input type="Submit" value="Ihre Bestellung">
22 </form>
23 <hr>
24 <script language="JavaScript">
25 <!--
26   for(i=0;i< window.document.forms[0].length;i++){
27     document.write("Typ des Formularelements " + (i + 1) + ": " +
28       window.document.forms[0].elements[i].type + "<br>");
29   }
30 //-->
31 </script>

```

Listing 165: Zugriff auf den Typ von Formularelementen

```
32 </body>
33 </html>
```

Listing 165: Zugriff auf den Typ von Formularelementen (Forts.)

Das Formular in dem Beispiel (Zeile 3 bis 22) enthält bis auf die versteckten Formularfelder und einfache Buttons sämtliche möglichen Formularelemente. In der Schleife (Zeile 26 bis 29) wird der jeweilige Typ ausgegeben. Beachten Sie, dass die einzelnen Einträge der Auswahlliste dabei nicht auftauchen (siehe dazu das Rezept »Wie greife ich auf die Elemente einer Auswahlliste in einem Listenfeld zu?« auf Seite 251).

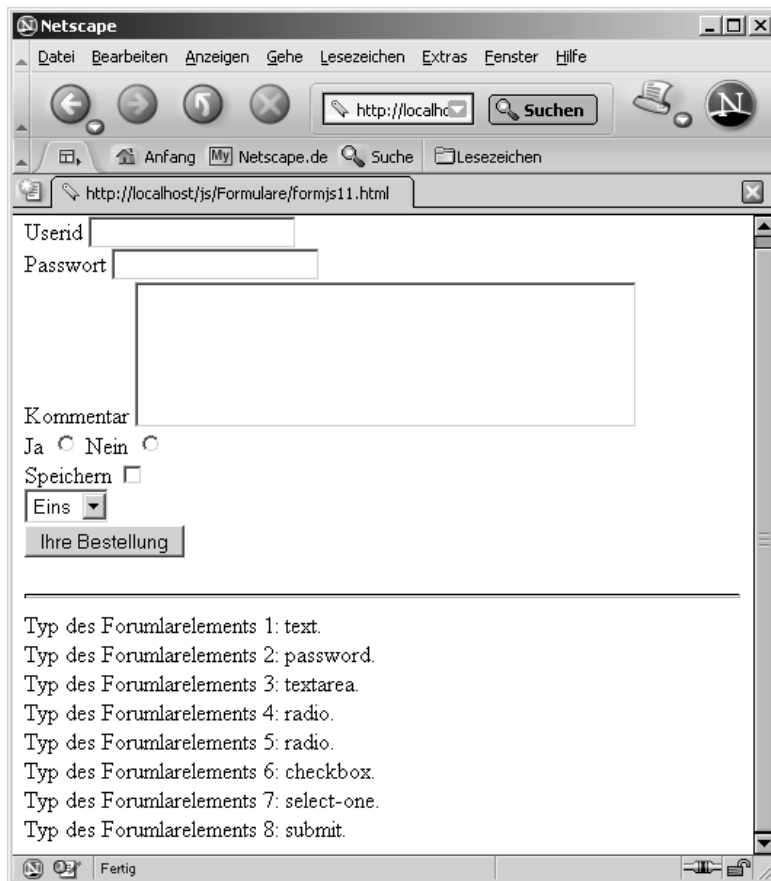


Abbildung 99: Die verschiedenen Typen der Formularelemente lassen sich per JavaScript abfragen.

77 Wie kann ich auf den Selektionszustand von einem Optionsfeld oder einem Kontrollkästchen zugreifen?

Sowohl die Objektrepräsentation von Optionsfeldern als auch Kontrollkästchen in einer Webseite verfügen über die Boole'sche Eigenschaft `checked`. Diese korrespondiert mit dem entsprechenden HTML-Attribut und ist aus JavaScript heraus sowohl zu lesen als auch zu schreiben. Hat diese Eigenschaft den Wert `true` beziehungsweise `1`, ist das Formularelement selektiert. Hat sie aber den Wert `false` beziehungsweise `0`, ist das Formularelement deselektiert.

Beispiel (*formjs12.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04   function radio() {
05     if(window.document.forms[0].elements[0].checked) {
06       alert("Klasse")
07     }
08     else {
09       alert("Mist")
10     }
11   }
12 }
13 //-->
14 </script>
15 <body>
16 <form action="" method="get">
17   Ja <input type="Radio" name="a" value="v">
18   Nein <input type="Radio" name="a" value="w">
19   <br>
20   <input type="button" value="OK" onClick="radio()">
21 </form>
22 </body>
23 </html>
```

Listing 166: Auslesen des Selektionszustands von dem ersten Radiobutton

In dem Beispiel wird ein Webformular (Zeilen 16 bis 21) mit zwei Optionsfeldern per HTML definiert (die Zeilen 17 und 18). Die Zeilen 4 bis 12 definieren die Funktion `radio()`, die über den Eventhandler `onClick` bei dem einfachen Button (Zeile 20) aufgerufen wird. In Zeile 5 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 deselektiert, wenn der erste Radiobutton selektiert ist, und umgekehrt.



Abbildung 100: Der erste Radiobutton ist selektiert, als auf den OK-Button geklickt wurde.



Abbildung 101: Der erste Radiobutton ist beim Klick auf OK deselektiert.

Das Setzen der Eigenschaft aus JavaScript heraus ist einfach. Sie weisen nur den passenden Boole'schen Wert zu, etwa so:

```
window.document.forms[0].elements[3].checked=true;
```

Listing 167: Setzen der Eigenschaft checked

78 Wie greife ich auf die Elemente einer Auswahlliste in einem Listefeld zu?

Der Zugriff auf die Elemente einer Auswahlliste in einem Listefeld aus JavaScript heraus verhält sich nicht wie bei den übrigen Formularelementen und muss gesondert behandelt werden. Der Grund ist die Art der Notation in HTML. Die einzelnen Elemente in der Auswahlliste werden in `<option>`-Container in einem umgebenden `<select>`-Container eingeschlossen (siehe das Rezept »Wie kann ich eine Auswahlliste mit einzeiligem Listefeld

erstellen?» auf Seite 199). Mit dem Zugriff auf das `elements`-Objektfeld referenzieren Sie diesen `<select>`-Container, jedoch noch nicht die einzelnen Einträge in den `<option>`-Containern.

Jede Objektrepräsentation von einem Formularelement, die einen `<select>`-Container verkörpert, verfügt als Eigenschaft über das Objektfeld `options`. Dieses stellt ein Datenfeld mit Objektrepräsentationen von den `<option>`-Containern dar und korrespondiert unmittelbar mit dem `<option>`-Container in der Auswahlliste. Sie haben darüber also vollständigen Zugriff auf die Einträge in einer Auswahlliste innerhalb eines Formulars. Die Einträge in einer Auswahlliste können wie jedes andere Formularelement mit einer Indexnummer oder dem Namen angesprochen werden.

Betrachten Sie das folgende Webformular als Beispiel für eine HTML-Struktur einer Auswahlliste (*form6.html*):

```
01 <html>
02 <body>
03 <form action="" method="get" name="eis">
04 <select name="eissorte">
05   <option name="a">Erdbeere</option>
06   <option name="b">Vanille</option>
07   <option name="c">Schoko</option>
08   <option name="d">Zitrone</option>
09 </select>
10 <hr>
11 <input type="Submit" value="Ihre Bestellung bitte">
12 </form>
13 </body>
14 </html>
```

Listing 168: Ein Formular mit Auswahlliste

In dem Beispiel wird ein Webformular (Zeilen 3 bis 11) mit einer Auswahlliste definiert (die Zeilen 4 und 9). Die Auswahlliste enthält vier Einträge.

The image shows a web browser window containing a form. At the top, there is a dropdown menu (select box) with the text 'Erdbeere' and a small downward arrow. Below the dropdown is a horizontal line. Underneath the line is a button with the text 'Ihre Bestellung bitte'.

Abbildung 102: Ein Formular mit einem einzeiligen Listenfeld

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 169: Zugriff auf einen Eintrag in einer Auswahlliste

Beispiel:

```
window.document.forms[0].elements[0].options[1]
```

Listing 170: Zugriff auf den zweiten Eintrag im ersten Formularelement im ersten Formular in einer Webseite (für die Beispielwebseite ist das der Wert Vanille)

Zugriff über einen Namen

Sollte ein Eintrag in einer Auswahlliste im HTML-Tag ein `name`-Attribut spezifiziert haben, können Sie statt einem Datenfeldelement des Objektfeldes diesen Namen für den Zugriff verwenden.

Beispiel:

```
window.document.eis.eissorte.c
```

Listing 171: Der Zugriff aus JavaScript auf den Eintrag in der Auswahlliste (für die Beispielwebseite ist das der Wert Schoko)

Achtung

Beachten Sie, dass beim Namen Groß- und Kleinschreibung relevant ist. Obwohl der Wert im HTML-Container gesetzt wird und Groß- und Kleinschreibung des Namens eines HTML-Elements bei Zugriffen aus HTML (etwa bei einer `target`-Angabe) irrelevant ist, spielt es beim Zugriff aus JavaScript eine Rolle.

Zugriff über `this.form` und `this`

In mehreren 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) 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 die Formulardaten ohne einen Submit-Button wegschicken?« auf Seite 229 oder »Wie kann ich die Formulardaten ohne einen Reset-Button zurücksetzen?« auf Seite 234.*

Hinweis

Da es sich bei jedem Eintrag in dem `options`-Objektfeld selbst um ein Objekt handelt, stellt dieses auch spezifische eigene Eigenschaften bereit, die auch erst den konkreten Nutzen darstellen (siehe die nachfolgenden Rezepte).

79 Wie greife ich auf den Wert eines Eintrags in einer Auswahlliste zu?

Mit Ausnahme des Containers einer Auswahlliste¹⁹ besitzt jede Objektrepräsentation eines Formularelements die Eigenschaft `value`, die mit dem gleichnamigen 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 HTML noch per JavaScript), wird der Text aus dem `<option>`-Container an den Server übermittelt. Und dieser steht über die Eigenschaft `text` bereit (siehe das Rezept »Wie greife ich auf die Beschriftung von einem Eintrag in einer Auswahlliste zu?« auf Seite 256). Aber oft ist es sinnvoll, die `value`-Eigenschaft abzufragen oder auch zu setzen.

Beispiel (*formjs15.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function liste(frm) {
05   var text = "Die Werte in der Auswahlliste:\n";
06   for(i=0; i < frm.form.eissorte.length; i++){
07     text = text + frm.form.eissorte.options[i].value + "\n";
08   }
09   alert(text);
10 }
11 //-->
12 </script>
13 <body>
14 <form action="" method="get" name="eis">
15 <select name="eissorte">
16   <option name="a" value="ErdBaer">Erdbeere</option>
17   <option name="b" value="MilliVanilli">Vanille</option>
18   <option name="c" value="DickMach">Schoko</option>
19   <option name="d" value="Sauer">Zitrone</option>
20 </select>
21 <hr>
22 <input type="button" value="Die Werte der Liste" onClick="liste(this)">
23 <input type="Submit" value="Ihre Bestellung bitte">
24 </form>
25 </body>
26 </html>
```

Listing 172: Zugriff auf die Werte eines Eintrags in der Auswahlliste

19. Das wäre natürlich nicht sinnvoll. Bei einer Auswahlliste interessieren die Einträge in der Liste, und die besitzen die Eigenschaft.

In den Zeilen 14 bis 24 wird ein Webformular mit einer Auswahlliste sowie zwei Schaltflächen definiert. In der Auswahlliste sind vier Listeneinträge zu finden (die Zeilen 16 bis 19), bei denen der `value`-Wert im HTML-Tag spezifiziert ist. In Zeile 22 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 6 und 7). In Zeile 6 nutzen wir mit `frm.form.eissorte.length` aus, dass das Objekt für die Auswahlliste eine Eigenschaft hat, die die Anzahl der enthaltenen Einträge enthält, und in Zeile 7 greifen wir über das Objektfeld `options` auf die einzelnen Einträge zu und fragen die gesetzten Werte ab.

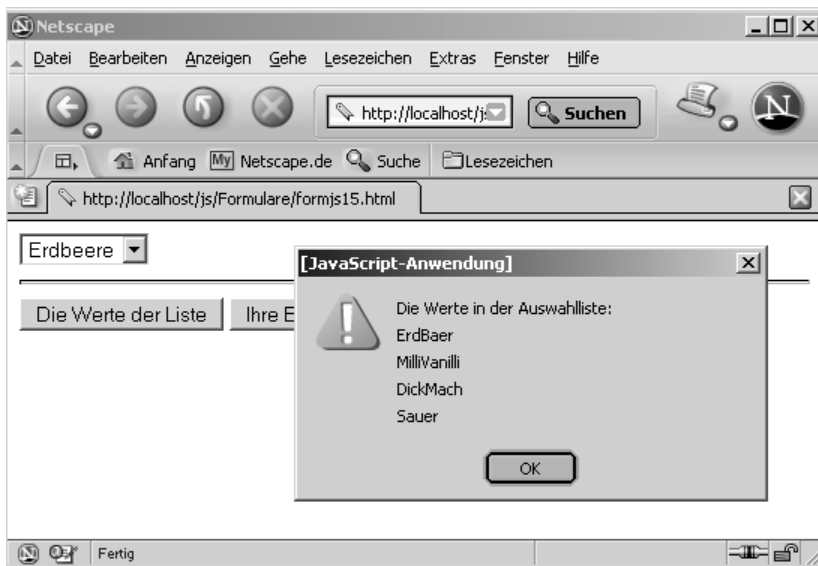


Abbildung 103: Die Werte, wie sie in den `<option>`-Tags über den Parameter `value` angegeben sind – beachten Sie die Unterschiede zu der Beschriftung.

Hinweis

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 173: Setzen der Eigenschaft `value` für einen Eintrag in einer Auswahlliste

80 Wie greife ich auf die Beschriftung eines Eintrags 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` bereit.

Beispiel (*form16.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function liste(frm) {
05   var text = "Die Texte der Einträge in der Auswahlliste:\n";
06   for(i=0; i < frm.form.eissorte.length; i++){
07     text = text + frm.form.eissorte.options[i].text + "\n";
08   }
09   alert(text);
10 }
11 //-->
12 </script>
13 <body>
14 <form action="" method="get" name="eis">
15 <select name="eissorte">
16   <option name="a" value="ErdBaer">Erdbeere</option>
17   <option name="b" value="MilliVanilli">Vanille</option>
18   <option name="c" value="DickMach">Schoko</option>
19   <option name="d" value="Sauer">Zitrone</option>
20 </select>
21 <br>
22 <input type="button" value="Die Einträge der Liste" onClick="liste(this)">
23 <input type="Submit" value="Ihre Bestellung bitte">
24 </form>
25 </body>
26 </html>
```

Listing 174: Zugriff auf die sichtbaren Texte der Auswahlliste

In den Zeilen 14 bis 24 wird ein Webformular mit einer Auswahlliste sowie zwei Schaltflächen definiert. In der Auswahlliste sind vier Listeneinträge zu finden (die Zeilen 16 bis 19), bei denen der `value`-Wert im HTML-Tag spezifiziert ist. In Zeile 22 wird mit dem Eventhandler `onClick` die Funktion `liste()` aufgerufen und mit `this` eine Referenz auf das aktuelle Formularelement (der 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 die Beschriftung der vier Einträge in der Auswahlliste abgefragt werden. Mit dem Übergabewert `frm` haben wir jedoch ein Formularelement zur Verfügung, und dessen Eigenschaft `form` ist eine Referenz auf das umgebende Formular. Also können wir mit `frm.form` auf dieses Formular zugreifen. Mit `frm.form.eissorte` haben wir Zugriff auf die Auswahlliste (in den Zeilen 6 und 7). In Zeile 6 nutzen wir mit `frm.form.eis-`

`sorte.length` aus, dass das Objekt für die Auswahlliste eine Eigenschaft hat, die die Anzahl der enthaltenen Einträge enthält, und in Zeile 7 greifen wir über das Objektfeld `options` auf die einzelnen Einträge zu und fragen die gesetzten Textwerte ab.



Abbildung 104: Die Werte, wie sie in den `<option>`-Containern notiert sind – diese Werte sind nicht (zwingend) identisch zu den Werten, die über den Parameter `value` angegeben sind.

Selbstverständlich können Sie auch die Texte über die Eigenschaft `text` ändern. Beachten Sie das nachfolgende Beispiel (`form17.html`):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04 function liste(frm) {
05   frm.form.eissorte.options[0].text=frm.form.neu.value;
06 }
07 //-->
08 </script>
09 <body>
10 <form action="" method="get" name="eis">
11 <input type="Text" name="neu">
12 <br>
13 <select name="eissorte">
14   <option name="a"></option>
15 </select>
16 <hr>
17 <input type="button" value="Die Einträge der Liste" onClick="liste(this)">
18 </form>

```

Listing 175: Setzen der Texte in einer Auswahlliste

```
19 </body>
20 </html>
```

Listing 175: Setzen der Texte in einer Auswahlliste (Forts.)

In den Zeilen 10 bis 18 wird ein Webformular mit einer Auswahlliste, einem Eingabefeld sowie einer Schaltfläche definiert. In der Funktion `liste()` wird in Zeile 5 der `text`-Eigenschaft des ersten Eintrags in der Auswahlliste über den Eingabewert des Anwenders in dem Eingabefeld (also der Eigenschaft `value`) zugewiesen.



Abbildung 105: Die Auswahlliste ist noch leer.



Abbildung 106: Nach der Übernahme der Benutzereingabe ist der Eintrag in der Auswahlliste beschriftet.

Sie können mit der Zuweisung der `text`-Eigenschaft via JavaScript nur bereits per HTML angelegte Einträge neu beschriften und keine neuen Einträge erzeugen. Wenn Sie das versuchen, erhalten Sie eine Fehlermeldung der folgenden Art (die Zuweisung müsste ein Objekt erzeugen, um zu funktionieren):



Abbildung 107: Der Versuch einer Textzuweisung zu einem nicht vorhandenen Eintrag in einer Auswahlliste

In einigen Browsern ist es möglich, zur Laufzeit einer Auswahlliste mit JavaScript weitere Einträge hinzuzufügen. Dazu wird ein Konstruktor `Option` verwendet. Das sieht schematisch so aus:

```
var neueOpt = new Option([optionText, optionWert, defaultSelected, selected]);
```

Listing 176: Erzeugen eines neuen Eintrags in einer Auswahlliste

Da diese Technik jedoch von zahlreichen Browsern nicht unterstützt wird, soll nicht weiter darauf eingegangen werden.

81 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.

Beispiel (*formjs18.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function liste(frm) {
05   alert("Anzahl der Einträge in der Auswahlliste: " +
06     window.document.eis.eissorte.length);
07 }
08 //-->
09 </script>
10 <body onLoad="liste(this)">
11 <form action="" method="get" name="eis">
12 <select name="eissorte">
13   <option>Erdbeere</option>
14   <option>Vanille</option>
15   <option>Schoko</option>
16   <option>Zitrone</option>
17 </select>
18 </form>
19 </body>
20 </html>
```

Listing 177: Zugriff auf die Anzahl der Einträge

In den Zeilen 11 bis 18 wird ein Webformular mit einer Auswahlliste definiert (Zeile 12 bis 17). Darin sind vier Einträge angegeben (Zeile 13 bis 16). Beim Laden der Webseite wird mit dem Eventhandler `onLoad` die Funktion `liste()` aufgerufen (in Zeile 10 über `<body onLoad="liste(this)">`). Diese Funktion ist von Zeile 4 bis 7 definiert (`function liste(frm) { alert("Anzahl de Einträge in der Auswahlliste: " + window.document.eis.eissorte.length); }`). In Zeile 6 wird auf die Anzahl der Einträge in der Auswahlliste zugegriffen.



Abbildung 108: Zugriff auf die Anzahl der Einträge in einer Auswahlliste

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. Mit anderen Worten: Die Zeile 6 in dem Beispiel könnte auch darüber formuliert werden:

```
window.document.eis.eissorte.options.length
```

Listing 178: Die Anzahl der Einträge in der Auswahlliste als Abfrage der Größe des options-Array

82 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 (*forms19.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function welcherIndex(op) {
05   window.status = "Ausgewählter Eintrag in der Auswahlliste: " +
06     op.selectedIndex;
07 }
08 //-->
09 </script>
10 <body>
11 <form action="" method="get" name="eis">
12 <select name="eissorte" onClick="welcherIndex(this)">
13   <option>Erdbeere</option>
14   <option>Vanille</option>
15   <option>Schoko</option>
16   <option>Zitrone</option>
17 </select>
18 </form>
19 </body>
20 </html>
```

Listing 179: Bestimmen des ausgewählten Eintrags in der Auswahlliste

In den Zeilen 11 bis 18 wird ein Webformular mit einer Auswahlliste definiert. In der Auswahlliste sind vier Listeneinträge zu finden (die Zeilen 13 bis 16). Im Beginn-Tag des `<select>`-Containers in Zeile 12 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 Aus-

wahlliste zur Verfügung, und wir können darüber auf `selectedIndex` zugreifen. Der Wert wird in den Zeilen 5 und 6 in der Statuszeile des Browsers ausgegeben.

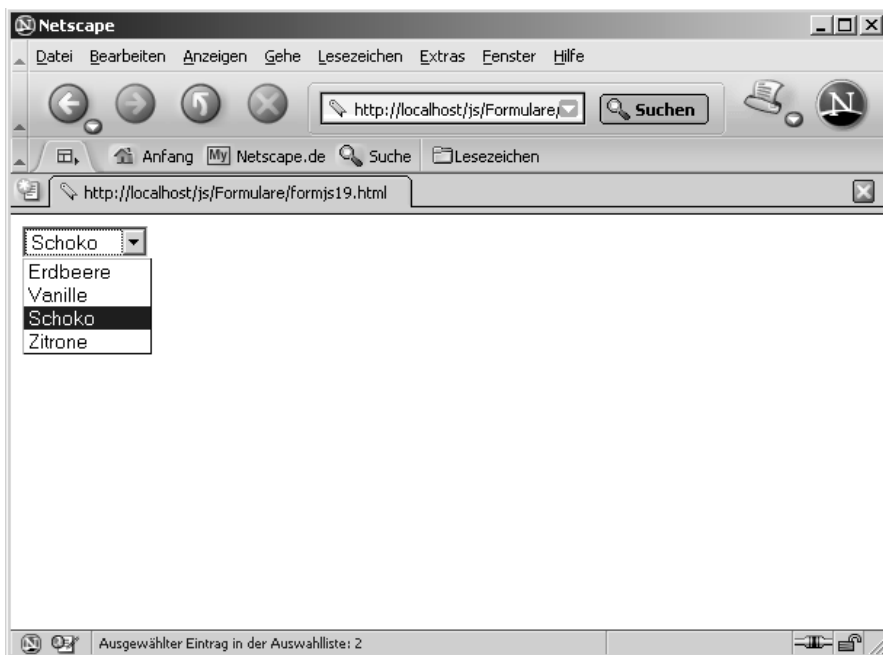


Abbildung 109: In der Statuszeile wird der selektierte Index (beginnend bei 0) angezeigt.

Das Setzen eines Eintrags als selektiert ist auch nicht kompliziert. Sie müssen nur den passenden Wert zuweisen.

Beispiel (*form19jsa.html*):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04 function welcherIndex(i) {
05     window.document.eis.eissorte.selectedIndex=i;
06 }
07 //-->
08 </script>
09 <body>
10 <form action="" method="get" name="eis">
11 <select name="eissorte" size=4>
12     <option>Erdbeere</option>
13     <option>Vanille</option>
14     <option>Schoko</option>
15     <option>Zitrone</option>

```

Listing 180: Selektieren eines Eintrags in einer Auswahlliste mit JavaScript

```

16 </select>
17 <hr>
18 <input type="radio" name="klicker" value="0" onClick="welcherIndex(0)">
19 <input type="radio" name="klicker" value="1" onClick="welcherIndex(1)">
20 <input type="radio" name="klicker" value="2" onClick="welcherIndex(2)">
21 <input type="radio" name="klicker" value="3" onClick="welcherIndex(3)">
22 </form>
23 </body>
24 </html>

```

Listing 180: Selektieren eines Eintrags in einer Auswahlliste mit JavaScript (Forts.)

In den Zeilen 10 bis 22 wird ein Webformular mit einer mehrzeiligen Auswahlliste sowie vier Radiobuttons definiert. In der Auswahlliste sind vier Listeneinträge zu finden (die Zeilen 12 bis 15). In jedem Radiobutton (in den Zeilen 18 bis 22) wird mit dem Eventhandler `onClick` die Funktion `welcherIndex()` aufgerufen. Dabei wird ein numerischer Wert übergeben. In der Funktion `welcherIndex()` (Zeile 4 bis 6) setzen wir diesen Wert als selektierten Index der Auswahlliste.



Abbildung 110: 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 181: Selektion eines Eintrags in der Auswahlliste über die Eigenschaft `selected`

In diesem Fall verwendet man für ein Element des `options`-Objektfeldes die Eigenschaft `selected`. Diese Boole'sche Eigenschaft kann auf `true` beziehungsweise `1` oder `false` beziehungsweise `0` gesetzt werden. Wenn ein Eintrag als selektiert gesetzt wird, deselektiert das alle anderen Einträge.

83 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.

Achtung

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 266 und dort das Beispiel*. Wenn ein Element den Fokus besitzt, wird das von den meisten Browsern (als 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.

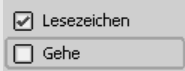


Abbildung 111: Das Kontrollkästchen `Lesezeichen` ist ausgewählt und besitzt den Fokus nicht, während das Kontrollkästchen `Gehe` nicht ausgewählt ist, aber den Fokus besitzt.

Bei den meisten grafischen Oberflächen verschieben Sie den Fokus mit dem  oder . Dabei selektieren Sie die jeweiligen Elemente nicht. Das machen Sie mit dem Anklicken durch die Maus oder oft auch mit der .

Beispiel (*formjs20.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function welcherIndex(i) {
05   i.blur();
06 }
07 //-->
08 </script>
09 <body>
```

Listing 182: Der Fokus wird per `blur()` entfernt.

```
10 <form action="" method="get" name="eis">
11 A <input type="radio" name="klicker" value="0"
12   onMouseOut="welcherIndex(this)">
13 B <input type="radio" name="klicker" value="1"
14   onMouseOut="welcherIndex(this)">
15 C <input type="radio" name="klicker" value="2"
16   onMouseOut="welcherIndex(this)">
17 D <input type="radio" name="klicker" value="3"
18   onMouseOut="welcherIndex(this)">
19 </form>
20 </body>
21 </html>
```

Listing 182: Der Fokus wird per `blur()` entfernt. (Forts.)

In dem Beispiel gibt es ein kleines Formular (Zeile 10 bis 19) mit vier Optionsfeldern. Bei jedem Optionsfeld ist der Eventhandler `onMouseOut` notiert. Dieser wird ausgelöst, wenn der Mauszeiger den Bereich eines Elements (in diesem Fall das jeweilige Optionsfeld) verlässt. Beim Verlassen wird die Funktion `welcherIndex()` ausgelöst. Dabei wird als Übergabewert mit `this` eine Referenz auf das jeweilige Optionsfeld übergeben. In der Funktion wird über diese Referenz auf das jeweilige Optionsfeld die Methode `blur()` ausgelöst (Zeile 5).

Wenn ein Anwender ein Optionsfeld anklickt, wird dieses selektiert, und alle anderen Optionsfelder der Gruppe werden automatisch deselektiert. Beim Klick auf ein Optionsfeld erhält dieses zudem den Fokus. Das ist die »natürliche« Verhaltensweise eines Optionsfelds und benötigt kein JavaScript oder sonst eine Ergänzungstechnologie.



Abbildung 112: Der Anwender hat den Radiobutton angeklickt, und der Mauszeiger ist im Bereich des Radiobuttons – der Radiobutton ist selektiert und besitzt den Fokus.

Wenn Sie jedoch den Mauszeiger aus dem Anzeigebereich des Optionsfelds bewegen, bleibt es zwar selektiert, aber ihm wird automatisch der Fokus entzogen. Das ist keine natürliche Verhaltensweise eines Optionsfelds²⁰, sondern wird explizit von der Methode `blur()` ausgelöst. Normalerweise müssen Sie mit der Maus ein anderes Element des aktiven Fensters anklicken, um den Fokus zu verschieben.

20. Das können Sie ganz leicht testen – kommentieren Sie einfach Zeile 5 in dem Beispiel aus.



Abbildung 113: Der Mauszeiger verlässt den Bereich des Optionsfelds – das Optionsfeld ist immer noch selektiert, hat aber den Fokus verloren.

84 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.

Achtung

Das Entfernen des Fokus ist nicht identisch mit einer Deselektion. Genauso wenig bedeutet der Erhalt des Fokus eine Selektion. *Beachten Sie die Warnung beim Rezept »Wie kann ich mit JavaScript den Fokus von einem Formularelement nehmen?« auf Seite 264.*

Beispiel (*formjs21.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function welcherIndex() {
05     var i = Math.round(Math.random() * 4 ) % 4;
06     window.document.eis.elements[i].focus();
07 }
08 //-->
09 </script>
10 <body onLoad="welcherIndex()">
11 <form action="" method="get" name="eis">
12 A <input type="radio" name="klicker" value="0">
13 B <input type="radio" name="klicker" value="1">
14 C <input type="radio" name="klicker" value="2">
15 D <input type="radio" name="klicker" value="3">
16 </form>
17 </body>
18 </html>
```

Listing 183: Der Fokus wird per `focus()` ersetzt.

In dem Beispiel finden Sie ein kleines Formular (Zeile 11 bis 16) mit vier Optionsfeldern vor. Beim `<body>`-Tag ist der Eventhandler `onLoad` notiert. Dieser wird ausgelöst, wenn die Webseite geladen wird, und führt die Funktion `welcherIndex()` aus. In der Funktion wird mit einem Zufallsmechanismus eine ganze Zahl zwischen 0 und 3 berechnet (Zeile 5). Diese wird als Index für die Zuordnung des Fokus zu einem Optionsfeld mit der `focus()`-Methode verwendet (Zeile 6).

Wenn ein Anwender die Webseite lädt, erhält einer der Optionsfelder der Gruppe zufällig den Fokus. Er wird aber nicht selektiert!

Hinweis

Genau genommen ist beim Laden der Webseite keines der Optionsfelder selektiert, denn in dem Formular wird keines der Optionsfelder durch eine entsprechende HTML-Anweisung vorselektiert. Solange ein Anwender keines der Optionsfelder explizit selektiert (durch Anklicken oder Drücken der Leertaste, wenn ein Optionsfeld den Fokus hat), ist keines der Optionsfelder selektiert. Sie können aber in HTML ein Optionsfeld bereits vorselektieren. Das macht der Parameter `checked`, etwa so:

```
<input type="radio" name="klicker" value="0" checked>
```

Listing 184: Ein Radiobutton wird vorselektiert.



Abbildung 114: Der zweite Radiobutton hat den Fokus, ist aber nicht selektiert.

85 Wie kann ich ein Formularelement selektieren bzw. 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 von einem Formularelement unterscheiden.

Selektion von Optionsfeldern und Kontrollkästchen

Sowohl die Objektrepräsentationen von Optionsfeldern als auch Kontrollkästchen verfügen über die Boole'sche Eigenschaft `checked` (siehe das Rezept »Wie kann ich auf den Selektionszustand von einem Radiobutton oder einer Checkbox zugreifen?« auf Seite 250). Mit der entsprechenden Wertzuweisung selektieren oder deselektieren Sie ein entsprechendes Element.

Selektion von Textfeldern

Die Boole'sche Eigenschaft `checked` hilft aber bei einem Textfeld in jeglicher Form nicht weiter, denn damit kann sie dort nicht arbeiten. Die Objektrepräsentationen von Textfeldern jeglicher Ausprägung 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 <!--
04 function welcherText() {
05   var i = Math.round(Math.random() * 3 ) % 3;
06   window.document.forms[0].elements[i].select();
07 }
08 //-->
09 </script>
10 <body onLoad="welcherText()">
11 <form action="" method="get">
12 <input type="Text" value="Fenchurch">
13 <br>
14 <input type="password" value="Wonko">
15 <br>
16 <textarea cols="20" rows="5">Wie gefällt Ihnen Bournemouth?</textarea>
17 </form>
18 </body>
19 </html>
```

Listing 185: Ein Textfeld wird mit `select()` ausgewählt.

In dem Beispiel gibt es ein kleines Formular (Zeile 11 bis 17) mit einem einzeiligen Eingabefeld, einem Passwortfeld und ein mehrzeiliges Textfeld. Beim `<body>`-Tag ist der Eventhandler `onLoad` notiert. Dieser wird ausgelöst, wenn die Webseite geladen wird, und führt die Funktion `welcherText()` aus. In der Funktion wird mit einem Zufallsmechanismus eine ganze Zahl zwischen 0 und 2 berechnet (Zeile 5). Diese wird als Index für die Selektion eines der Texteingabefelder mit der `select()`-Methode verwendet (Zeile 6).

Wenn ein Anwender nun eine Eingabe vornimmt, erfolgt diese in dem selektierten Textfeld. Mit anderen Worten – dieses Formularelement hat durch die Selektion implizit auch den Fokus erhalten.

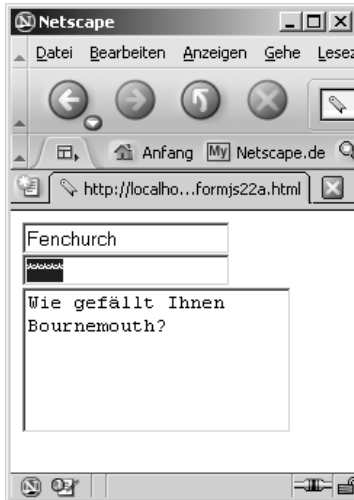


Abbildung 115: Das Passwortfeld wurde zufällig selektiert.

Selektion von Schaltflächen

Wie verhält es sich nun aber mit Schaltflächen? Wenden wir die `select()`-Methode einmal auf Schaltflächen an.

Beispiel (*formjs22b.html*):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04 function welcherText() {
05     var i = 1 + Math.round(Math.random() * 3) % 3;
06     window.document.forms[0].elements[i].select();
07 }
08 function neuerText() {
09     window.document.forms[0].elements[0].value = "";
10 }
11 //-->
12 </script>
13 <body onLoad="welcherText()">
14 <form action="" method="get">
15 <input name="a" value="Brockianisches Ultra-Kricket">
16 <br>
17 <input type="Submit">
18 <input type="reset">
19 <input type="button" value="OK" onClick="reset()"
20     onMouseOver="neuerText()">
21 </form>
22 </body>
23 </html>

```

Listing 186: Drei Schaltflächen werden mit `select()` selektiert.

Grund-
lagen

Core

Formu-
lare

Aus-
nahme

Arrays

Strings

Datum

Platt-
form

DHTML

Events

In dem Beispiel gibt es wieder ein kleines Formular (Zeile 14 bis 21). Hier finden Sie drei verschiedenen Arten an Schaltflächen (eine Submit-, eine Reset- und eine normale Schaltfläche) sowie ein einzeliges Texteingabefeld mit Vorgabewert vor. Beim `<body>`-Tag in Zeile 10 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 auf die Zahl 1 aufaddiert, damit das erste Element des Formulars (das einzelige Texteingabefeld) aus der zufälligen Selektion ausgenommen ist (Zeile 5). 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 den Netscape Navigator 7.1 laden, passiert gar nichts. Ebenfalls im Mozilla, Opera oder Konqueror.



Abbildung 116: Keine Wirkung von `select()` bei Schaltflächen im Navigator

Im Internet Explorer hingegen hat die Methode eine Wirkung, aber nicht die gewünschte Selektion der Schaltfläche. Die Beschriftung der Schaltfläche wird selektiert.

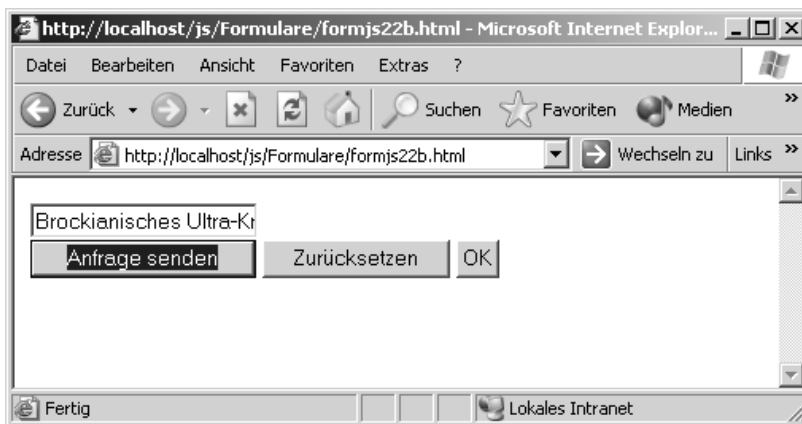


Abbildung 117: Im Internet Explorer wird die Beschriftung der Schaltfläche mit `select()` ausgelöst.

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 19 und 20 einen gewöhnlichen Button mit zwei Eventhandlern (`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 8 bis 10 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 zum Selektieren von Schaltflächen im Allgemeinen nicht weiter.

Achtung

Grundsätzlich ist die Wirkung von `select()` auf Schaltflächen also bezüglich verschiedener Browser indifferent, und man sollte die Verwendung nur mit Vorsicht einsetzen¹.

1. Sie sollten vor allem die Wirkung in verschiedenen Browsern testen. Und es ist nicht vorhersehbar, wie sich neuere Versionen von Browsern verhalten.

Die Lösung ist die Methode `focus()`. Sie müssen bloß Zeile 6 in dem Listing wie folgt austauschen (*formjs22c.html*):

```
window.document.forms[0].elements[i].focus();
```

Listing 187: Die Verwendung von `focus()` selektiert einen Button so, dass er mit Return ausgelöst werden kann.

Diese Technik funktioniert in allen neueren Browsern.

Selektion von Einträgen in einer Auswahlliste

Im Rezept »Wie kann ich testen oder festlegen, ob ein Eintrag in einer Auswahlliste selektiert ist?« auf Seite 261 können Sie nachlesen, dass ein Eintrag in einer Auswahlliste über `selectedIndex` oder `selected` ausgewählt werden kann, etwa so:

```
window.document.forms[0].elements[0].options[i].selected=true;
```

Listing 188: Selektion eines Eintrags in einer Auswahlliste über `selected`

Selektion eines Formularelements unabhängig vom Typ

Wenn Sie unabhängig vom Typ eines Formularelements eine Funktion bauen wollen, die dieses auf jeden Fall selektiert, sollten Sie den Typ auswerten und vier unterschiedliche Behandlungsmethoden bereitstellen, etwa so, wie die nachfolgende Funktionsschablone zeigt (*Beachten Sie dabei auch auf Seite 247 das Rezept »Wie bestimme ich den Typ eines Formularelements?« und dort die möglichen Ausprägungen von 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() == "TEXTAREA") ||
        (window.document.forms[j].elements[i].type.toUpperCase() == "FILE") ||
        (window.document.forms[j].elements[i].type.toUpperCase() == "URL") ||
        (window.document.forms[j].elements[i].type.toUpperCase() == "INT") ||
        (window.document.forms[j].elements[i].type.toUpperCase() == "DATE") ||
        (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 189: 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.

Beispiel (*formjs22.html*):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04 function selektFeld() {
05     var f = window.document.forms[0].length;
06     var i = Math.round(Math.random() * f ) % f;
07     if(
08         (window.document.forms[0].elements[i].type.toUpperCase()
09             == "RADIO") ||
10         (window.document.forms[0].elements[i].type.toUpperCase()
11             == "CHECKBOX")
12     ){
13         window.document.forms[0].elements[i].checked=true;
14     }
15     else if(
16         (window.document.forms[0].elements[i].type.toUpperCase()
17             == "PASSWORD") ||
18         (window.document.forms[0].elements[i].type.toUpperCase()
19             == "TEXT") ||
20         (window.document.forms[0].elements[i].type.toUpperCase()
21             == "TEXTAREA") ||
22         (window.document.forms[0].elements[i].type.toUpperCase()
23             == "FILE") ||
24         (window.document.forms[0].elements[i].type.toUpperCase()
25             == "URL") ||
26         (window.document.forms[0].elements[i].type.toUpperCase()
27             == "INT") ||
28         (window.document.forms[0].elements[i].type.toUpperCase()
29             == "DATE") ||
30         (window.document.forms[0].elements[i].type.toUpperCase()
31             == "FLOAT")
32     ){
33         window.document.forms[0].elements[i].select();
34     }
35     else if(
36         (window.document.forms[0].elements[i].type.toUpperCase()
37             == "BUTTON") ||
38         (window.document.forms[0].elements[i].type.toUpperCase()
39             == "RESET") ||
40         (window.document.forms[0].elements[i].type.toUpperCase()
41             == "SUBMIT")
42     ){
43         window.document.forms[0].elements[i].focus();
44     }
45     else if(window.document.forms[0].elements[i].type.toUpperCase()
46         == "SELECT-ONE") {
47         j = Math.round(
48             Math.random() * window.document.forms[0].elements[i].length ) %
49             window.document.forms[0].elements[i].length;
50         window.document.forms[0].elements[i].selectedIndex=j;

```

Listing 190: Ein Webformular mit automatischer Selektion eines der Webformularelemente

Grund-
lagen

Core

Formu-
lareAus-
nahme

Arrays

Strings

Datum

Platt-
form

DHTML

Events

```
51  }
52  }
53  //-->
54  </script>
55  <body onLoad="selektFeld()">
56  <table >
57  <form action="" method="get"
58  <tr>
59  <td>Userid</td>
60  <td><input type="Text" value="Thor"> </td>
61  </tr>
62  <tr>
63  <td>Passwort</td>
64  <td><input type="password" value="Hammer"> </td>
65  </tr>
66  <tr>
67  <td>Ihre Signatur </td>
68  <td><input type="file"> </td>
69  </tr>
70  <tr>
71  <td>Mann</td>
72  <td><input type="radio" name="geschl" value="0"> </td>
73  </tr>
74  <tr>
75  <td>Frau</td>
76  <td><input type="radio" name="geschl" value="1"> </td>
77  </tr>
78  <tr>
79  <td>Kinder</td>
80  <td><input type="Checkbox" name="kind" value="v"> </td>
81  </tr>
82  <tr>
83  <td>Was darf ich bringen?</td>
84  <td>
85  <select name="getraenk" size="3">
86    <option>Durchblickstrudel</option>
87    <option>Donnergurgel</option>
88  </select>
89  </td>
90  </tr>
91  <tr>
92  <td>Ihr Kommentar </td>
93  <td><textarea cols="20" rows="5">Froschstern</textarea> </td>
94  </tr>
95  <tr>
96  <td><input type="Submit"> </td>
97  <td><input type="reset"> </td>
98  </tr>
99  </form>
100 </table>
101 </body>
102 </html>
```

Listing 190: Ein Webformular mit automatischer Selektion eines der Webformularelemente (Forts.)

In dem Beispiel gibt es ein Formular (Zeile 57 bis 99), 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 vorkommen²¹. Beim `<body>`-Tag in Zeile 55 ist der Eventhandler `onLoad` notiert, der die Funktion `selektFeld()` ausführt. In der Funktion sind ein paar Spezialitäten zu besprechen:

- ▶ In der Zeile 5 wird mit `var f = window.document.forms[0].length;` die Anzahl der Formularelemente bestimmt. Dieser Wert wird in Zeile 6 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 47 bis 49 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.

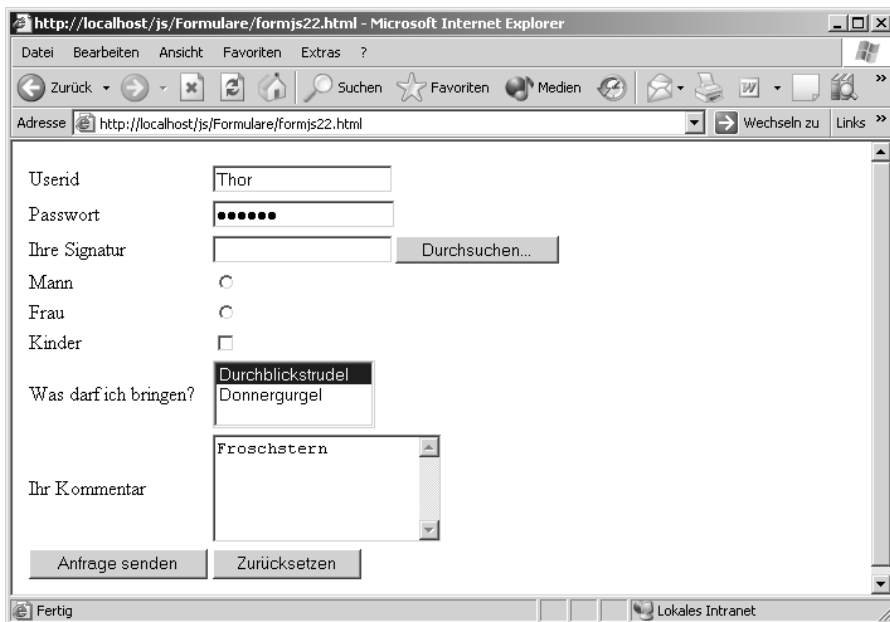


Abbildung 118: Selektion beim Laden

21. Ausnahme sind die Ausprägungen von `type`, die von Browsern allgemein ignoriert werden.

86 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 (*formjs32.html*):

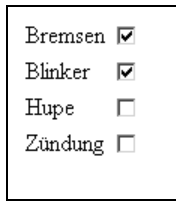
```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function test() {
05   window.setTimeout("window.document.forms[0].elements[0].click()",1000);
06   window.setTimeout("window.document.forms[0].elements[1].click()",2000);
07   window.setTimeout("window.document.forms[0].elements[2].click()",3000);
08   window.setTimeout("window.document.forms[0].elements[3].click()",4000);
09 }
10 //-->
11 </script>
12 <body onLoad="test()">
13 <form action="" method="get">
14 <table >
15 <tr>
16   <td>Bremsen </td>
17   <td><input type="Checkbox" name="a" value="1"> </td>
18 </tr>
19 <tr>
20   <td>Blinker </td>
21   <td><input type="Checkbox" name="b" value="2"> </td>
22 </tr>
23 <tr>
24   <td>Hupe </td>
25   <td><input type="Checkbox" name="c" value="3"> </td>
26 </tr>
27 <tr>
28   <td>Zündung </td>
29   <td><input type="Checkbox" name="d" value="4"> </td>
30 </tr>
31 </table>
32 </form>
33 </body>
34 </html>
```

Listing 191: Verwendung der `click()`-Methode

Das Beispiel verwendet ein Formular (Zeile 13 bis 32) mit vier Kontrollkästchen, die in Tabellen aus zwei Spalten angeordnet sind. Damit ist die optische Gestaltung harmonischer als bei einem Formular ohne Tabellen. Sonst hat die Tabellenstruktur aber keine Bedeutung.

In Zeile 12 finden Sie im `<body>`-Tag den Eventhandler `onLoad`, der beim Laden der Webseite die Funktion »`test()`« aufruft. Diese Funktion (von Zeile 4 bis 9 definiert) verwendet die `win-`

down-Methode `setTimeout()`, um jeweils im Abstand von einer Sekunde die verschiedenen Kontrollkästchen anzuklicken.



Bremsen	☒
Blinker	☒
Hupe	☐
Zündung	☐

Abbildung 119: Nacheinander werden die Kontrollkästchen automatisch durch das Skript angeklickt.

87 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 HTML-Tag machen, also die Anwendung von den bekannten Eventhandlern, die auch bei Bildern, Überschriften etc. zum Einsatz kommen. Es hängt (wie bei all diesen Ereignissen) am jeweiligen Browser, ob er diese Aktionen unterstützt. Denn die Fähigkeiten von Formularen gehen weiter. Es gibt zum einen spezielle Eventhandler, die unmittelbar auf die Anwendung in Formularen abgestimmt sind. Zum anderen sind die meisten Formularelemente sensitiv, und Browser verhalten sich bezüglich der Unterstützung von Ereignissen wie bei Hyperlinks. Schauen wir uns die verschiedenen Eventhandler an, die im Zusammenhang mit Formularen von Bedeutung sind.

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 action="" method="get" onClick="alert(1)">
```

Listing 192: Ein Formular mit Vertretern der verschiedenen Formularelemente, bei denen `onClick` notiert wird

```
05 <tr>
06 <td>
07 Userid
08 </td>
09 <td>
10 <input type="Text" onClick="alert(2)">
11 </td>
12 </tr>
13 <tr>
14 <td>
15 Ihre Datei
16 </td>
17 <td>
18 <input type="file"onClick="alert(3)">
19 </td>
20 </tr>
21 <tr>
22 <td>
23 Links
24 </td>
25 <td>
26 <input type="radio" name="r" value="0" onClick="alert(4)">
27 </td>
28 </tr>
29 <tr>
30 <td>
31 Rechts
32 </td>
33 <td>
34 <input type="radio" name="r" value="1" onClick="alert(5)">
35 </td>
36 </tr>
37 <tr>
38 <td>
39 Werbung
40 </td>
41 <td>
42 <input type="Checkbox" name="kind" value="v" onClick="alert(6)">
43
44 </td>
45 </tr>
46 <tr>
47 <td>
48 Welche Stadt
49 </td>
50 <td>
51
52 <select name="getraenk" size="3" onClick="alert(7)">
53   <option onClick="alert(8)">Hinterdemon</option>
54   <option onClick="alert(9)">Anderbach</option>
```

Listing 192: Ein Formular mit Vertretern der verschiedenen Formularelemente, bei denen onClick notiert wird (Forts.)

```

55 </select>
56
57 </td>
58 </tr>
59 <tr>
60 <td>
61 Kommentar
62 </td>
63 <td>
64 <textarea cols="20" rows="5" onClick="alert(10)"></textarea>
65 </td>
66 </tr>
67 <tr>
68 <td>
69 <input type="Submit" onClick="alert(11)">
70 </td>
71 <td>
72 <input type="reset" onClick="alert(12)">
73 </td>
74 </tr>
75 <tr>
76 <td>
77 <input type="button" value="OK" onClick="alert(13)">
78 </td>
79 <td>
80
81 </td>
82 </tr>
83 </form>
84 </table>
85 </body>
86 </html>

```

Listing 192: Ein Formular mit Vertretern der verschiedenen Formularelemente, bei denen onClick notiert wird (Forts.)

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 einsortiert 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.

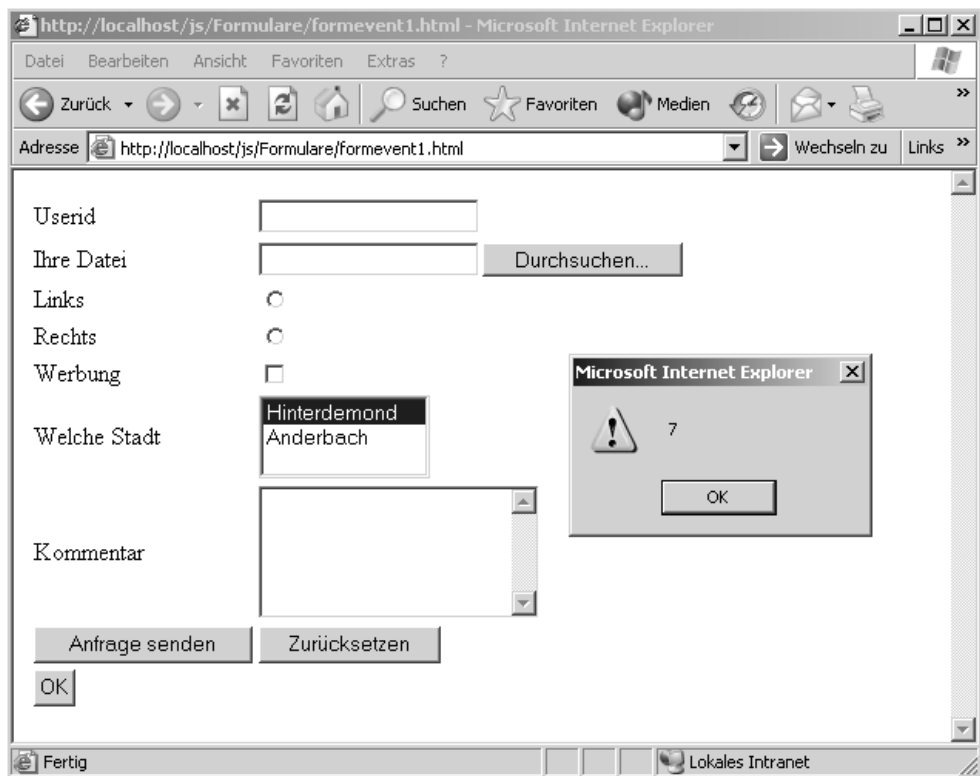


Abbildung 120: Test des Eventhandlers `onClick` bei verschiedenen Strukturen eines Formulars

Zwar werden Sie in einem Test des Beispiels im Internet Explorer bei allen anderen Formularelementen `onClick` auslösen können und wenig Weiteres an Überraschungen entdecken. Aber wie sieht es in anderen Browsern aus?

Schauen wir uns zunächst 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 einzeliges Eingabefeld wird `onClick` nicht (!) auslösen, weder beim normalen einzeligen Eingabefeld als auch dem zugehörigen einzeligen 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 `<select>` Sinn.
- ▶ Ein Klick in das mehrzeilige Textfeld wird `onClick` nicht (!) auslösen. Der Eventhandler macht also bei `<textarea>` keinen Sinn.

Sie können nun 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 verhält sich die neuere Variante – etwa der Navigator 7.1²²?

- ▶ Ein Klick in ein einzeliges Eingabefeld oder ein mehrzeiliges Textfeld wird `onClick` auslösen. So weit ist das wie beim Internet Explorer.
- ▶ Der Eventhandler des 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 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. Die beiden Browser unterstützen etwa – auch in den neuen Versionen – nicht die Auslösung von `onClick` beim Klick in das Eingabefeld beim Dateifeld. Der Konqueror unterstützt keinen Klick in ein mehrzeiliges Textfeld. Opera löst beide Eventhandler bei der Auswahlliste aus etc. Es ist leider so, dass je nach Browserversion 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 von einem Eventhandler, der auf das Zusammenspiel mit Formularen besser abgestimmt ist.

88 Wie setze ich `onmouseover` bei Formularelementen ein?

Ein in vielen Situationen sehr beliebter Eventhandler ist `onmouseover`. Allgemein wird der Eventhandler `onmouseover` ausgelöst, wenn ein Anwender ein sensibles Element einer Webseite überstreicht, bei dessen Tag der Eventhandler angegeben ist. Aber um es kurz zu

22. Das Verhalten ist beim Mozilla 1.6 gleich.

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.

Tipp

Eine sinnvolle Anwendung für `onMouseOver` bei Formularen ist die Anzeige von optionalen Hilfeinformationen zu einzelnen Formularelementen in 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 193: Zusatzinformationen zu einem Formularfeld in der Statuszeile des Browsers

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, der Navigator 7.1 beziehungsweise Mozilla 1.6 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.

Hinweis

Sollten Sie `onMouseOver` 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 mehr noch als bei `onClick` umfangreiche Tests auf allen verfügbaren (beziehungsweise zu unterstützenden) Plattformen unabdingbar, und/oder eine Browserweiche. 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.

89 Wie setze ich `onFocus` bei Formularelementen ein?

Der Eventhandler `onFocus` ist zwar nicht ausschließlich auf Formularelemente beschränkt, macht aber in Verbindung mit Formularfeldern besonderen Sinn. Außerdem wird er sehr weit reichend in Browsern unterstützt. Der Eventhandler wird ausgelöst, wenn ein Element den Fokus erhält.

Achtung

Beachten Sie, dass bei dem 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 von dem Dialogfenster 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 der Art:

```
<input type=button value ="Ende" onFocus='alert("Nö - keine Chance")'>
```

Listing 194: Ein unglückliches Perpetuum mobile

Beispiel (*formevent2.html*):

```
01 <html>
02 <body>
03 <table >
04 <form action="" method="get" onFocus="alert('1')">
05 <tr>
06 <td>
07 Userid
08 </td>
09 <td>
10 <input type="Text" onFocus="window.status = '2'">
11 </td>
12 </tr>
13 <tr>
14 <td>
15 Ihre Datei
16 </td>
17 <td>
18 <input type="file"onFocus="window.status= '3'">
19 </td>
20 </tr>
21 <tr>
22 <td>
23 Links
24 </td>
25 <td>
26 <input type="radio" name="r" value="0" onFocus="window.status = '4'">
27 </td>
28 </tr>
29 <tr>
30 <td>
31 Rechts
32 </td>
33 <td>
34 <input type="radio" name="r" value="1" onFocus="window.status = '5'">
35 </td>
```

Listing 195: Die Anwendung von onFocus

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

```
36 </tr>
37 <tr>
38   <td>
39     Werbung
40   </td>
41   <td>
42     <input type="Checkbox" name="kind" value="v" onFocus="window.status = '6'">
43   </td>
44 </tr>
45 <tr>
46   <td>
47     Welche Stadt
48   </td>
49   <td>
50     <select name="getraenk" size="3" onFocus="window.status = '7'">
51       <option onFocus="window.status = '8'">
52         Hinterdemon
53       </option>
54       <option onFocus="window.status = '9'">
55         Anderbach
56       </option>
57     </select>
58   </td>
59 </tr>
60 <tr>
61   <td>
62     Kommentar
63   </td>
64   <td>
65     <textarea cols="20" rows="5" onFocus="window.status = '10'">
66   </textarea>
67 </td>
68 </tr>
69 <tr>
70   <td>
71     <input type="Submit" onFocus="window.status = '11'">
72   </td>
73   <td>
74     <input type="reset" onFocus="window.status = '12'">
75   </td>
76 </tr>
77 <tr>
78   <td>
79     <input type="button" value="OK" onFocus="window.status = '13'">
80   </td>
81   <td>
82   </td>
83 </tr>
84 </form>
85 </table>
86 </body>
87 </html>
```

Listing 195: 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` durch `onFocus` und `alert()` durch eine Wertzuweisung an `window.status`²³ 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 erhält²⁴. 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 logisch²⁵, 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.

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 – dem Navigator 7.1 und Mozilla 1.6! Wenn das zu dem Button zum Öffnen eines Dateiauswahlfensters zugehörnde einzelzeilige Eingabefeld den Fokus erhält, wird der Eventhandler nicht ausgelöst! Beim Klick auf den Button hingegen schon. Und zwar *vor* dem Öffnen von dem Dateiauswahlfenster. Wenn Sie – statt mit einem Mausklick – den Fokus mit dem Tabulator weiterreichen, erkennen Sie den Effekt besonders gut.

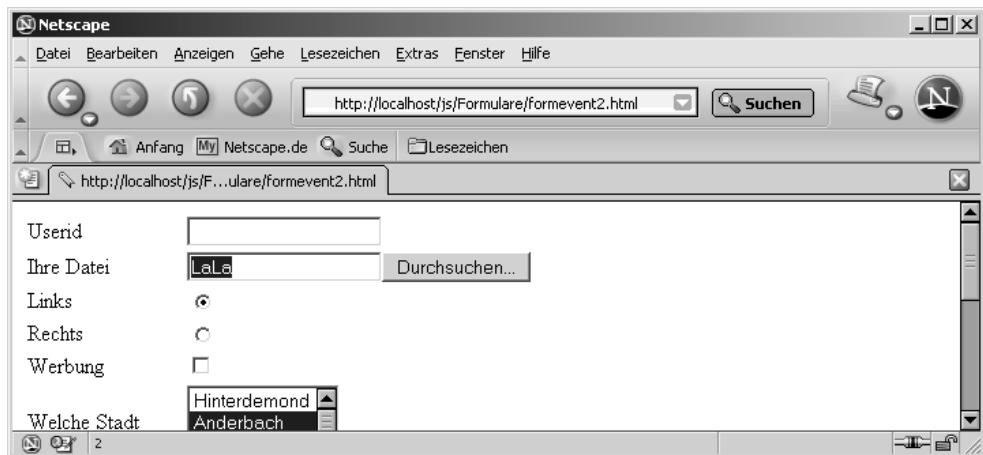


Abbildung 121: Das Eingabefeld vor dem Dateiauswahlbutton hat den Fokus, aber die Statuszeile zeigt immer noch die von dem einzelzeiligen Eingabefeld davor ausgelöste Meldung an.

23. Das gibt eine Meldung in der Statuszeile des Browsers aus.

24. Eine Meldung in der Statuszeile würde unmittelbar von der Meldung eines Formularelements überschrieben.

25. Wobei – seit wann ist das Verhalten eines Browsers schon logisch ;-)?

Tipp

Allgemein reichen Sie in einer Webseite den Fokus mit dem  weiter. Neuere Browser unterstützen das grundsätzlich, und der Ersteller der Webseite muss das nicht mehr explizit mit einem HTML-Attribut (`tabindex`) programmieren. Beachten Sie jedoch beim Weiterreichen des Fokus per Tastatur, dass Sie bei einigen Browsern bei einer Gruppe zusammengehörender Elemente (etwa Optionsfeldern) mit dem  nur das selektierte Feld der Gruppe erreichen. Wenn Sie innerhalb der Gruppe den Fokus mit der Tastatur weiterreichen wollen, verwenden Sie die Pfeiltasten. 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  +  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  an das Feld weiterreichen, wird der Eventhandler ausgelöst. Dafür wird der Eventhandler gar nicht ausgelöst, wenn Sie auf den Button zum Öffnen von dem Dateiauswahldialog 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 – wie Web-Verhältnisse – sehr gut und einheitlich unterstützt wird und bei allen Formularelementen (außer Button) `onClick` in jedem Fall vorgezogen werden sollte, wenn es keine gravierenden Ungereimtheiten mit der zu programmierenden Situation gibt.

90 Wie setze ich `onBlur` bei Formularelementen ein?

Der Eventhandler `onBlur` ist zwar nicht ausschließlich auf Formularelemente beschränkt, macht aber in Verbindung mit Formularfeldern besonderen Sinn. Außerdem wird er sehr weit reichend in Browsern unterstützt. Der Eventhandler wird ausgelöst, wenn ein Element den Fokus verliert.

Beispiel (*formevent3.html*):

```

01 <html>
02 <body>
03 <table >
04 <form action="" method="get" onBlur="alert('1')">
05 <tr>
06 <td>
07 Userid
08 </td>
09 <td>
10 <input type="Text" onBlur="window.status = '2'">
11 </td>
12 </tr>
13 <tr>

```

Listing 196: Die Anwendung von `onBlur` bei Formularelementen

```

14 <td>
15 Ihre Datei
16 </td>
17 <td>
18 <input type="file"onBlur="window.status= '3'">
19 </td>
20 </tr>
21 <tr>
22 <td>
23 Links
24 </td>
25 <td>
26 <input type="radio" name="r" value="0" onBlur="window.status = '4'">
27 </td>
28 </tr>
29 <tr>
30 <td>
31 Rechts
32 </td>
33 <td>
34 <input type="radio" name="r" value="1" onBlur="window.status = '5'">
35 </td>
36 </tr>
37 <tr>
38 <td>
39 Werbung
40 </td>
41 <td>
42 <input type="Checkbox" name="kind" value="v" onBlur="window.status = '6'">
43 </td>
44 </tr>
45 <tr>
46 <td>
47 Welche Stadt
48 </td>
49 <td>
50 <select name="getraenk" size="3" onBlur="window.status = '7'">
51   <option onBlur="window.status = '8'">
52     Hinterdemon
53   </option>
54   <option onBlur="window.status = '9'">
55     Anderbach
56   </option>
57 </select>
58 </td>
59 </tr>
60 <tr>
61 <td>
62 Kommentar
63 </td>
64 <td>

```

Listing 196: Die Anwendung von onBlur bei Formularelementen (Forts.)

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

```
65 <textarea cols="20" rows="5" onBlur="window.status = '10'">
66 </textarea>
67 </td>
68 </tr>
69 <tr>
70 <td>
71 <input type="Submit" onBlur="window.status = '11'">
72 </td>
73 <td>
74 <input type="reset" onBlur="window.status = '12'">
75 </td>
76 </tr>
77 <tr>
78 <td>
79 <input type="button" value="OK" onBlur="window.status = '13'">
80 </td>
81 <td>
82 </td>
83 </tr>
84 </form>
85 </table>
86 <form>
87 <input>
88 </form>
89 </body>
90 </html>
```

Listing 196: Die Anwendung von onBlur bei Formularelementen (Forts.)

Das Beispiel entspricht dem, das zu `onFocus` eingesetzt wurde. Nur wurde der Eventhandler `onFocus` durch `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 Browserhersteller 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 in unterschiedlichen Browsern, die auch bei `onFocus` auftreten. Allerdings sind die betreffenden Situationen nicht vollkommen identisch mit den Situationen, bei denen Differenzen in den Browsern bei `onFocus` auftreten²⁶.

26. Das wäre ja auch zu einfach ;-).

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 aber funktioniert der Eventhandler sehr gut, und man kann damit rechnen, dass beim Wegnehmen des Fokus von einem Formularfeld eine fast nahtlose Unterstützung in Browsern gewährleistet ist (noch besser als bei `onFocus`).

91 Wie setze ich `onChange` bei Formularelementen ein?

Der Eventhandler `onChange`) erlaubt die Reaktion für den Fall, dass ein Element in einem Formular seinen Wert geändert hat. Es wird in den 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 *
document.forms[0].elements[1].value;
05   }
06 </script>
07 <body>
08   <form>
09     <input onChange="change()">
10     <input onChange="change()">
11   </form>
12 </body>
13 </html>
```

Listing 197: 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 4).

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 am sinnvollsten bei den Formular-Tags `<input>` und `<textarea>`, 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 beinhaltet, wollen wir auf ein Beispiel verzichten.

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 notiert.

Beispiel:

```
<form name="meinForm" action="mailto:hotdampf@yahoo.de" method="post"
onSubmit="weg()">
```

Listing 198: Die Anwendung von `onSubmit` – hier in Verbindung mit einem 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.

Im Allgemeinen setzt man jedoch JavaScript-Funktionalität beim Abschicken eines Formulars ein, um die Versendung der Daten bei gewissen Konstellationen abubrechen. 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 beim Plausibilisieren von Formulardaten aus.

Beispiel:

```
<form name="meinForm" action="mailto:hotdampf@yahoo.de" method="post"
onSubmit="return weg()">
```

Listing 199: Die Anwendung von `onSubmit` – hier mit vorangestelltem `return`

Schauen wir uns ein vollständiges Beispiel an (*formsubmit1.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function weg() {
05     if(Math.random()*2 > 1) {
06         alert("Daten werden versendet");
07         return true;
08     }
09     else {
10         alert("Daten werden nicht versendet");
11         return false;
12     }
13 }
14 //-->
15 </script>
16 <noscript></noscript>
17 <body>
18 <table >
19 <form action="" method="get" onSubmit="return weg()">
20 <tr>
21 <td>
22 Name
23 </td>
24 <td>
25 <input type="text" name="nn">
26 </td>
27 </tr>
28 <tr>
29 <td>
30 Vorname
31 </td>
32 <td>
33 <input type="text" name="vn">
34 </td>
35 </tr>
36 <tr>
37 <td>
38 <input type="Submit">
39 </td>
40 <td>
41 </td>
42 </tr>
43 </form>
44 </table>
45 </body>
46 </html>
```

Listing 200: Das Abschicken der Formulardaten wird in manchen Konstellationen abgebrochen.

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

In dem Beispiel wird mit `onSubmit` die Funktion `weg()` aufgerufen (mit vorangestelltem `return`), die einen Booleschen Wert zurückgibt (in Zeile 19). In der Funktion `weg()` (Zeile 4 bis 13) wird mit einem Zufallsmechanismus festgelegt²⁷, ob die Funktion `true` oder `false` zurückgibt (Zeile 5). Zur Verdeutlichung, ob die Daten verschickt werden oder nicht, wird für unsere Testsituation jeweils eine entsprechende Meldung angezeigt. Das wird in der Praxis so natürlich nicht gemacht (zumindest nicht mit einem Dialogfenster, das vom Anwender einen Klick erwartet).

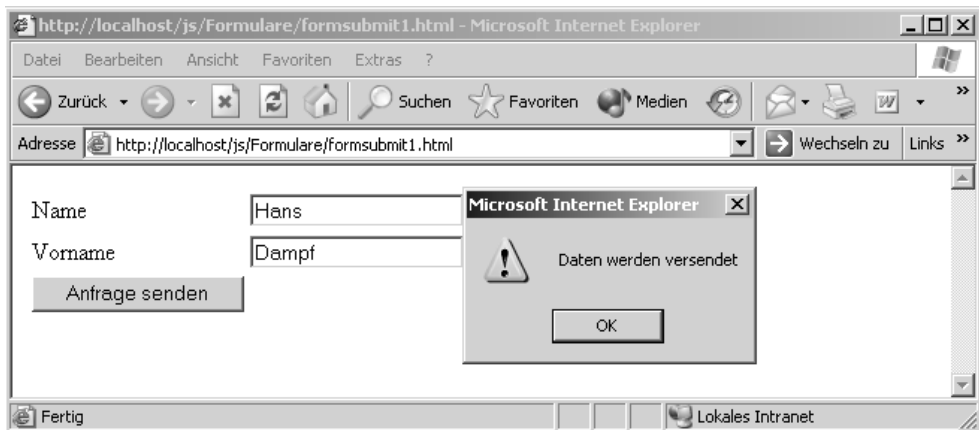


Abbildung 122: Die mit `onSubmit` aufgerufene Funktion liefert `true`, und die Formulardaten werden versendet.



Abbildung 123: Die mit `onSubmit` aufgerufene Funktion liefert `false`, und die Formulardaten werden nicht versendet.

27. In der Praxis wird hier natürlich eine Plausibilisierung stattfinden.

Noch einmal ausdrücklich die Warnung, dass `onSubmit` nicht beim Submit-Button, sondern beim `<form>`-Tag zu notieren ist.

Wie verhält sich der Eventhandler `onSubmit` jetzt aber, wenn Sie mit der `submit()`-Methode des Formulars die Formulardaten verschicken? Verändern wir das Beispiel von oben leicht. Sie müssen nur die Zeile 38 wie folgt abändern:

```
38 <input type="button" value="OK" onClick="this.form.submit()">
```

Listing 201: Die Zeile 38 soll so geändert werden.

Wir verwenden jetzt statt eines Submit-Buttons eine gewöhnliche Schaltfläche und lösen das Verschicken der Daten mit Hilfe von dem `onClick`-Eventhandler 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 werden²⁸, sondern die Werte in dem Formular. Die Eigenschaft `form` von dem Button ist eine Referenz auf das umgebende Formular. Also können wir mit `this.form` auf dieses Formular zugreifen und dann dessen `submit()`-Methode aufrufen.

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.

Obwohl man geneigt ist, beim Zurücksetzen eines Formulars mit der `reset()`-Methode eine gleiche Reaktion zu erwarten, ist das Verhalten dort anders. 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 295.*

Wenn Sie also mit der `submit()`-Methode des aktuellen Formulars die Formulardaten verschicken, 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 auslöst. Diese Funktion wird dann im Formular aufgerufen (beispielsweise mit einem gewöhnlichen Button). Stricken wir das Beispiel entsprechend um.

28. :-)

Beispiel (*forms submit2.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function weg(f) {
05     if(Math.random()*2 > 1) {
06         alert("Daten werden versendet");
07         f.form.submit();
08     }
09     else {
10         alert("Daten werden nicht versendet");
11     }
12 }
13 //-->
14 </script>
15 <noscript></noscript>
16 <body>
17 <table >
18 <form action="" method="get">
19 <tr>
20 <td>
21 Name
22 </td>
23 <td>
24 <input type="text" name="nn">
25 </td>
26 </tr>
27 <tr>
28 <td>
29 Vorname
30 </td>
31 <td>
32 <input type="text" name="vn">
33 </td>
34 </tr>
35 <tr>
36 <td>
37 <input type="button" value="OK" onClick="weg(this)">
38 </td>
39 <td>
40 </td>
41 </tr>
42 </form>
43 </table>
44 </body>
45 </html>
```

Listing 202: Kontrolliertes Versenden der Formulardaten mit der submit()-Methode

Gegenüber dem Beispiel mit der Verwendung von dem Eventhandler `onSubmit` gibt es in der neuen Version zwei gravierende Änderungen.

1. Der Eventhandler `onSubmit` beim `<form>`-Tag (Zeile 18) ist nicht mehr vorhanden. Stattdessen gibt es in Zeile 37 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 4 bis 12) 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 tun (höchstens ein Hinweis an den Anwender).

92 Wie kann ich auf das Zurücksetzen eines Formulars reagieren?

Die Reaktion auf das Zurücksetzen eines Formulars über den Eventhandler `onReset` oder die `reset()`-Methode eines Formulars per JavaScript erlaubt die Erweiterung der Funktionalität, welche über die durch 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 stattdessen 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 203: 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 204: Die Anwendung von onReset – hier mit vorangestelltem return

Schauen wir uns ein vollständiges Beispiel an (*formreset1.html*):

```
01 <html>  
02 <body>  
03 <table >  
04 <form action="" method="get"  
05 onReset="return confirm(  
06 'Das Formular wird zurückgesetzt. Sind Sie sicher, dass Sie das wirklich  
07 wollen?')">  
08 <tr>  
09 <td>  
10 Name  
11 </td>  
12 <td>  
13 <input type="text" name="nn">  
14 </td>  
15 </tr>  
16 <tr>  
17 <td>  
18 Vorname  
19 </td>  
20 <td>  
21 <input type="text" name="vn">  
22 </td>  
23 </tr>  
24 <tr>  
25 <td>  
26 <input type="Reset">  
27 </td> </tr>  
28 </table>  
29 </body>  
30 </html>
```

Listing 205: Zurücksetzen des Formulars nur nach Bestätigung

Das Beispiel verwendet ein Formular (Zeile 4 bis 29). 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. Diesen Wert wertet der Eventhandler direkt aus, wenn dem Aufruf `return` vorangestellt ist. Wenn also ein Anwender drückt, liefert `confirm()` den Wert `true`. Dieser Wert wird an den Eventhandler weitergereicht und das Formular auf seinen Ausgangszustand zurückgesetzt. Drückt der Anwender die -Schaltfläche, liefert `confirm()` den Wert `false`, und das Zurücksetzen des Formulars wird nicht durchgeführt.



Abbildung 124: Das Formular wird nur zurückgesetzt, wenn der Anwender das bestätigt.

Wie verhält sich jetzt aber der Eventhandler `onReset`, wenn Sie mit der `reset()`-Methode des Formulars das Formular zurücksetzen? Verändern wir das Beispiel von oben leicht. Sie müssen nur die Zeile 25 wie folgt abändern:

```
25 <input type="button" value="Abbruch" onClick="this.form.reset()">
```

Listing 206: 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 von dem `onClick`-Eventhandler 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` von dem Button 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!

Obwohl es kaum erklärbar ist, besteht so eine Beziehung in der verwandten Situation beim Abschicken von Formulardaten zwischen dem Eventhandler `onSubmit` und der `submit()`-Methode nicht! Dort wird der Eventhandler `onSubmit` nicht von der `submit()`-Methode ausgelöst. Siehe dazu das Rezept »Wie kann ich auf das Abschicken eines Formulars reagieren? Der Eventhandler `onSubmit` und die `submit()`-Methode« auf Seite 290.

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 auslöst. Stricken wir also auch das Beispiel entsprechend um.

Beispiel (*formreset3.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function sicher(f) {
05     if(confirm(
06         'Das Formular wird zurückgesetzt. Sind Sie sicher, dass Sie das wirklich
07         wollen?')) {
08         alert("Die Daten werden zurückgesetzt");
09         f.form.reset();
10     }
11     else {
12         alert("Das Formular wird nicht zurückgesetzt");
13     }
14 }
15 </script>
16 <noscript></noscript>
17 <body>
18 <table >
19 <form action="" method="get">
20 <tr>
21 <td>
22 Name
23 </td>
24 <td>
25 <input type="text" name="nn">
26 </td>
27 </tr>
28 <tr>
29 <td>
30 Vorname
31 </td>
32 <td>
```

Listing 207: Zurücksetzen des Formulars in einer eigenen Funktion mit `reset()`

```

33 <input type="text" name="vn">
34 </td>
35 </tr>
36 <tr>
37 <td>
38 <input type="button" value="Abbrechen" onClick="sicher(this)">
39 </td>
40 <td>
41 </td>
42 </tr>
43 </form>
44 </table>
45 </body>
46 </html>

```

Listing 207: 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.

1. Der Eventhandler `onReset` beim `<form>`-Tag in Zeile 19 ist nicht mehr vorhanden. Stattdessen gibt es in Zeile 38 bei einem einfachen Button den Eventhandler `onClick`, der die Funktion `sicher()` 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 `sicher()` (Zeile 4 bis 13) 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 5 und 6 innerhalb der `if`-Bedingung). Der Anwender erhält zusätzlich einen Hinweis, wobei das in der Praxis zu viel des Guten ist. Wenn der Anwender das Zurücksetzen nicht bestätigt, braucht die `sicher()`-Methode gar nichts tun. In dem Beispiel steht nur ein Hinweis an den Anwender, damit Sie im Test etwas erkennen.

93 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 einzeiligen Eingabefeld hervorragend mit HTML lösen (siehe das Rezept »Wie kann ich rein mit HTML bei einem einzeiligen Eingabefeld eine maximale Anzahl der einzugebenden Zeichen festlegen?« auf Seite 177). 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 freie Texteingabe über ein `Textarea`-Feld oder den Rück-

gabewert von einer `prompt()`-Eingabe Maßnahmen ergreifen²⁹, wobei nicht alle Rezepte bei der `prompt()`-Eingabe greifen werden.

Sofern Sie bei einem einzeiligen Eingabefeld die Anzahl der maximal erlaubten Zeichen mit 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 von einem Rezept 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 Sinn³⁰.

94 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 Vorgehensweise 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.

Achtung

Beachten Sie aber, dass diese Eventhandler zur Auswertung der Tastatur nur sehr eingeschränkt in Browsern unterstützt werden. Insbesondere die Eventhandler `onKeyDown` und `onKeyUp` werden in kaum einem Browser unterstützt. Falls Sie tatsächlich die Kontrolle beim unmittelbaren Eingeben eines Zeichens benötigen, verwenden Sie zur Sicherheit nur `onKeyPress`. Dieser Eventhandler ist zumindest einigermaßen verfügbar.

²⁹. Hier sieht HTML keinerlei Möglichkeiten vor.

³⁰. Es sei denn, der Rückgabewert wird in einem (versteckten) Formularfeld zwischengespeichert. Diese Vorgehensweise ist jedoch in der Praxis vollkommen unüblich.

Hinweis

Beachten Sie auch das verwandte Rezept »Wie kann ich mit JavaScript bei freier Texteingabe eine minimale Anzahl der einzugebenden Zeichen festlegen?« auf Seite 317, 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 HTML bereits liefert. Also der einfache Stopp der Entgegennahme von Zeichen in einem einzeiligen Listenfeld an einer gewissen Grenze. Das ist zwar möglich, aber die Neuerfindung des Rades ist ziemlich unsinnig. Anders ist es, wenn Sie ein Editorfeld über den `<textarea>`-Container betrachten. Diese Eingabemöglichkeit kann rein mit 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 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.

```
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 208: 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.

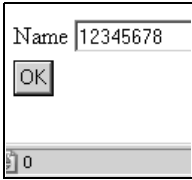


Abbildung 125: Maximal acht Zeichen sind erlaubt – im Eingabefeld sind acht Zeichen eingegeben und in der Statuszeile wird der Wert der Variablen `start` angezeigt.

In Zeile 4 kontrollieren Sie, ob die Anzahl der eingegebenen Zeichen größer als der Wert ist, den Sie als Grenze vorgeben. Wahrscheinlich 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 soll³¹. 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 des `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 von dem Eingabefeld 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.

Tip

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 auch gar nicht zur Kenntnis genommen. 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:

31. Da nur dafür in gängigen Browsern einigermaßen Unterstützung gewährleistet ist – siehe oben.

```
f.value = f.value.substring(0,max-1);
```

Listing 209: Austausch des letzten Zeichens

Mit der Methode `substring(0, max - 1)` erhalten wir die ersten Eingabezeichen des Anwenders abzüglich des letzten Zeichens. Dieser Teilstring wird als aktueller Wert von dem Eingabefeld 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 <!--
04 function maxAnzahl(f,max) {
05   if(f.value.length > max - 1) {
06     f.value = f.value.substring(0,max-1);
07     window.status = "Sie haben zu viele Zeichen eingegeben";
08   }
09 }
10 function statusLeer() {
11   window.status="";
12 }
13 //-->
14 </script>
15 <noscript></noscript>
16 <body>
17 <table >
18 <form action="" method="get">
19 <tr>
20 <td>
21 Name
22 </td>
23 <td>
24 <input type="text" name="nn" onkeypress="maxAnzahl(this,20)"
25   onBlur="statusLeer()">
26 </td>
27 </tr>
28 <tr>
29 <td>
30 Vorname
31 </td>
32 <td>
33 <input type="text" name="vn" onkeypress="maxAnzahl(this,15)"
34   onBlur="statusLeer()">
```

Listing 210: Ein vollständiges Beispiel, in dem mehrere Eingabefelder beschränkt werden

Grund-
lagen

Core

Formu-
lare

Aus-
nahme

Arrays

Strings

Datum

Platt-
form

DHTML

Events

```

35 </td>
36 </tr>
37 <tr>
38 <td>
39 <input type="submit" value="OK">
40 </td>
41 <td>
42 </td>
43 </tr>
44 </form>
45 </table>
46 </body>
47 </html>

```

Listing 210: Ein vollständiges Beispiel, in dem mehrere Eingabefelder beschränkt werden (Forts.)

Das Beispiel verwendet ein Formular (Zeile 18 bis 46) mit zwei einzeiligen Eingabefeldern und einem Submit-Button, die in Tabellen aus zwei Spalten angeordnet sind. Damit ist die optische Gestaltung harmonischer als bei einem Formular ohne Tabellen. Sonst hat die Tabellenstruktur aber keine Bedeutung.

Bei beiden Eingabefeldern ist der Eventhandler `onKeyPress` notiert (in Zeile 25 und 34), worüber die Funktion `maxAnzahl()` aufgerufen wird. Diese Funktion (von Zeile 4 bis 9 definiert) beschränkt die Anzahl der erlaubten Zeichen und tauscht permanent das letzte Zeichen aus³². 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 und der Vorname 15 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.

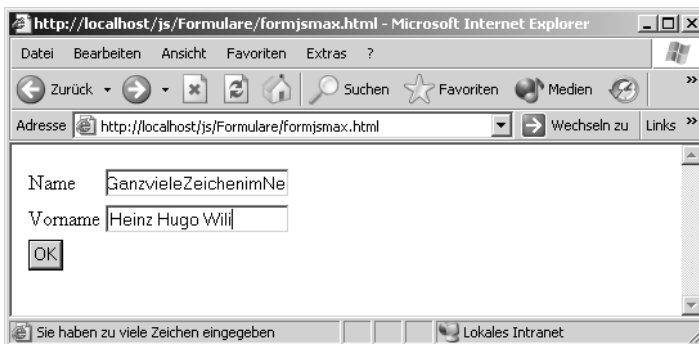


Abbildung 126: Beschränkung der Anzahl der Eingabezeichen samt Hinweis in der Statuszeile

32. Wie oben beschrieben.

Das folgende Beispiel funktioniert nahezu identisch, soll aber die Anwendung bei einem mehrzeiligen Editorfeld demonstrieren (*formjsmax1.html*):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04 function maxAnzahl(f,max) {
05     var start = f.value.length - max + 1;
06     window.status = start;
07     if(f.value.length > max - 1) {
08         window.status = "Sie haben zu viele Zeichen eingegeben";
09         f.value = f.value.substring(0,max);
10     }
11 }
12 function statusLeer() {
13     window.status="";
14 }
15 //-->
16 </script>
17 <noscript></noscript>
18 <body>
19 <table>
20 <form action="" method="get">
21 <tr>
22 <td>
23 Kommentar
24 </td>
25 <td>
26 <textarea cols="30" rows="5" name="nn" onkeypress="maxAnzahl(this,10)"
27     onBlur="statusLeer()">
28 </textarea>
29 </td>
30 </tr>
31 <tr>
32 <td>
33 <input type="submit" value="OK">
34 </td>
35 <td>
36 </td>
37 </tr>
38 </form>
39 </table>
40 </body>
41 </html>

```

Listing 211: Beschränkung der Anzahl an Zeichen, die ein Anwender in einem mehrzeiligen Textfeld eingeben kann

Der wesentliche Unterschied zu der ersten Variante ist, dass hier ein Formular mit einem mehrzeiligen Eingabefeld vorliegt (Zeile 26 bis 28). Die Funktion zum Abfragen und Setzen des Wertes ist unverändert. Das ist vor allem deshalb bemerkenswert, da der `<textarea>`-Container nicht über ein `value`-Parameter verfügt und eine Vorbelegung über den Text

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

innerhalb des Containers erfolgt (siehe das Rezept »Wie kann ich ein mehrzeiliges Eingabefeld generieren?« auf Seite 198).

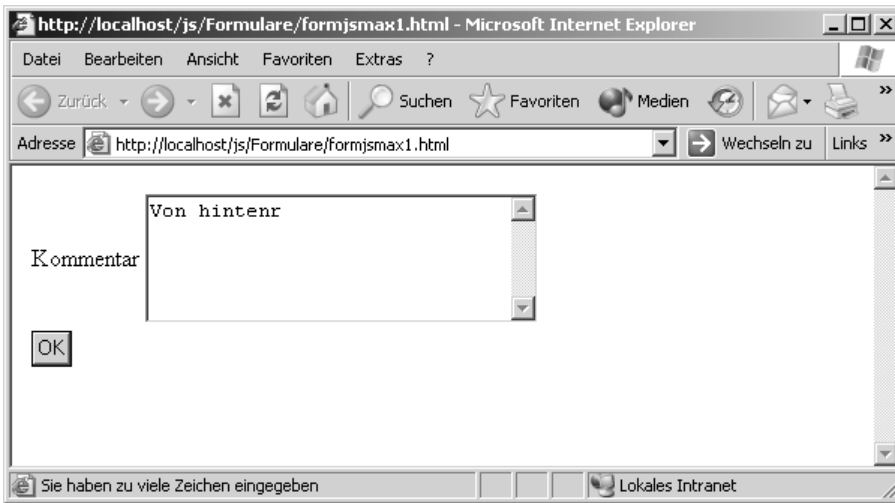


Abbildung 127: Der Zugriff auf ein mehrzeiliges Textfeld funktioniert analog.

Hinweis

So flexibel die Möglichkeiten bei der Kontrolle beim unmittelbaren Eingeben eines Zeichens in ein Webformular sind – diese Art der Reaktion auf Benutzereingaben ist in der Praxis im Web unüblich.

95 Wie erfolgt die Kontrolle der Benutzereingabe beim Verlassen des Feldes beziehungsweise bei der Entgegennahme eines Rückgabewerts?

Mit dem Eventhandler `onBlur` können Sie eine Benutzereingabe unmittelbar kontrollieren, wenn ein Element den Fokus verliert. Damit haben Sie nicht die zeitnahe Kontrolle pro unmittelbarer 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 muss³³. 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.

33. Die Kontrollfunktion kann damit einfacher aufgebaut werden.

Wie Sie beim Auftreten einer Fehleingabe nun 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 212: Die Funktion zur Kontrolle der eingegebenen Anzahl Zeichen

Sie sehen, dass die Funktion 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` 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.

Realisieren wir nun ein paar mehr praxisorientierte Gegenmaßnahmen in einem vollständigen Beispiel (*formjsmax3.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function maxAnzahl(f,max) {
05   if(f.value.length > max ) {
06     window.status = "Sie haben zu viele Zeichen eingegeben";
07     f.value = "";
08     f.focus();
09   }
10 }
11 //-->
12 </script>
13 <noscript></noscript>
14 <body>
15 <table >
16 <form action="" method="get">
17 <tr>
18 <td>
19 Name
20 </td>
```

Listing 213: Gegenmaßnahmen beim Verlassen eines Eingabefeldes, wenn die erlaubte Anzahl der Zeichen überschritten wird

```
21 <td>
22 <input type="text" name="nn" onBlur="maxAnzahl(this,5)">
23 </td>
24 </tr>
25 <tr>
26 <td>
27 Vorname
28 </td>
29 <td>
30 <input type="text" name="vn" onBlur="maxAnzahl(this,20)">
31 </td>
32 </tr>
33 <tr>
34 <td>
35 <input type="submit" value="OK">
36 </td>
37 <td>
38 </td>
39 </tr>
40 </form>
41 </table>
42 </body>
43 </html>
```

Listing 213: Gegenmaßnahmen beim Verlassen eines Eingabefeldes, wenn die erlaubte Anzahl der Zeichen überschritten wird (Forts.)

Das Formular ist fast identisch mit dem Beispiel vorher. Nur wird bei den Eingabefeldern der Eventhandler `onBlur` zum Aufruf der Funktion `maxAnzahl()` verwendet (Zeile 30 und 35). In der Funktion (definiert von Zeile 4 bis 10) wird die Eingabe in dem Feld geleert, wenn die Anzahl der eingegebenen Zeichen die maximal erlaubte Anzahl überschreitet (Zeile 7), und der Fokus in das Feld zurückgestellt (Zeile 8).

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 401.*

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 <!--
04 function maxAnzahl(f,max) {
05   if(f.length > max ) {
06     window.status = "Sie haben zu viele Zeichen eingegeben";
07   }
08 }
09 //-->
10 </script>
11 <noscript></noscript>
12 <body>
13 <a href="javascript:maxAnzahl(prompt('Bitte geben Sie was ein',''),5)">
14 Eingabe
15 </a>
16 </body>
17 </html>
```

Listing 214: Kontrolle der Anzahl der Zeichen, die ein Nutzer mit `prompt()` eingegeben hat

In Zeile 13 bis 15 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.

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 6 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 per `prompt()` in einer Schleife erneut öffnen. Beachten Sie dann jedoch, dass der Benutzer den `prompt`-Dialog auch abbrechen kann und dann der definierte Wert `null` als Rückgabewert entsteht. Dies sollte zwingend berücksichtigt werden. Absolut verboten sollte 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 vielleicht gar nicht die richtige Eingabe. Die Möglichkeit zum Abbrechen einer Aktion ist ein unabdingbares Muss in der Benutzerführung.

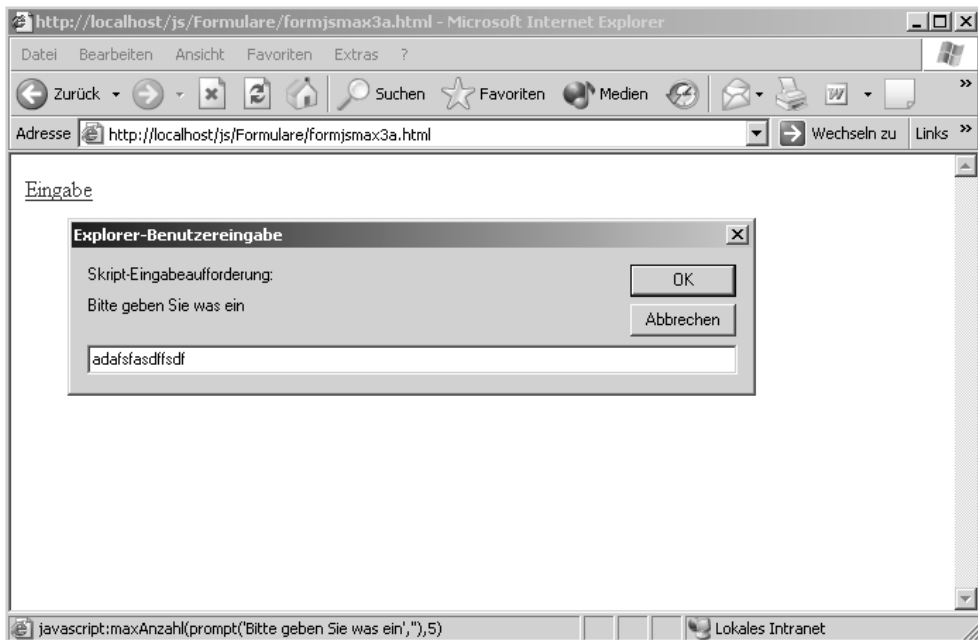


Abbildung 128: Eine Eingabe mit prompt()

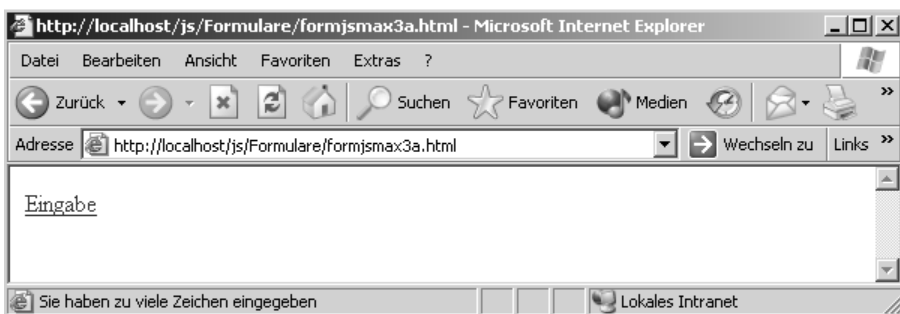


Abbildung 129: In der Statuszeile bekommt der Anwender einen Hinweis, dass zu viele Zeichen eingegeben wurden.

Beispiel (*formjsmax3b.html*):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04 function maxAnzahl(f,max) {
05     if (f != null){
06         if(f.length > max ) {
07             alert("Sie haben zu viele Zeichen eingegeben");
08             maxAnzahl(prompt('Bitte geben Sie was ein',''),max);

```

Listing 215: Kontrolle der Anzahl an Zeichen über prompt() in einer rekursiven Schleife

```

09     }
10   }
11 }
12 //-->
13 </script>
14 <noscript></noscript>
15 <body>
16 <a href="javascript:maxAnzahl(prompt('Bitte geben Sie was ein',''),5)">
17 Eingabe
18 </a>
19 </body>
20 </html>

```

Listing 215: Kontrolle der Anzahl an Zeichen über `prompt()` in einer rekursiven Schleife (Forts.)

In Zeile 16 wird wieder eine Benutzereingabe angefordert und in der Funktion überprüft, ob der Anwender die `Abbrechen`-Schaltfläche betätigt hat (Zeile 5 – `if (f != null)`). Falls nein, bekommt der Anwender in Zeile 7 in einem Dialog eine entsprechende Mitteilung (`alert("Sie haben zu viele Zeichen eingegeben");`). In Zeile 8 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.

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 216: 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.

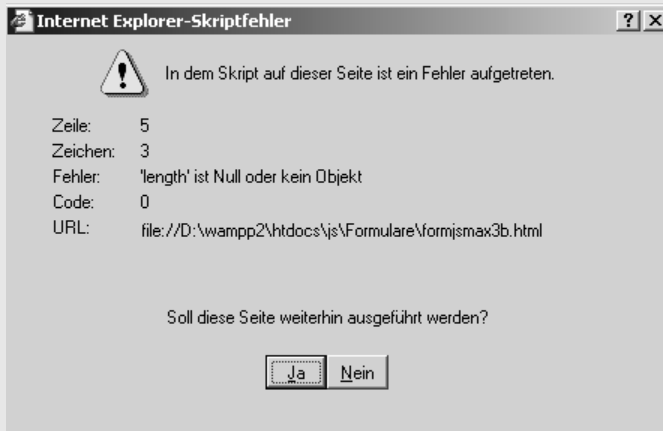


Abbildung 130: Der Anwender hat die Abbrechen-Schaltfläche betätigt.

Achtung

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 217: Rekursiver Endlosaufruf ohne Unterbrechungsmöglichkeit

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.

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()`.

96 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 Verschieken 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 290). 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 muss³⁴. 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 401.). Nachteil ist, dass der Anwender erst beim endgültigen Absenden der Daten beziehungsweise Verlassen des Formulars auf seine Fehleingabe aufmerksam gemacht wird. Ebenfalls steht Ihnen in der kontrollierenden Funktion keine unmittelbare Referenz auf das Formularfeld mit `this` zur Verfügung, da der Eventhandler `onSubmit` beim `<form>`-Tag 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 nicht die Rahmenbedingungen für ein einzelnes Feld als Übergabewert an die Kontrollfunktion notieren.

Wie Sie beim Auftreten einer Fehleingabe nun reagieren, bleibt Ihnen – wie auch 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   }
```

Listing 218: Beim Verlassen des Formulars wird das Feld `nn` kontrolliert.

34. Die Kontrollfunktion kann damit einfacher aufgebaut werden.

```
05  }  
06  }
```

Listing 218: Beim Verlassen des Formulars wird das Feld nn kontrolliert. (Forts.)

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 (*formjsmax4.html*):

```
01 <html>  
02 <script language="JavaScript">  
03 <!--  
04 function maxAnzahl(f) {  
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   if(f.vn.value.length > 10 ) {  
12     alert("Sie haben zu viele Zeichen für den Vornamen eingegeben");  
13     f.nn.value = "";  
14     f.nn.focus();  
15     return false;  
16   }  
17 }  
18 //-->  
19 </script>  
20 <noscript></noscript>  
21 <body>  
22 <table >  
23 <form action="" method="get" onSubmit="return maxAnzahl(this)">  
24 <tr>  
25 <td>  
26 Name  
27 </td>  
28 <td>  
29 <input type="text" name="nn">  
30 </td>  
31 </tr>  
32 <tr>  
33 <td>  
34 Vorname  
35 </td>  
36 <td>
```

Listing 219: Kontrolle von zwei Eingabefeldern beim Verlassen des Formulars

```

37 <input type="text" name="vn">
38 </td>
39 </tr>
40 <tr>
41 <td>
42 <input type="submit" value="OK">
43 </td>
44 <td>
45 </td>
46 </tr>
47 </form>
48 </table>
49 </body>
50 </html>

```

Listing 219: Kontrolle von zwei Eingabefeldern beim Verlassen des Formulars (Forts.)

Das Beispiel verwendet ein Formular (Zeile 23 bis 47) mit zwei einzeiligen Eingabefeldern und einem Submit-Button, die zur optischen Gestaltung in Tabellen aus zwei Spalten angeordnet sind. Bei beiden Eingabefeldern sind explizit keine Eventhandler notiert. Stattdessen finden Sie beim `<form>`-Tag in Zeile 23 den Eventhandler `onSubmit` vor, worüber die Funktion `maxAnzahl()` aufgerufen wird. Diese Funktion (von Zeile 4 bis 17 definiert) erhält als Übergabewert mit `this` das aktuelle Objekt verwiesen. Dies ist eine Referenz auf das aktive Formular. In der Funktion wird dann mit den zwei nacheinander notierten `if`-Abfragen und namentlicher Angabe des zu testenden Eingabefeldes ein Kontrollsystem 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 für das zweite Eingabefeld analog. Im Fall der Rückgabe von `false` bricht der Browser das Versenden des Formulars ab, denn der Aufruf der Funktion `maxAnzahl()` in Zeile 23 erfolgt mit vorangestelltem `return`.

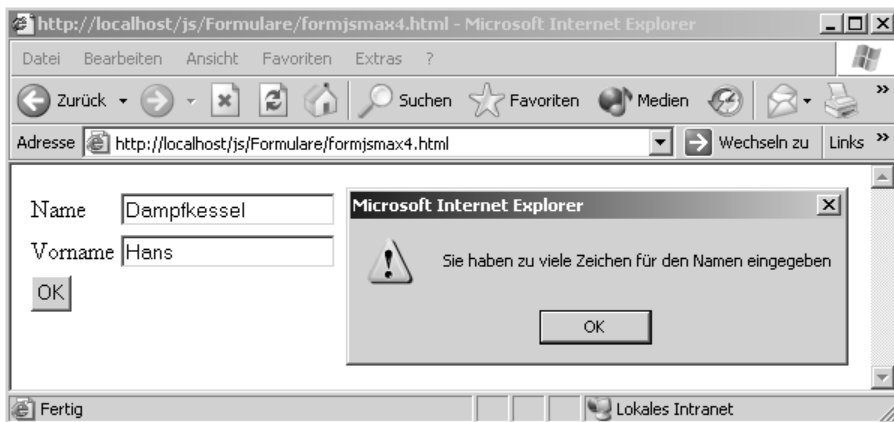


Abbildung 131: Der Name enthält zu viele Zeichen.

Sollte der Name die erlaubte Anzahl der Zeichen nicht überschreiten, muss der Vorname die gleiche Hürde nehmen. 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).



Abbildung 132: Der Name ist in Ordnung, aber der Vorname zu lang.

Tipp

Im Kapitel über 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 sie 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).

Hinweis

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 401.

Hinweis

Sollte es keinen zwingenden Grund geben, sollten Sie bei der Festlegung einer maximal erlaubten Anzahl an Zeichen (als auch grundsätzlich) kein JavaScript einsetzen, wenn Ihnen HTML bereits eine bestimmte Funktionalität bietet. Die potenziellen Probleme reduzieren sich erheblich, je weniger Techniken Sie beim Anwender voraussetzen.

97 Wie kann ich bei freier Texteingabe eine minimale Anzahl einzugebender Zeichen festlegen?

Sehr oft ist es bei Webeingaben 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 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 unsinnig³⁵. Sie kontrollieren die Anzahl der Zeichen sinnvollerweise beim Verlassen des Feldes oder beim Verlassen beziehungsweise Absenden des kompletten Formulars.

Hinweis

Beachten Sie auch das verwandte Rezept »Wie kann ich mit JavaScript bei freier Texteingabe eine maximale Anzahl der einzugebenden Zeichen festlegen?« auf Seite 299, um darüber eine Maximalanzahl an einzugebenden Zeichen festzulegen.

98 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 beim Auftreten einer Fehleingabe nun 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.

35. Zwar können 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.

Beispiel:

```

01 function minAnzahl(f,min) {
02   if(f.value.length < min ) {
03     window.status = "Sie haben zu wenige Zeichen eingegeben";
04   }
05 }

```

Listing 220: Die Funktion zur Kontrolle der eingegebenen Anzahl Zeichen

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.

Abbildung 133: Die Minimalanzahl der Zeichen ist unterschritten.

Hinweis

Das Rezept hier 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 299. 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 <!--
04 function minAnzahl(f,min) {
05   if(f.value.length < min ) {
06     window.status = "Sie haben zu wenige Zeichen eingegeben";
07     f.value = "";
08     f.focus();

```

Listing 221: Gegenmaßnahmen beim Verlassen eines Eingabefeldes, wenn die notwendige Anzahl der Zeichen nicht erreicht wird

```
09  }
10  }
11  //-->
12  </script>
13  <noscript></noscript>
14  <body>
15  <table>
16  <form action="" method="get">
17  <tr>
18  <td>
19  Name
20  </td>
21  <td>
22  <input type="text" name="nn" onBlur="minAnzahl(this,5)">
23  </td>
24  </tr>
25  <tr>
26  <td>
27  Vorname
28  </td>
29  <td>
30  <input type="text" name="vn" onBlur="minAnzahl(this,2)">
31  </td>
32  </tr>
33  <tr>
34  <td>
35  <input type="submit" value="OK">
36  </td>
37  <td>
38  </td>
39  </tr>
40  </form>
41  </table>
42  </body>
43  </html>
```

Listing 221: Gegenmaßnahmen beim Verlassen eines Eingabefeldes, wenn die notwendige Anzahl der Zeichen nicht erreicht wird (Forts.)

Das Beispiel verwendet ein Formular (Zeile 16 bis 40) mit zwei einzeliligen 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 22 und 30). In der Funktion (definiert von Zeile 4 bis 10) wird in der Statuszeile eine Meldung angezeigt (Zeile 6), die Eingabe in dem Feld geleert, wenn beim Verlassen des Feldes die Anzahl der eingegebenen Zeichen die minimal notwendige Anzahl unterschreitet (Zeile 7) und der Fokus in das Feld zurückgestellt (Zeile 8).

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

HTML

Events

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 401.*

99 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 Verschieken 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 290*). 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 sinnvolleren Benutzerführung (*beachten Sie auch die allgemeinen Ausführungen zur Plausibilisierung von Formularen auf Seite 401*) und ist bei einem mehrzeiligen Eingabefeld sogar mit HTML gar nicht zu realisieren. Nachteil ist, dass der Anwender erst beim endgültigen Absenden der Daten beziehungsweise Verlassen des Formulars auf seine Fehleingabe aufmerksam gemacht wird. Ebenfalls steht Ihnen in der kontrollierenden Funktion keine unmittelbare Referenz auf das Formularfeld mit `this` zur Verfügung, da der Eventhandler `onSubmit` beim `<form>`-Tag 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 nicht die Rahmenbedingungen für ein einzelnes Feld 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 beim Auftreten einer Fehleingabe nun 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.

```
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 222: 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 <!--
04 function minAnzahl(f) {
05   if(f.nn.value.length < 8 ) {
06     alert("Sie haben zu wenige Zeichen für den Namen eingegeben");
07     f.nn.value = "";
08     f.nn.focus();
09     return false;
10   }
11   if(f.vn.value.length < 10 ) {
12     alert("Sie haben zu wenige Zeichen für den Vornamen eingegeben");
13     f.nn.value = "";
14     f.nn.focus();
15     return false;
16   }
17 }
18 //-->
19 </script>
20 <noscript></noscript>
21 <body>
22 <table >
23 <form action="" method="get" onSubmit="return minAnzahl(this)">
24 <tr>
25 <td>
26 Name
27 </td>
28 <td>
29 <input type="text" name="nn">
30 </td>
31 </tr>
```

Listing 223: Kontrolle von zwei Eingabefeldern beim Verlassen des Formulars

```
32 <tr>
33 <td>
34 Vorname
35 </td>
36 <td>
37 <input type="text" name="vn">
38 </td>
39 </tr>
40 <tr>
41 <td>
42 <input type="submit" value="OK">
43 </td>
44 <td>
45 </td>
46 </tr>
47 </form>
48 </table>
49 </body>
50 </html>
```

Listing 223: Kontrolle von zwei Eingabefeldern beim Verlassen des Formulars (Forts.)

Das Beispiel verwendet ein Formular (Zeile 23 bis 47) mit zwei einzeiligen Eingabefeldern und einem Submit-Button, die zur optischen Gestaltung in Tabellen aus zwei Spalten angeordnet sind. Beachten Sie, dass bei beiden Eingabefeldern keine Eventhandler notiert sind. Stattdessen finden Sie im Einleitungs-Tag des Webformulars in Zeile 23 den Eventhandler `onSubmit`, worüber beim Verschicken der Formulardaten die Funktion `minAnzahl()` aufgerufen wird. Diese Funktion (von Zeile 4 bis 17 definiert) erhält als Übergabewert mit `this` das aktuelle Objekt verwiesen. 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 Eingabefeldes ein Kontrollsystem aufgebaut. Zuerst wird die Länge des ersten Eingabefeldes kontrolliert (`if(f.nn.value.length < 8 )` in Zeile 5). Ist die geforderte Länge unterschritten, 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. Damit bricht der Browser das Versenden des Formulars ab, denn der Aufruf der Funktion `minAnzahl()` in Zeile 23 erfolgt mit vorangestelltem `return`.

Sollte der Name die geforderte Anzahl der Zeichen erreichen, muss der Vorname die gleiche Hürde nehmen. 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).

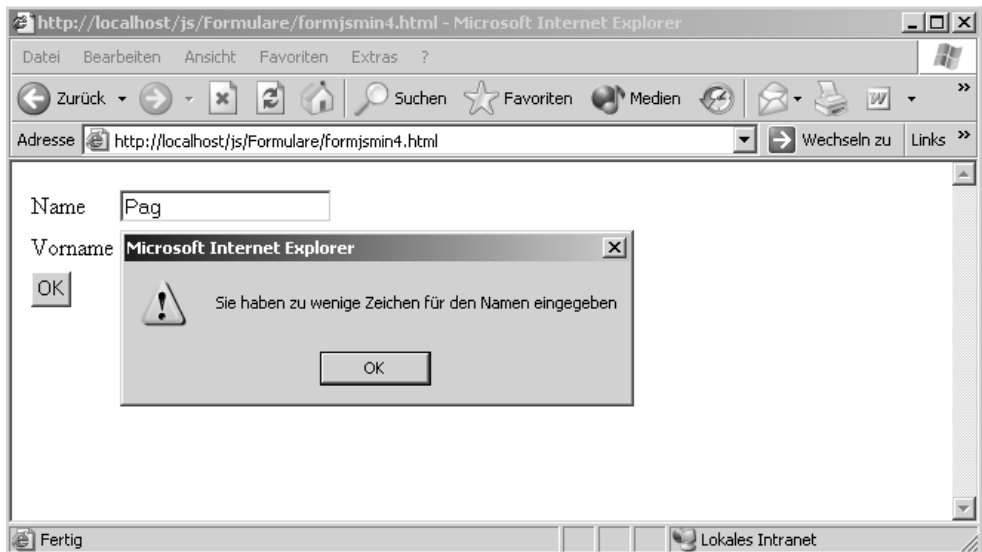


Abbildung 134: Der Name enthält zu wenige Zeichen.



Abbildung 135: Der Name ist in Ordnung, aber der Vorname ist zu kurz.

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 401.

Im Gegensatz zu vielen anderen Festlegungen wie die Angabe einer maximal erlaubten Anzahl an Zeichen ist es bei der Angabe einer Minimalanzahl zwingend notwendig, eine Ergänzungstechnologie wie JavaScript einzusetzen. Rein mit HTML kommen Sie nicht weiter. Natürlich haben Sie dann aber ein Problem, wenn beim Client die Technik nicht vorhanden oder deaktiviert ist.

100 Wie kann ich bei freier Texteingabe eine maximale Anzahl an angezeigten Zeichen festlegen?

Oft soll bei einer freien Texteingabe die Anzahl der angezeigten Zeichen in einem Eingabe-element beschränkt werden. 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 HTML lösen (*siehe das Rezept »Wie kann ich mit HTML bei einem einzeiligen Eingabefeld eine maximale Anzahl an angezeigten Zeichen festlegen?« auf Seite 181*). Wenn Sie den `<input>`-Tag mit dem Parameter `size=[numerischer Wert]` erweitern, legen Sie damit die Anzahl der maximal anzeigbaren Zeichen in dem Feld fest³⁶. Wenn mehr Zeichen eingegeben werden, wird der Inhalt seitwärts gescrollt. 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 anzeigbaren Zeichen verändern wollen, können Sie über die `size`-Eigenschaft eines einzeiligen Formulareingabefeldes (diese korrespondiert mit dem `size`-Attribut in HTML) diesen Wert aus JavaScript heraus verändern (und natürlich auch abfragen). Diese Vorgehensweise ist auch notwendig, wenn Sie bei einem mehrzeiligen Formulareingabefeld eine maximale Anzahl an angezeigten Zeichen festlegen wollen. Aus HTML heraus können Sie dieses Verhalten überhaupt nicht festlegen (*siehe das Rezept »Wie kann ich ein mehrzeiliges Eingabefeld generieren?« auf Seite 198*).

Bei der Entgegennahme von Benutzereingabe 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 <!--
04 function verschoben(f) {
05     window.status= f.size;
06     f.size++;
```

Listing 224: Dynamisches Verändern der size-Eigenschaft eines Eingabefeldes

36. Und das wird von jedem Browser so auch unterstützt.


```
07  }
08  //-->
09  </script>
10  <noscript></noscript>
11  <body>
12  <table >
13  <form action="" method="get">
14  <tr>
15  <td>
16  Name
17  </td>
18  <td>
19  <input type="text" name="nn" size="10" onkeypress="verschoben(this)">
20  </td>
21  </tr>
22  <tr>
23  <td>
24  <input type="submit" value="OK">
25  </td>
26  <td>
27  </td>
28  </tr>
29  </form>
30  </table>
31  </body>
32  </html>
```

Listing 224: Dynamisches Verändern der size-Eigenschaft eines Eingabefeldes (Forts.)

Die Zeilen 13 bis 29 legen das Webformular fest. In dem Formular ist in Zeile 19 ein einzelnes Eingabefeld mit vorgegebener Maximalanzahl anzuzeigender Zeichen (in unserem Beispiel 10) definiert. Der 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 4 bis 7 definiert ist, verwendet die Referenz auf das Formularfeld und gibt darüber die Größe in der Statuszeile des Browsers aus (Zeile 5 – `window.status= f.size;`).



Abbildung 136: Die Größe von size ist in der Statuszeile zu sehen.

Beachten Sie Zeile 6. 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 von der Webseite!).

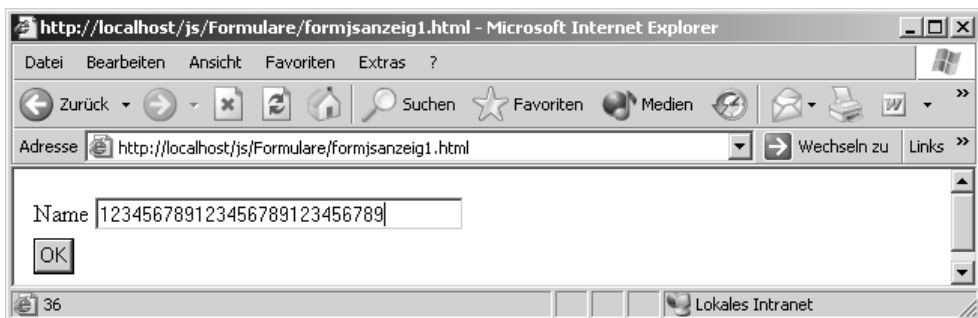


Abbildung 137: Der Wert von `size` beträgt nun 36, und das Eingabefeld ist viel größer als in der Grundeinstellung.

Hinweis

Da die Statuszeile nicht geleert beziehungsweise der Wert von `size` niemals zurückgesetzt wird, ist das Beispiel natürlich nicht praxisfähig. Aber es ist eine interessante Basis, um ein Formular so zu gestalten, dass man das Layout dynamisch an die Anzahl der eingegebenen Zeichen anpassen kann und nicht Zeichen seitwärts scrollen muss, wenn diese die vorgegebene Größe des Feldes überschreiten.

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.



Abbildung 138: In der Statuszeile erkennen Sie das Problem.

Aber auch in neueren Browsern kann es zu Problemen kommen. So können Sie zwar im Konqueror auf die Eigenschaft `size` zugreifen, aber das Setzen der Eigenschaft führt nicht dazu, dass sich das Eingabefeld vergrößert.



Abbildung 139: In der Statuszeile des Konquerors sehen Sie, dass `size` den Wert 65 hat und das Eingabefeld dennoch nicht vergrößert wurde.

Das dynamische Setzen der `size`-Eigenschaft ist nur eine Möglichkeit, um Zusatzfunktionalität gegenüber dem statischen Setzen des `size`-Parameters in HTML zu erreichen. Viel sinnvoller ist wahrscheinlich die Anzeige, ob in einem Eingabefeld von dem Browser 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.

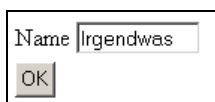


Abbildung 140: Wurde da Text nach links gescrollt oder nicht? Für den Anwender ist nicht erkennbar, wie viele Zeichen eingegeben wurden.

Sie können nun auf verschiedene Weisen 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 bei dem Rezept »Die Kontrolle der Benutzereingabe beim unmittelbaren Eingeben eines Zeichens« auf Seite 300). Schauen wir uns zuerst den einfachen Fall an, in dem dem Anwender einfache Zusatzinformationen in der Statuszeile des Browsers angezeigt werden. Er sieht, ob und wie viele Zeichen sich noch links von den angezeigten Zeichen befinden.

Beispiel (*formjsanzeigt.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function verschoben(f) {
05   if((f.size - f.value.length - 1) < 0)
06     window.status= "Das Feld enthält " +
07       (f.value.length - f.size) +
08       " weitere Zeichen, die nicht angezeigt werden";
09   }
10 function statusLeer() {
11   window.status="";
12 }
13 //-->
14 </script>
15 <noscript></noscript>
16 <body>
17 <table >
18 <form action="" method="get">
19 <tr>
20 <td>
21 Name
22 </td>
23 <td>
24 <input type="text" name="nn" size="10"
25   onkeypress="verschoben(this)"
26   onblur="statusLeer()">
27 </td>
28 </tr>
29 <tr>
30 <td>
31 <input type="submit" value="OK">
32 </td>
33 <td> </td>
34 </tr>
35 </form>
36 </table>
37 </body>
38 </html>
```

Listing 225: Zusatzinformationen in der Statuszeile des Browsers, ob und wie viele Zeichen sich noch links von den angezeigten Zeichen befinden

In dem Beispiel wird ein Webformular mit einem Eingabefeld und einem Submit-Button definiert (die Zeilen 18 bis 35). Die Formularelemente befinden sich aus optischen Gründen in einer Tabellenstruktur aus zwei Spalten. Beim Eingabefeld wird der Eventhandler `onkeypress` verwendet, um in Zeile 25 die Funktion `verschoben()` aufzurufen. Dabei wird mit `this` eine Referenz auf das Eingabefeld übergeben. In der Funktion `verschoben()` (Zeile 4 bis 9) wird kontrolliert, ob der Wert von `size` von der Anzahl der eingegebenen Zeichen überschritten wird (in der Zeile 5 – `if((f.size - f.value.length - 1) < 0)`)³⁷. Falls dem so ist, wird eine Meldung in der Statuszeile

des Browsers angezeigt (Zeile 6 bis 8 – `window.status= "Das Feld enthält " + (f.value.length - f.size) + " weitere Zeichen, die nicht angezeigt werden";`).

In Zeile 25 wird mit dem Eventhandler `onBlur` noch die Funktion `statusLeer()` aufgerufen. Diese leert die Statuszeile des Browsers beim Verlassen von dem Eingabefeld (Zeile 10 bis 12).



Abbildung 141: In der Statuszeile wird dem Anwender angezeigt, dass es Zeichen gibt, die nicht angezeigt werden.

Hinweis

Das letzte Beispiel ist nur begrenzt praxisfähig. Sie sollten beachten, dass die Anzahl der anzeigbaren Zeichen nicht exakt mit dem Wert von `size` übereinstimmt. Der Browser wird aufgrund 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

Die Statuszeile in einem Browser ist die wahrscheinlich am meisten ignorierte Stelle im Browser. Ist es scheinbar manchen Anwendern schon unmöglich, ein Dialogfenster mit einem kurzen Satz richtig wahrzunehmen³⁷, wird die unscheinbare Statuszeile nur sehr sel-

37. 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.

38. Mitarbeiter von Hotlines werden das Problem kennen – dem Anwender blockiert ein riesiges Dialogfenster den gesamten Rechner. Darin steht nur groß und deutlich der kurze Satz »Klicken Sie auf OK«, und der Anwender wird auf Teufel komm raus nicht wissen, was er tun soll ;-).

ten überhaupt beachtet. Es macht bei einer Webseite viel Sinn, eine Information über verschobene Zeichen in einem Eingabefeld über zusätzliche Details kenntlich zu machen, etwa ein Symbol wie einen Pfeil, der neben dem Eingabefeld angezeigt wird, wenn Zeichen gescrollt wurden. Betrachten Sie das nachfolgende Listing.

Beispiel (*formjsanzeigt3.html*):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04 function verschoben(f,i) {
05   if((f.size - f.value.length - 1) < 0){
06     window.status= "Das Feld enthält " +
07       (f.value.length - f.size) +
08       " weitere Zeichen, die nicht angezeigt werden";
09     window.document.images[i].src="links.gif";
10   }
11   else {
12     window.status="";
13     window.document.images[i].src="leer.gif";
14   }
15 }
16 function statusLeer(i) {
17   window.status="";
18   window.document.images[i].src="leer.gif";
19 }
20 function anfang(i) {
21   window.document.f.elements[((i + 1 )% 2)].focus();
22   window.document.f.elements[i].focus();
23 }
24 //-->
25 </script>
26 <noscript></noscript>
27 <body>
28 <table >
29 <form action="" method="get" name="f">
30 <tr>
31 <td>
32 Name
33 </td>
34 <td>
35 
36 </td>
37 <td>
38 <input type="text" name="nn" size="10"
39   onFocus="verschoben(this,0)"
40   onKeyPress="verschoben(this,0)"
41   onBlur="statusLeer(0)">
42 </td>
43 </tr>

```

Listing 226: Erweiterte Informationen, ob Zeichen gescrollt wurden

```

44 <tr>
45 <td>
46 Vorname
47 </td>
48 <td>
49 
50 </td>
51 <td>
52 <input type="text" name="vn" size="10"
53   onFocus="verschoben(this,1)"
54   onKeyPress="verschoben(this,1)"
55   onBlur="statusLeer(1)">
56 </td>
57 </tr>
58 <tr>
59 <td>
60 <input type="submit" value="OK" name="sub">
61 </td>
62 <td>
63 </td>
64 </tr>
65 </form>
66 </table>
67 </body>
68 </html>

```

Listing 226: Erweiterte Informationen, ob Zeichen gescrollt wurden (Forts.)

In dem Beispiel wird erneut ein Webformular aufgebaut (die Zeilen 29 bis 65). 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 Eingabefeld³⁹ (Zeile 31 bis 33 – `<td>Name</td>` und Zeile 45 bis 47 – `<td>Vorname</td>`), welches sich in Spalte 3 befindet (Zeile 37 bis 42 und Zeile 52 bis 56). In der zweiten Spalte wird jedoch hier jeweils eine Grafik geladen (Zeile 34 bis 36 – `<td></td>` und Zeile 48 bis 50 – `<td></td>`). Die Grafik `leer.gif` ist – wie der Name schon andeutet – ein leerer Platzhalter, der in der entsprechenden Situation ersetzt wird.

In den Tags der beiden Eingabefelder wird sowohl mit `onKeyPress` als auch `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 von diesem Listing 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

39. Wie auch bei den meisten anderen Beispielen in dem Kapitel, bei denen eine Tabelle das Formular gestaltet.

dem die Grafik einsortiert wird. Dieses Array ist über `window.document.images` zugänglich. In dem Beispiel befinden sich zwei Stellen mit Grafiken (in 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]` anzusprechen ist, bekommt `verschoben()` in der ersten Tabellenzeile als zweiten Übergabewert den Wert `0` übergeben. Entsprechend verhält es sich bei der zweiten Zeile.

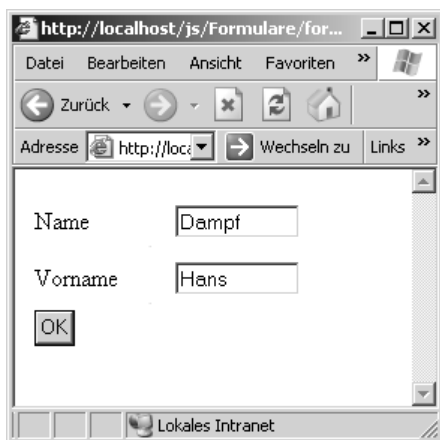


Abbildung 142: Das Formular, solange keine Zeichen in den Eingabefeldern unsichtbar sind

In der Funktion `verschoben()` (Zeile 4 bis 15) wird kontrolliert, ob der Wert von `size` von der Anzahl der eingegebenen Zeichen überschritten wird (in der Zeile 5 – `if((f.size - f.value.length - 1) < 0)`)⁴⁰. Falls dem so ist, wird wie im Beispiel zuvor eine Meldung in der Statuszeile des Browsers angezeigt (Zeile 6 bis 8). Zusätzlich 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 ein neuer URL einer anderen Grafik zugewiesen wird (Zeile 9 – `window.document.images[i].src="links.gif";`). Die Grafik `links.gif` ist im konkreten Fall ein Pfeil, der nach links zeigt.



Abbildung 143: Wenn Zeichen verdeckt sind, wird links neben dem Eingabefeld ein Pfeil angezeigt.

Der `else`-Zweig in der Funktion `verschoben()` (Zeile 11 bis 14) leert die Statuszeile (Zeile 12 – `window.status=""`;) und setzt das Bild wieder auf `leer.gif` (Zeile 13 – `window.document.images[i].src="leer.gif";`).

40. Die `-1` ergibt sich, weil die Funktion mit dem Eventhandler `keypress` angerufen 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.

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 Statuszeile des Browsers geleert, wenn ein Anwender den Fokus von einem Eingabefeld entfernt. Aber das ist nicht alles. Die aufgerufene Methode `statusLeer()` 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 in Zeile 18 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.

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 von einem Feld geben wollen.



Abbildung 144: Das Formular mit Anzeige, dass im Feld für den Namen Zeichen verdeckt werden

Bleibt noch das Ereignis `onClick` auf den Grafiken zu besprechen (`<td></td>` und `<td></td>`). Dabei wird die Funktion `anfang()` aufgerufen und als Übergabewert der Index des Formularelementes übergeben. Diese Funktion, die in den Zeilen 20 bis 23 definiert ist, setzt den Fokus auf das übergebene 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. In Zeile 21 wird zuerst der Fokus auf ein anderes Eingabefeld in dem Formular gesetzt (`window.document.f.elements[(((i + 1) % 2)].focus();`). Das Moduloverfahren stellt sicher, dass immer ein anderes Formularfeld als das aktuelle Feld den Fokus erhält. In Zeile

22 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 anderen Browsern wird nur der Fokus in das Feld gestellt.

Hinweis

Sollte es keinen zwingenden Grund geben, sollten Sie bei der Festlegung einer maximal anzuzeigenden Anzahl an Zeichen (als auch grundsätzlich) kein JavaScript einsetzen, wenn Ihnen HTML bereits eine bestimmte Funktionalität bietet. Die potenziellen Probleme reduzieren sich erheblich, je weniger Techniken Sie beim Anwender voraussetzen. Aber es spricht nichts dagegen, dem Besucher optionale Zusatzinformationen zu präsentieren.

101 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 einzeiligen Eingabefelds als auch eines mehrzeiligen Eingabefelds hervorragend mit HTML lösen (*siehe das Rezept »Wie kann ich rein mit HTML bei einem einzeiligen Eingabefeld einen Vorgabewert festlegen?« auf Seite 182 sowie das Rezept »Wie kann ich ein mehrzeiliges Eingabefeld generieren?« auf Seite 198*). Wenn Sie diese Aufgabe jedoch mit JavaScript erledigen wollen, erhalten Sie auf Wunsch ergänzende Möglichkeiten. Mit der Eigenschaft `value` eines Eingabefeldes (einzeilig als auch mehrzeilig⁴¹) haben Sie Zugang zum korrespondierenden Parameter `value="[Wert]"` vom `<input>`-Tag unter HTML beziehungsweise dem Inhalt des `<textarea>`-Containers. Sie können also aufgrund 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 <!--
04 function vorgabe() {
05   var zufall = Math.round(Math.random() * 1);
06   var text = new Array();
07   text[0] = "In der Ruhe liegt die Langeweile";
08   text[1] = "Kommt Zeit, kommt die Schwiegermutter";
09   window.document.f.a.value = text[zufall];
10 }
11 //-->
12 </script>
13 <noscript></noscript>
```

Listing 227: Ein einzeiliges Listenfeld mit Vorgabewert

41. Beim mehrzeiligen Eingabefeld ist bemerkenswert, dass es dort kein HTML-Attribut `value` gibt.

```

14 <body onload="vorgabe()">
15 <form action="" method="get" name="f">
16 <input type="Text" name="a">
17 <br>
18 <input type="submit" value="OK">
19 </form>
20 </body>
21 </html>

```

Listing 227: Ein einzeliges Listenfeld mit Vorgabewert (Forts.)

Das Beispiel verwendet ein einfaches Formular mit einem einzeligen 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 14). Die dort angegebene Funktion `vorgabe()` wird beim Laden der Webseite ausgeführt. *Beachten Sie in diesem Zusammenhang die Warnung auf Seite 209.* Das dort beschriebene Problem wird mit der Verwendung des Eventhandlers umgangen.

In den Zeilen 4 bis 10 wird mit einem Zufallsmechanismus (in Zeile 5 `var zufall = Math.round(Math.random() * 1);`) eine Zahl berechnet, die in Zeile 9 als Index für die Auswahl eines Textes verwendet, der als Vorbelegungswert dienen soll (`window.document.f.a.value = text[zufall];`). In Zeile 6 wird dazu ein Array angelegt (`var text = new Array();`), und in den Zeilen 7 und 8 werden zwei Einträge erzeugt. Durch den Zufallsprozess wird zufällig ein Wert als Vorgabewert in dem Eingabefeld angezeigt.

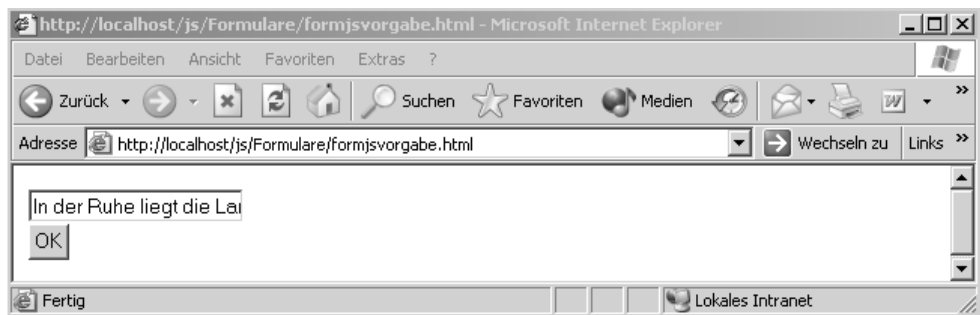


Abbildung 145: Das Eingabefeld wird zufällig vorbelegt.

Hinweis

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 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.

102 Wie kann ich bei freier Texteingabe gewährleisten, dass nur Dezimalkommazahlen gestattet sind?

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 einzeligen Eingabefelds diese Aufgabe mit HTML lösen, aber kein einziger Browser unterstützt die offiziellen Vorgaben dazu (*siehe das Rezept »Wie kann ich rein mit HTML ein einzeliges Eingabefeld realisieren, in dem nur die Eingabe von Dezimalkommazahlen gestattet ist?« auf Seite 193*). Bei einem mehrzeiligen Eingabefeld bietet HTML selbst von der Theorie her keine Möglichkeiten dazu. Sie sind also auf JavaScript angewiesen. Dabei können Sie 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 Sicht⁴² 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 von dem Anwender 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 ein 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):

42. Die ganzen Zahlen sind ja ein Teilmenge der Dezimalzahlen.

```
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 228: Kontrolle, ob die Eingabe eine Zahl ungleich 0 war

Der Funktion wird als Übergabewert das zu kontrollierende Eingabefeld übergeben. Das macht man am sinnvollsten, indem man die Funktion mit einem Eventhandler beim 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 verwendet die Funktion `isNaN()`, um einen Boole'schen Wert zu erhalten. Dies können Sie oben natürlich auch machen und 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 es 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 Eintreffen 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 (*formsdez1.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function dez(a) {
05   window.status = a.value / 1;
06   if(a.value / 1) {
07     alert("OK");
08   }
09   else {
10     alert("Nicht OK");
11     a.value="";
12     a.focus();
13   }
14 }
```

Listing 229: Kontrolle, ob der Wert in einem Eingabefeld numerisch ist

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

```
13 }  
14 }  
15 //-->  
16 </script>  
17 <noscript></noscript>  
18 <body>  
19 <form action="" method="get" name="f">  
20   <input type="text" onBlur="dez(this)">  
21   <br>  
22   <input type="submit" value="OK">  
23 </form>  
24 </body>  
25 </html>
```

Listing 229: Kontrolle, ob der Wert in einem Eingabefeld numerisch ist (Forts.)

In den Zeilen 19 bis 23 wird ein Webformular mit einem Eingabefeld sowie einer Submit-Schaltfläche definiert. In dem Eingabefeld in Zeile 20 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 tatsächlich die Division ergeben hat. Das sollte dem Endanwender natürlich nicht gezeigt werden.

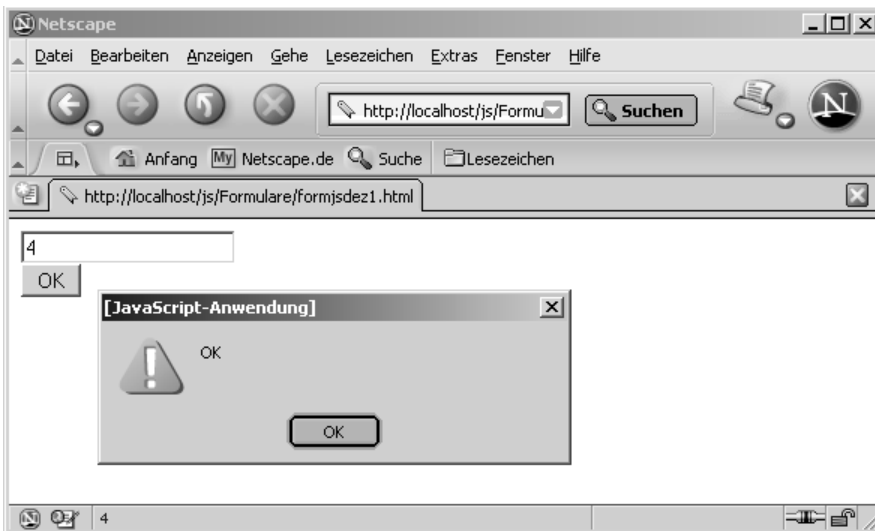


Abbildung 146: Die Division durch den Wert 1 hat funktioniert.

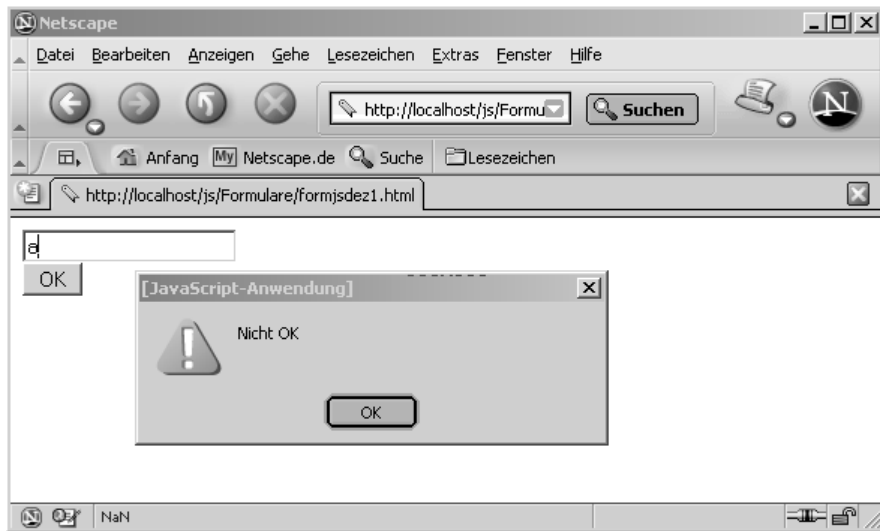


Abbildung 147: Bei Eingabe von einem nichtnumerischen Zeichen entsteht NaN (siehe in der Statuszeile).

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 230: 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)` als auch jede andere gültige Division den Wert `false`. Das Ausrufezeichen dreht diesen Boole'schen Wert in `true` um.

Die dritte Variante von dem Rezept 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 231: Kontrolle, ob die Eingabe eine Zahl war

Die Veränderung bewirkt, dass alle Zahlen außer 0 eingegeben werden können. Bei 0 oder einem nichtnumerischen Ausdruck – der liefert NaN – wird der else-Zweig ausgelöst.

Hinweis

Wenn Sie explizit sicherstellen wollen, dass eine numerische Eingabe als Dezimalzahl behandelt wird, können Sie bei der ersten Variante von dem Rezept 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 wird. Da JavaScript für Zahlen jedoch nur den gemeinsamen Datentyp `Number` verwendet, wird JavaScript immer eine Optimierung vornehmen. Wenn es also möglich ist, auf den Nachkommaanteil zu verzichten (etwa beim Wert 3.0), wird der Nachkommateil nicht verwendet. Beachten Sie aber das nachfolgende Rezept mit einem anderen Trick.

Sie können allerdings einen Nachkommaanteil auf jeden Fall bewahren, wenn Sie einen String als Datentyp verwenden. Dazu nehmen Sie die Benutzereingabe, testen Sie auf numerisch und dann, ob es eine Zahl mit Nachkommaanteil 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 232: Sicherstellen, dass nach Aufruf der Funktion ein Nachkommaanteil 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. 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 kann 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()` `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. Beachten Sie das nachfolgende Konstrukt:

```
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 233: 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 aufgrund 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 auszuhebeln.

Beispiel (*formsdez4.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function dez(a) {
05     var zahl = ""+a.value * 1;
06     if(zahl) {
07         if(parseInt(zahl) == parseFloat(zahl))
08             zahl = zahl + ".00";
09         return zahl;
10     }
11     else {
12         return "";
13     }
14 }
15 //-->
16 </script>
17 <noscript></noscript>
18 <body>
19 <form action="" method="get" name="f">
20     <input type="text" onBlur=
21         "alert(
22 'Typ des Rückgabewertes: ' + typeof(dez(this)) + '\nWert: ' + dez(this))">
23     <br>
24     <input type="submit" value="OK">
25 </form>
26 </body>
27 </html>
```

Listing 234: Sicherstellung, dass auch eine Ganzzahl in der Folge mit einem Nachkommaanteil geführt wird

Das Formularfeld in Zeile 20 bis 22 ruft die Funktion `dez()` in diesem Beispiel so auf, 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.

Wenn ein Anwender eine Dezimalzahl eingibt, bleibt der Wert der Variablen `zahl` der ursprünglich eingegebene Wert.

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 sonst eine Reaktion durchführen, die für den Anwender der Funktion eine sinnvolle Weiterverwendung gestattet.



Abbildung 148: Eingabe einer Ganzzahl – dennoch hat der Rückgabewert einen Nachkommaanteil .00.



Abbildung 149: Eine Dezimalzahl als Eingabe führt zu einer unveränderten Rückgabe.



Abbildung 150: Eine Texteingabe liefert einen Leerstring.

Tipp

Im Kapitel über 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 sie 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.

103 Wie kann ich bei freier Texteingabe gewährleisten, dass nur Ganzzahlen gestattet sind?

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 einzeiligen Eingabefelds diese Aufgabe mit HTML lösen, aber kein einziger Browser unterstützt die offiziellen Vorgaben dazu (*siehe das Rezept »Wie kann ich rein mit HTML ein einzeiliges Eingabefeld realisieren, in dem nur die Eingabe von ganzen Zahlen gestattet ist?« auf Seite 193*). Bei einem mehrzeiligen Eingabefeld bietet HTML selbst von der Theorie her keine Möglichkeiten dazu. Sie sind also auf JavaScript angewiesen. Dabei können Sie 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 336. 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 235: 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 aufgrund 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 damit 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 `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></noscript>
17 <body>
18 <form action="" method="get" name="f">
19     <input type="text" onBlur="alert(int(this))">
20     <br>
21     <input type="submit" value="OK">
22 </form>
23 </body>
24 </html>

```

Listing 236: Das Beispiel *formjsint.html* gibt entweder eine ganze Zahl oder undefined aus.

In den Zeilen 18 bis 22 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.

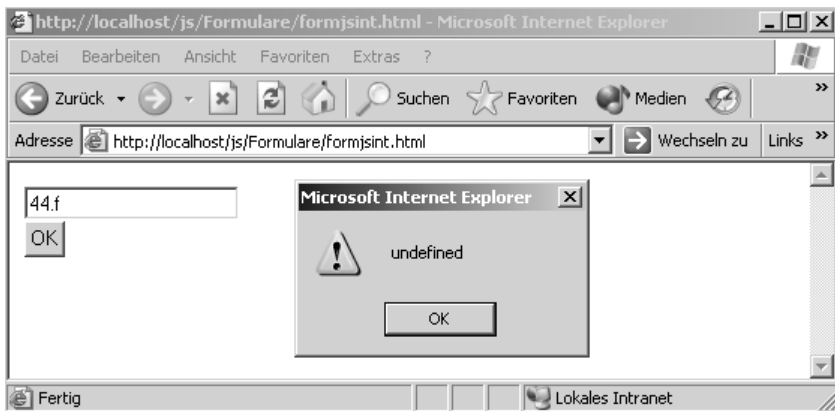


Abbildung 151: Die Anwendereingabe war keine Ganzzahl.



Abbildung 152: Der Anwender hat eine Ganzzahl eingegeben.

Tipp

Im Kapitel über 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 sie 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.

104 Wie kann ich bei freier Texteingabe sicherstellen, dass dort nur Werte kleiner als ein vorgegebener Grenzwert gestattet sind?

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 einzeiligen Eingabefelds diese Aufgabe mit HTML lösen, aber kein einziger Browser unterstützt die offiziellen Vorgaben dazu (*siehe das Rezept »Wie kann ich rein mit HTML ein einzeiliges Eingabefeld realisieren, in dem nur Werte kleiner als ein vorgegebener Grenzwert gestattet sind?« auf Seite 194*). Bei einem mehrzeiligen Eingabefeld bietet HTML selbst von der Theorie her keine Möglichkeiten dazu. Sie sind also auf JavaScript angewiesen. Dabei können Sie die nachfolgende Vorgehensweise auch auf den Rückgabewert der `prompt()`-Methode anwenden.

Über die `value`-Eigenschaft von einem einzeiligen 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 lose Typisierung. Sie brauchen keinerlei Sicherstellung einer numerischen Eingabe (auch nicht, ob es eine Ganzzahl 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.

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.

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 {
```

Listing 237: Die schematische Kontrolle, ob ein Wert in einem Eingabefeld kleiner als ein vorgegebener Maximalwert ist

```

    // Bedingung nicht erfüllt - tue das, was dann passieren soll
  }
}

```

Listing 237: Die schematische Kontrolle, ob ein Wert in einem Eingabefeld kleiner als ein vorgegebener Maximalwert ist (Forts.)

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 353.

Der Funktion wird als erster Übergabewert das zu kontrollierende Eingabefeld übergeben. Das macht man am sinnvollsten, indem man die Funktion mit einem Eventhandler beim HTML-Tag des zu kontrollierenden Eingabefeldes aufruft und mit `this` eine Referenz darauf übergibt. Der zweite Übergabewert ist der Grenzwert, der nicht überschritten werden soll.

Wie Sie nun konkret bei Eintreffen 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></noscript>
17 <body>
18 <form action="" method="get" name="f">
19   <input type="text" onBlur="kontrolliereMax(this,7)">
20   <br>
21   <input type="submit" value="OK">
22 </form>
23 </body>
24 </html>

```

Listing 238: 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).

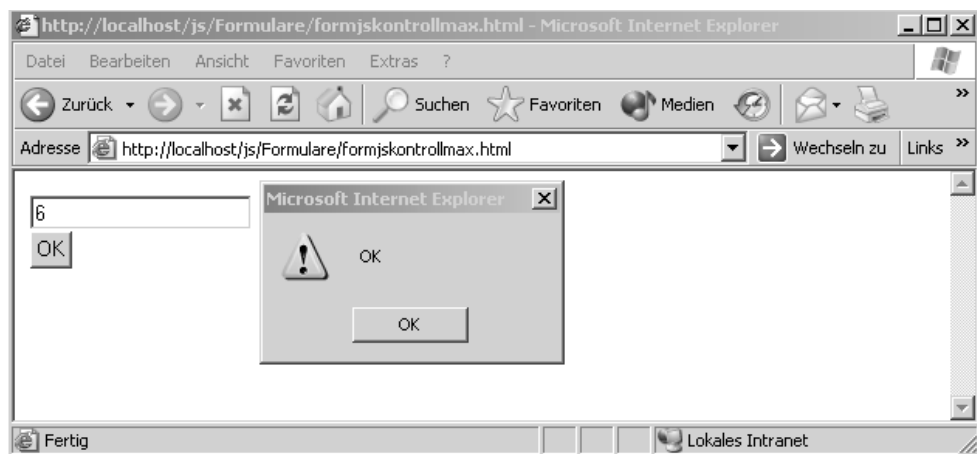


Abbildung 153: Die Eingabe bleibt unter dem vorgegebenen Grenzwert.

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 von dem 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 401.

In dem kleinen, unscheinbaren Rezept hier steckt aber noch viel mehr, als es im ersten Moment erscheint! 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 Programmieretechniken nicht selbstverständlich.

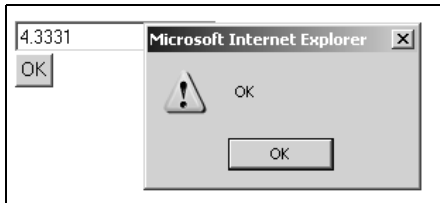


Abbildung 154: Keine Probleme mit Gleitkommazahlen

Und was passiert denn, wenn Sie als Grenzwert für eine Benutzereingabe keine ganze Zahl vorgeben, sondern eine Gleitkommazahl vorgeben? Tauschen Sie in dem Beispiel einmal Zeile 19 durch `<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 nichtnumerische Eingabe eingibt? Sie passiert die Bedingung in der `if`-Anweisung nicht. Mit anderen Worten – der Test liefert `false`.

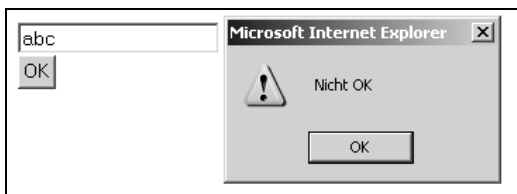


Abbildung 155: Texteingaben werden nicht akzeptiert.

Das gilt auch, wenn die Eingabe mit einer Zahl beginnt und dann nichtnumerische Zeichen folgen! Selbst dann, wenn der numerische Wert alleine der Bedingung genügen würde.

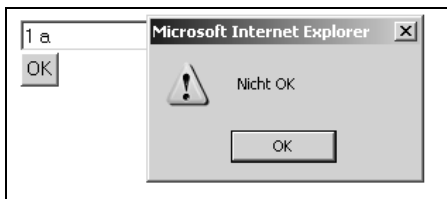


Abbildung 156: Auch mit Zahlen beginnende Eingaben werden nicht akzeptiert, wenn dann noch nichtnumerische Zeichen folgen.

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 von einem vorgegebenen Testwert 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 durch `<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 157: Der Test auf die Größe von nichtnumerischen Eingaben ist mit entsprechendem Testwert möglich.

Bei Eingaben, die mit D oder einem folgenden Buchstaben des Alphabetes beginnen, liefert die if-Bedingung `false`.

Was ist aber, wenn Sie in dem Eingabefeld nun mehr wie ein Zeichen eingeben? Auch dann liefert die if-Bedingung `true`, solange die Kodierung von dem ersten (!) Zeichen kleiner als die Kodierung von dem Testwert ist. Auch wenn dieser nur aus einem Zeichen besteht!



Abbildung 158: Solange die Kodierung von dem ersten Zeichen kleiner als die Kodierung von dem Testwert ist, ist der Bedingung Genüge getan.

Tip

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 159: Kleinbuchstaben passieren die Prüfung des Beispiels in der aktuellen Form nicht.

Tipp

Wenn Sie Groß- und Kleinschreibung nicht unterscheiden wollen, 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 gewandelt und bei `toLowerCase()` in Kleinbuchstaben. Sie sollten die Methode einfach auf die Benutzereingabe anwenden, etwa so:

```
a.value.toUpperCase()
```

Listing 239: 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!

Hinweis

Hintergrund des Verhaltens bezüglich 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 die Werte der Kodierung von Zahlen kleiner als die Werte der Kodierung von Buchstaben sind sowie die Werte der Kodierung von Großbuchstaben kleiner 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 240: Die Kodierung aller Großbuchstaben ist kleiner als die der Kleinbuchstaben.

Tipp

Im Kapitel über 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 sie 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.

Sollten Sie dieses extrem flexible Verhalten von dem Rezept 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 344) 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 336). 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 299 beziehungsweise »Wie kann ich mit JavaScript bei freier Texteingabe eine minimale Anzahl der einzugebenden Zeichen festlegen?« auf Seite 317). 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.

105 Wie kann ich sicherstellen, dass dort nur Werte größer als ein vorgegebener Grenzwert gestattet sind?

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 einzeiligen Eingabefelds diese Aufgabe mit HTML lösen, aber kein einziger Browser unterstützt die offiziellen Vorgaben dazu (siehe das Rezept »Wie kann ich rein mit HTML ein einzeiliges Eingabefeld realisieren, in dem nur Werte größer als ein vorgegebener Grenzwert gestattet sind?« auf Seite 194). Bei einem mehrzeiligen Eingabefeld bietet HTML selbst von der Theorie her keine Möglichkeiten dazu. Sie sind also auf JavaScript angewiesen. Dabei können Sie die nachfolgende Vorgehensweise auch auf den Rückgabewert der `prompt()`-Methode anwenden.

Über die `value`-Eigenschaft von einem einzeiligen 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. Diese ist wirklich sehr einfach und hat es dennoch in sich:

```
function kontrolliereMin(a, min) {
```

Listing 241: Die schematische Kontrolle, ob ein Wert in einem Eingabefeld größer als ein vorgegebener Minimalwert ist

```
if(a.value > min) {  
    // Bedingung erfüllt - tue das, was dann passieren soll  
}  
else {  
    // Bedingung nicht erfüllt - tue das, was dann passieren soll  
}  
}
```

Listing 241: Die schematische Kontrolle, ob ein Wert in einem Eingabefeld größer als ein vorgegebener Minimalwert ist (Forts.)

Der Funktion wird als erster Übergabewert das zu kontrollierende Eingabefeld übergeben. Das macht man am sinnvollsten, indem man die Funktion mit einem Eventhandler beim HTML-Tag des zu kontrollierenden Eingabefeldes aufruft und mit `this` eine Referenz darauf übergibt. Der zweite Übergabewert ist der Grenzwert, der nicht unterschritten werden soll.

Wie Sie nun konkret bei Eintreffen 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></noscript>  
17 <body>  
18 <form action="" method="get" 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 242: Ein Test, ob die Eingaben in einem Eingabefeld einen Minimalwert unterschreiten

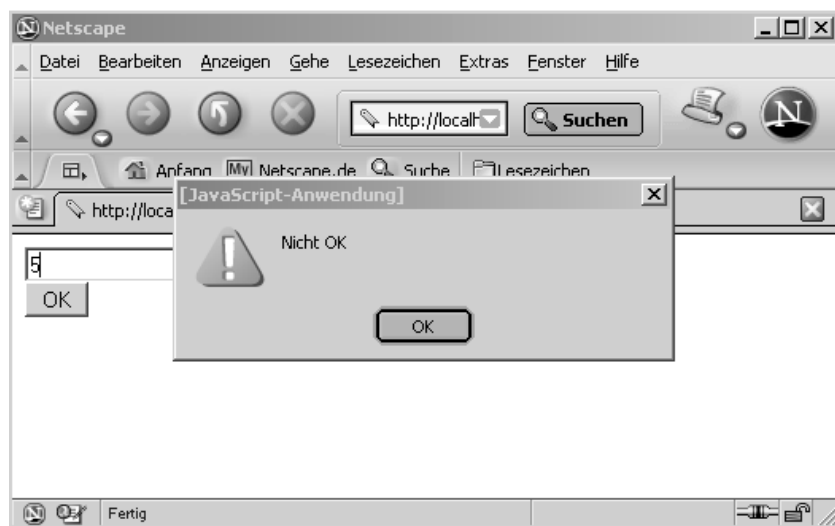


Abbildung 160: Der Minimalwert wurde unterschritten.

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 347. Die Ausführungen gelten vollkommen analog für dieses Rezept. Nur müssen Sie gegebenenfalls mit größer und kleiner aufpassen. Das sollten Sie insbesondere bei den Werten der Kodierung von Groß- und Kleinbuchstaben sowie Zahlen im Vergleich zu Buchstaben beachten.

106 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*, ob zum Beispiel ein Name eingegeben oder das Feld vergessen wurde. Sie können ein Pflichtfeld in einem Formular leicht erzwingen, indem Sie einfach darauf prüfen, ob die Eigenschaft `value` 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:

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

```
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 243: Schematische Darstellung für den Test eines Pflichtfeldes

Das nachfolgende Listing zeigt eine Anwendung von so einer 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" action="" method=POST>
15       Name (<font color="#FF0000">*</font>):
16       <input name="eins" onBlur="testPflichtFeld(this)">
17       <br>
18       <input type="submit" value="Ok">
19   </form>
20 </body>
21 </html>
```

Listing 244: Überprüfung auf ein Pflichtfeld

Das Beispiel besteht aus einem einfachen Formular (Zeile 14 bis 19) 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)⁴³.

43. Der Rückgabewert wird in diesem Beispiel nicht weiter ausgewertet.

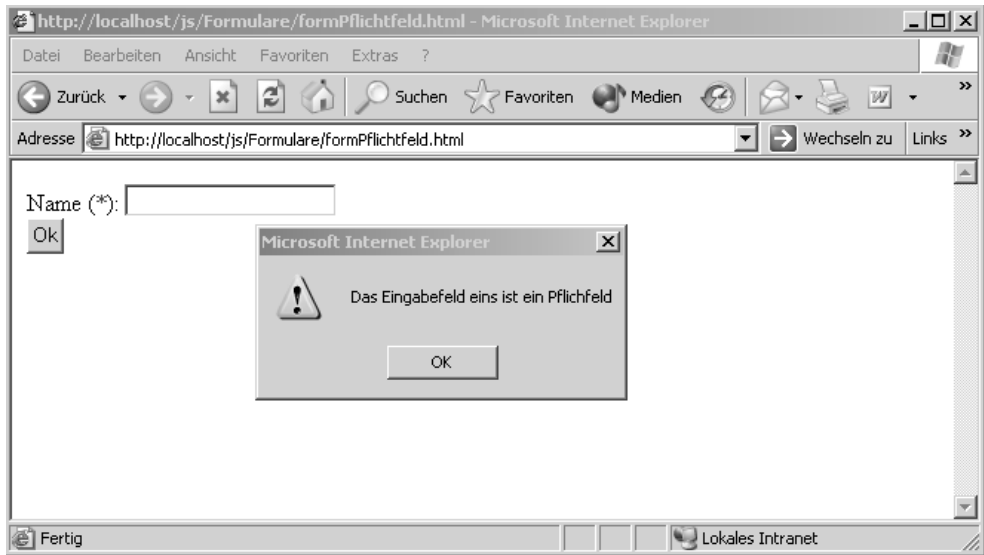


Abbildung 161: In dem Eingabefeld wurde keine Eingabe vorgenommen.

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 401.*

Tipp

In einem Webformular werden Pflichtfelder oft mit einem roten Stern oder einer anderen 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 162: Eine oft zu findende Kennzeichnung als Pflichtfeld ist ein roter Stern.

107 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 meist nicht. So muss beispielsweise eine E-Mail-Adresse oder der 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 auf die `split()`-Methode. Mit `split([Separator][, Limit])` trennen Sie einen String in ein Array mit Teilstrings auf, indem an dem angegebenen Separator geteilt wird. 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. 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 anlog 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]` anzuweisen, 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

Im Kapitel über Stringmanipulationen besprechen wir den Umgang mit so genannten *regulären Ausdrücken*.

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 245: 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`, und auf den prüft Zeile 2 ab.

108 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-Adresse. Zwar können Sie nicht testen, ob es eine eingegebene E-Mail-Adresse wirklich gibt⁴⁴, aber Sie können

44. Das ist jetzt keine ausschließliche Einschränkung von JavaScript, wobei hier aber auch grundsätzlich die Möglichkeiten für so eine Überprüfung vom Client aus fehlen. Ein solcher Test ist generell nicht zuverlässig möglich, mit keiner Technologie. Weder auf Client- noch auf Serverseite. Die einzig praktikable Möglichkeit zur echten Überprüfung einer E-Mail-Adresse besteht, wenn Sie unmittelbaren Zugang zu dem E-Mail-Server haben, auf dem die eingegebene 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 gibt nicht jeder E-Mail-Server nach dem Empfang einer E-Mail oder einer Testanfrage eine Bestätigung heraus. 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. 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 (wenn er etwa intelligent abgesichert ist) oder der Server einfach nur temporär nicht erreichbar ist. Der gewichtigste Grund ist aber, dass TCP/IP ein verbindungsloses Protokoll ist und die Verbindung zu einem Client nicht permanent gehalten werden kann, um im gesamten WWW nach der Erfüllung von Plausibilitäten zu suchen. 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 dauert, ist er meist bereits weitergesurft und verlässt sich darauf, dass die Formulardaten korrekt entgegengenommen wurden.

zumindest die grundsätzliche Struktur plausibilisieren. Dabei können Sie mehrere Stufen auswählen, wie plausibel die E-Mail-Adresse wirklich sein soll. Als Eingabe ist sowohl ein einzeliges als auch mehrzeiliges 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. Die Rückgabe einer `prompt()`-Methode kann dagegen gut genutzt werden.

Tip

Im Kapitel über Stringmanipulationen besprechen wir den Umgang mit so genannten *regulären Ausdrücken*. Im Rezept »Wie kann ich mit regulären Ausdrücken in Strings suchen und sie 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 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" action="" method="POST">
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 246: Ein einfacher Test der E-Mail-Struktur auf ein enthaltenes @-Zeichen

Hinweis

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 401.

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 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)⁴⁵.

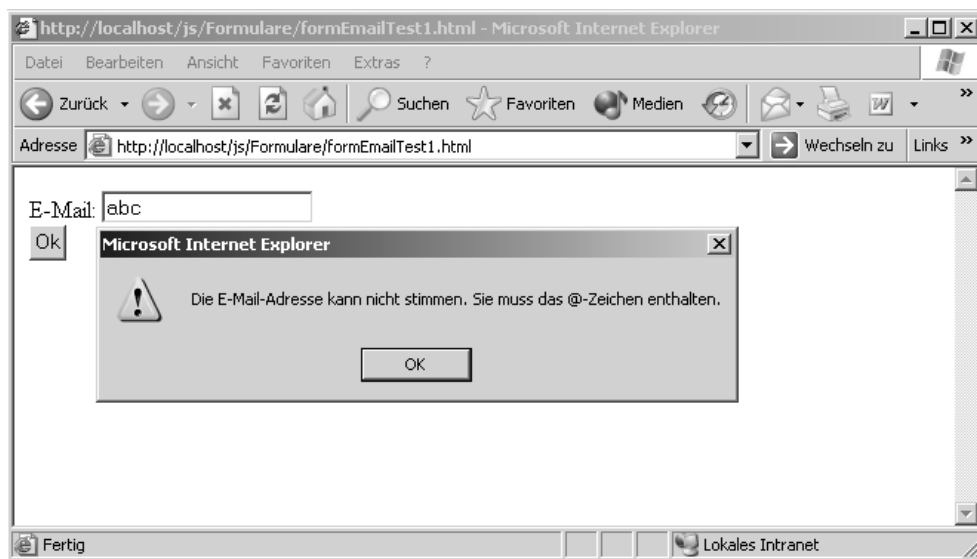


Abbildung 163: Das Zeichen @ fehlt bei der Eingabe der E-Mail-Adresse.

Wie erfolgt ein qualifizierter Test auf das Vorkommen des Zeichens @, einen Punkt und eine Mindestlänge?

Der Test auf das Vorhandensein von dem Zeichen `@` ist für eine einfache Absicherung ausreichend, aber keinesfalls zuverlässig. Natürlich genügt es für eine Formulareingabe nicht, wenn nur das Zeichen `@` in einer Eingabe vorkommt, um eine gültige E-Mail-Adresse zu sein. Vor dem `@`-Zeichen muss mindestens ein Zeichen notiert sein und nach dem `@`-Zeichen ebenfalls. Genau genommen muss nach dem `@`-Zeichen eine Domain-Angabe für den E-Mail-Server folgen, und auch die hat eine kontrollierbare Struktur (siehe unten). Zumindest muss

45. Der Rückgabewert wird in diesem Beispiel nicht weiter ausgewertet.

bei der Domain-Angabe mindestens ein Punkt auftauchen⁴⁶. Ebenso kann eine E-Mail-Adresse kaum unter fünf Zeichen realisiert werden⁴⁷. Sie können also auch eine verschärfte 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 sein.\n";
13         fehler++;
14     }
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)">
36     <br>
37     <input type="submit" value="Ok">
38 </form>
39 </body>
40 </html>
```

Listing 247: Ein etwas aufwändigerer Test der E-Mail-Struktur auf ein enthaltenes @-Zeichen, einen Punkt und eine Mindestlänge

46. Die Angabe `localhost` außer Acht gelassen – diese ist auch nur dann verwendbar, wenn Sie auf dem lokalen Rechner E-Mails verschicken (zu Testzwecken).

47. 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.

Das Beispiel besteht wie in der Version davor aus einem einfachen Formular (Zeile 33 bis 38) 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. Darin wird aber in dieser Version nur ein Parameter mit dem Schlüsselwort `this` als ersten Parameter übergeben, um darüber Zugriff auf das aktuelle Eingabefeld als Objekt in der Funktion zu haben. 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";`) und der Wert von `fehler` erhöht (Zeile 9 – `fehler++`). Damit ist die erste `if`-Bedingung abgeschlossen. In Zeile 11 kommt die Methode `indexOf()` zum Einsatz (`if(feld.value.indexOf(".") == -1)`), um auf das Vorhandensein eines Punktes zu prüfen. 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 `indexOf()` 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 `fehler` 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 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 `fehler` in Zeile 26 ebenso und der Wert `true` in Zeile 27 zurückgegeben. Diese Maßnahmen 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 `testMail()` 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.



Abbildung 164: Die Eingabe in dem E-Mail-Feld ist leer, und die Fehlermeldung zeigt alle Teilmeldungen an.

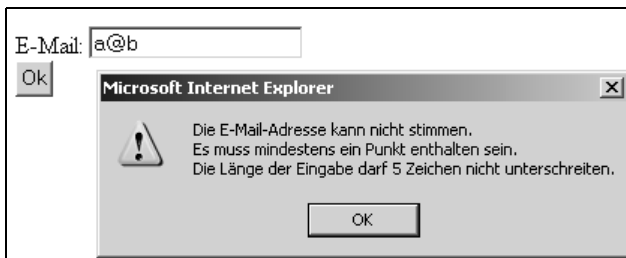


Abbildung 165: Die Eingabe im E-Mail-Feld verletzt nur noch zwei der drei zu erfüllenden Bedingungen.

Wie kann ich einen Test auf eine gültigen Toplevel-Domain durchführen?

Die potenziellen Kontrollmöglichkeiten des Aufbaus einer E-Mail-Adresse gehen aber noch weiter, 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 Host wäre also so darstellbar:

123.10.2.111

Listing 248: 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 dokumentiert. 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 System*⁴⁸, 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 größte logische beziehungsweise inhaltliche Einteilung dar. Das ist die so genannte *Toplevel-Domain*, der etwa eine Nation oder eine Organisationsform bezeichnet. Der (unter Umständen optionale) vorderste Teil bezeichnet direkt den einzelnen Host. 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
- ▶ webscripting.de
- ▶ www.gmx.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.

Tabelle 3: DNS-Toplevel

48. In der Literatur finden Sie aber für DNS auch *Domain Name Service* und *Domain Name Server*.

Toplevel	Bedeutung
mil	Militärische Computer in den USA.
name	Hosts mit persönlichen oder privaten Informationsangeboten.
net	Hosts von Organisationen, welche ein eigenes Netzwerk betreiben.
org	Hosts von nicht kommerziellen Organisationen.
rec	Hosts mit Freizeit und Unterhaltung als Thema.

Tabelle 3: DNS-Toplevel (Forts.)

Neben den logischen DNS-Topleveln gibt es Abkürzungen für die Ländernamen, die bei Toplevel-Domain-Namen zu Einsatz kommen. Diese Ländercodes sind in der ISO-3166-Norm festgelegt. Ein paar wichtige Beispiele sind folgende:

Toplevel	Land
at	Österreich
be	Belgien
ca	Kanada
ch	Schweiz
de	Deutschland
dk	Dänemark
es	Spanien
fr	Frankreich
it	Italien
nl	Niederlande
no	Norwegen
pl	Polen
pt	Portugal
ru	Russland
se	Schweden
tr	Türkei
uk	Großbritannien

Tabelle 4: Internationale DNS-Domains

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-Domain
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 249: Schematischer Test auf Übereinstimmung mit vorgegebenen Toplevel-Domains

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 wollen⁴⁹.

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 einem 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 250: Theoretisch mögliche E-Mail-Adressen

In diesen Adressen kommen mehrere Punkte vor, und nur der letzte dient zur Abtrennung des Toplevel-Domains. 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.

49. Sinnvoll sind auf jeden Fall `de`, `com` und `net`, dazu natürlich die weiteren Ländercodes im deutschsprachigen Raum. Die meisten offiziellen E-Mail-Provider sowie die firmeneigenen E-Mail-Dienste großer Firmen werden damit mit hoher Wahrscheinlichkeit abgedeckt sein.

Der Extrakt wird der Variable `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 extrahiertem 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 Zeile 13 bis 19 erfolgt dann die Reaktion auf die Situation.

Ein praktisches Beispiel für einen qualifizierten Test auf das Vorkommen des Zeichens @, einen Punkt und eine Mindestlänge sowie einen gültigen Toplevel

Schauen wir uns ein vollständiges Beispiel an, das dieses schematische Rezept zur Überprüfung eines gültigen Toplevels 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*):

```

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) {

```

Listing 251: Eine sehr weit reichend plausibilisierte E-Mail-Adresse

```

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 + "Der Toplevel-Domain ist ungültig.\n ";
49   fehler++;
50 }
51 if(fehler){
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" action="" method="POST">
65     E-Mail:
66     <input name="eins" onBlur="testEMail(this)">
67     <br>
68     <input type="submit" value="Ok">
69   </form>
70 </body>
71 </html>

```

Listing 251: 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 68). Mit dem Eventhandler `onBlur` wird die Funktion `testEMail()` aufgerufen, die mit `this` eine Referenz auf das aktuelle Eingabefeld übergeben bekommt. In den Zeile 4 bis 22 wird das Datenfeld mit den Toplevel-Domains angelegt, von denen Sie eine E-Mail akzeptieren wollen. 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.

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

HTML

Events



Abbildung 166: Der Anwender sieht alle Informationen, die er zur korrekten Ausfüllen einer E-Mail-Angabe benötigt.

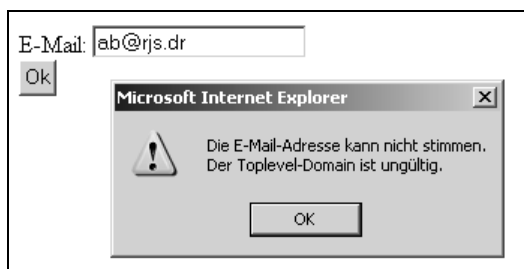


Abbildung 167: Die E-Mail-Adresse sieht auf den ersten Blick vernünftig aus, aber die Toplevel-Angabe passt nicht zu den Vorgaben.

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 den Sie nicht gedacht haben. Selbst das Fehlen des Punktes bei einer Domain-Angabe ist theoretisch denkbar (bei der Angabe `localhost`), obwohl das in der Praxis 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.

109 Wie kann ich ein freies Eingabefeld realisieren, in dem nur die Eingabe eines URLs gestattet ist?

In einigen Situationen ist es gewünscht, bei einer freien Texteingabe einem Anwender in dem Eingabefeld nur die Eingabe von einem gültigen URL zu gestatten. Rein von der Theorie her können Sie im Falle eines einzeiligen Eingabefelds diese Aufgabe mit HTML lösen, aber kein einziger Browser unterstützt die offiziellen Vorgaben dazu (*siehe das Rezept »Wie kann ich rein mit HTML ein einzeiliges Eingabefeld realisieren, in dem nur die Eingabe eines URLs gestattet ist?« auf Seite 194*). Über den `<input>`-Tag mit der `type`-Angabe `"url"` können Sie ein einzeiliges Eingabefeld generieren, in dem nach den offiziellen HTML-Vorgaben nur die Eingabe eines URLs 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 HTML können Sie dieses Problem nicht lösen. Bei einem mehrzeiligen Eingabefeld bietet HTML selbst von der Theorie her keine Möglichkeiten dazu, aber die Verwendung eines mehrzeiligen Eingabefelds ist in dieser Situation sowieso kaum anzuraten⁵⁰.

Sie sind auf eine Ergänzungstechnologie angewiesen, etwa JavaScript, was eine ideale Lösung darstellt. Dabei können Sie die nachfolgende Vorgehensweise auch auf den Rückgabewert der `prompt()`-Methode anwenden. Sie überprüfen 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 sinnvoll⁵¹ testen, ob es eine eingegebene URL-Angabe wirklich gibt, aber Sie können zumindest die grundsätzliche Struktur sowie den Inhalt auf einige Kernkriterien eines gültigen URLs plausibilisieren. Auch hier können Sie mehrere Stufen auswählen, wie plausibel die URL-Angabe wirklich sein soll. Allerdings muss gleich betont werden, dass auch hiermit nur eine eingeschränkte Überprüfung möglich ist.

Bei dem Test auf einen gültigen Toplevel-Domain bei einer E-Mail-Adresse haben wir das Grundkonzept für die Namensauflösung nach dem DNS-Verfahren gesehen. 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 bereit. Aber es sind zusätzlich noch einige andere Angaben notwendig.

- ▶ Die Angabe eines Protokolls, um den richtigen Dienst zu verwenden.
- ▶ Die Angabe eines so genannten Ports. Dabei handelt es sich um eine Art Kanal, auf dem ein Dienst Daten sendet und empfängt (so wie beim Funk). Standardports im Internet werden durch numerische Werte zwischen 0 und 1023 angegeben (insgesamt sind die Portwerte von 0 bis 65.535 möglich), und in der Regel hat jeder Internet-Standarddienst

50. Ein URL darf ja sowieso nur aus einer einzelnen Zeile bestehen.

51. Rein theoretisch könnten Sie einen angegebenen URL mit einer Testanfrage zu öffnen beziehungsweise anzupingen versuchen, 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.

einen Defaultwert, der immer dann verwendet wird, wenn man bei einer Adressangabe keinen expliziten Port angibt⁵². 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, welches konkret adressiert werden soll. Im WWW ist das etwa eine bestimmte Webseite. Diese Auswahl erfolgt über die Angabe von einem Pfad und der konkreten Dateibezeichnung, wie Sie es auch in einem lokalen Dateisystem – etwa auf einem PC – vornehmen. Wenn keine Datei explizit 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 den so genannten *URL (Uniform Resource Locator)*. Übersetzen kann man das mit »einheitlichem 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 eines 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 einen 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 von einem Pfad (der wiederum auch mit Slash getrennte Verzeichnisebenen enthalten kann) und einer Datei, sofern dies notwendig ist.

Achtung

Bitte beachten Sie, dass zur Trennung von Ebenen im URL der Slash (/) verwendet wird und nicht der Backslash (\) wie bei DOS-basierenden Systemen. Der Grund ist die Unix-Vergangenheit des Internets. DOS hatte mit dem Trennzeichen Backslash einen Sonderweg eingeschlagen, dem man im viel älteren Internet natürlich nicht gefolgt ist, der aber leider immer noch unter Windows verwendet wird. PC-Anwender mit WINTEL-Basis müssen sich deshalb für sämtliche Pfadangaben 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 dies intern wieder umsetzt, und darauf sollte man sich nicht verlassen oder gar daran gewöhnen.

52. Das ist der Regelfall.

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 – dumm 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.

Wenn wir nun die Struktur eines URLs betrachten, können Sie automatisiert nach einigen Bestandteilen plausibilisieren. Sie haben die Möglichkeit, nach einem gültigen Protokoll am Beginn des URLs zu suchen, ebenso die zwingende Trennsequenz `://` unmittelbar danach, sofern wir Sonderwege wie den Aufruf eines E-Mail-Clients aus dem Browser heraus außer Acht lassen. 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 einem URL nehmen. In der abschließenden Pfad- beziehungsweise Dateiangabe können Punkte vorkommen. Aber es müssen dort keine Punkte vorkommen, und die Pfad- beziehungsweise Dateiangabe kann ja auch unterbleiben. Ebenso können Sie nicht einfach die Punkte von vorne zählen. Es sollte besonders betont werden, dass bei einem DNS-Namen nur der 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 des jeweiligen Toplevel-Domains 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 – Bestandteilen⁵³, 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 keinsten 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 einem URL nicht explizit notiert, da es Defaultwerte gibt. Für eine automatisierte Plausibilisierung sind solche unverbindlichen Konventionen in jedem Fall unbrauchbar.

53. Es können auch weniger (genauer zwei – das findet man sehr oft) oder mehr sein.

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);
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 des URLs
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(tpleveldomain[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   }
```

Listing 252: Ein schematisches Rezept zum Prüfen auf einen gültigen URL

```
46 else {
47     // geeignete Maßnahmen, wenn kein Fehler in der URL-Angabe
48     // zu finden ist
49 }
50 }
```

Listing 252: Ein schematisches Rezept zum Prüfen auf einen gültigen 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 im URL akzeptieren wollen. Dabei gelten die gleichen Überlegungen wie für eine E-Mail-Adresse. 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. 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 hoch gezählt 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 den in Einzelstrings zerlegten URL aufnehmen.

In Zeile 14 wird die Eingabe des Anwenders mit der Methode `toLowerCase()` in Kleinbuchstaben konvertiert, um von Groß- und Kleinschreibung bei der Eingabe unabhängig zu sein. Mit der Anweisung 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 einen gültigen absoluten URL handelt, ist das in jedem Fall das Protokoll plus die Sequenz `//`. Diese Angabe ist dann in der Variablen `proto` gespeichert. 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 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 Teilstrings auf, indem an dem ange-

gebenen Separator / geteilt wird. Bei einem gültigen absoluten URL ist in jedem Fall im dritten⁵⁴ Arrayeintrag der DNS-Name oder die IP-Adresse enthalten. Und zwar ohne vorangestelltes Protokoll oder nachgestellte Pfadangaben! Wenn die Eingabe die grundsätzliche Struktur mit `[Protokoll]://[Host]` als Beginn des URL nicht einhält, wird die Arraygröße möglicherweise kleiner als 3. Zeile 21 fängt das ab und leitet im Fehlerfall geeignete Maßnahmen ein, wenn die grundsätzliche Struktur des URLs nicht stimmen kann (`if (dns.length < 3)`).

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 einen 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 des Toplevel-Domains. Mit dem Wert der Variablen `letzterPunkt` haben Sie nach der Ausführung von Zeile 27 die Position des letzten Punktes in diesem Teilstring des URL zur Verfügung. Dieser Wert plus 1 dient als Startwert der Methode `slice()` (Zeile 28), um das Ende von diesem Teilstring des 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 am besten die Sammlung von Informationen über alle Fehlerfälle sein. In Zeile 43 bis 50 erfolgt dann die Reaktion auf die Situation.

Ein praktisches Beispiel für einen qualifizierten Test auf einen gültigen URL

Schauen wir uns ein vollständiges Beispiel an, das dieses schematische Rezept zur Überprüfung eines gültigen URLs 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";
```

Listing 253: Test auf die Eingabe eines gültigen URL

54. Der Doppelslash zum Abtrennen des Protokolls sorgt für die ersten beiden Arrayeinträge, die wir aber hier nicht benötigen.

```
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 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 dem 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 + "Der Toplevel-Domain ist ungültig.\n";
55             fehler++;
56         }
57     }
58     if(fehler){
59         alert(meldung);
60         return false;
61     }
```

Listing 253: Test auf die Eingabe eines gültigen URL (Forts.)

Grund-
lagen

Core

Formu-
lareAus-
nahme

Arrays

Strings

Datum

Platt-
form

DHTML

Events

```

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>
73      <form name="meinForm" action="" method="POST">
74          E-Mail (absoluter URL):
75          <input name="eins" onBlur="testURL(this)">
76          <br>
77          <input type="submit" value="Ok">
78      </form>
79  </body>
80  </html>

```

Listing 253: Test auf die Eingabe eines gültigen URL (Forts.)

In den Zeile 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.

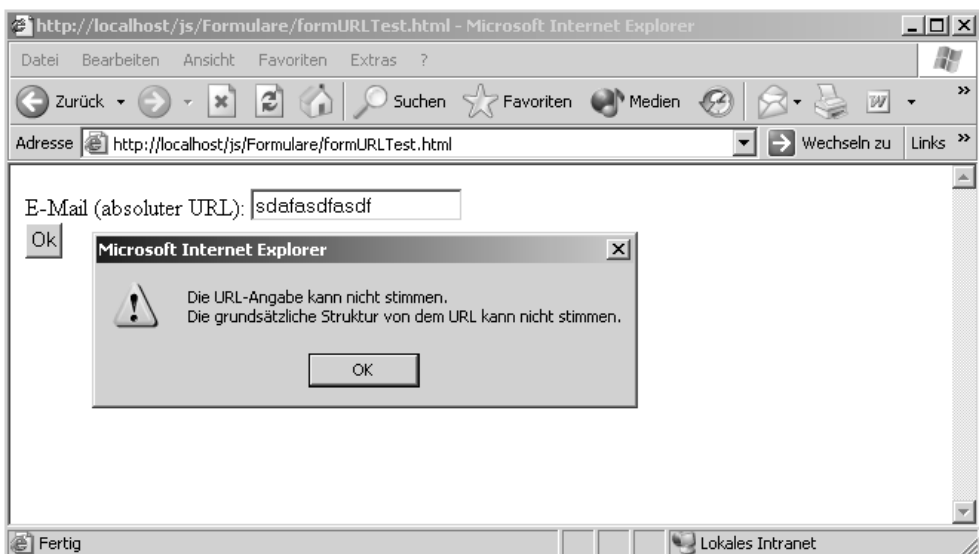


Abbildung 168: In der URL-Angabe ist ein grundsätzliches Problem in der Struktur.

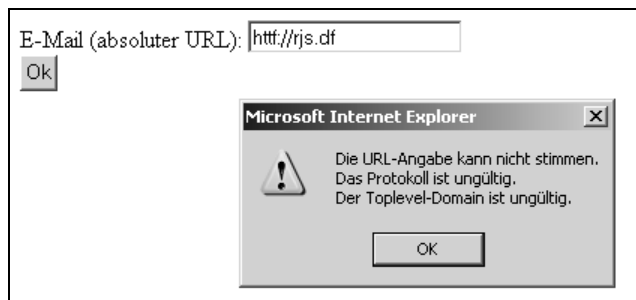


Abbildung 169: Der DNS-Name ist korrekt, aber das Protokoll stimmt nicht.

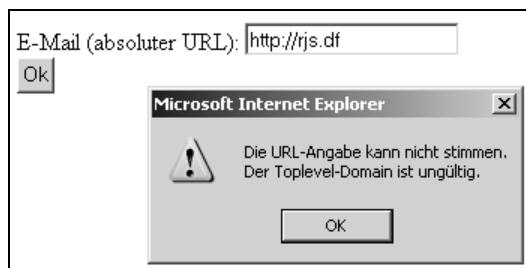


Abbildung 170: Struktur und Protokoll sind o.k., aber die Toplevel-Domain stimmt nicht mit den Vorgaben überein.



Abbildung 171: Struktur und Toplevel-Domain sind o.k., aber das Protokoll stimmt nicht mit den Vorgaben überein.

Hinweis

Je genauer Sie die Struktur eines URL festlegen, desto eher blockieren Sie auch versehentlich gültige Angaben. Ein 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 entscheiden, ob es nicht besser ist, auch falsche Angaben für einen URL zu akzeptieren und beispielsweise nur das Protokoll zu überprüfen.

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

HTML

Events

110 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 einzeiligen Eingabefelds diese Aufgabe mit HTML lösen, aber kein einziger Browser unterstützt die offiziellen Vorgaben dazu (*siehe das Rezept »Wie kann ich rein mit HTML ein einzeiliges Eingabefeld realisieren, in dem nur die Eingabe eines Kalenderdatums erlaubt ist?« auf Seite 192*). Bei einem mehrzeiligen Eingabefeld bietet HTML selbst von der Theorie her keine Möglichkeiten dazu, aber ein mehrzeiliges Eingabefeld ist in dieser Situation auch kein sonderlich sinnvoller Eingabeweg. Sie sind also auf JavaScript angewiesen. Dabei können Sie 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 kaum 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 *TT.MM.JJJJ*⁵⁵ und nicht jede denkbare Variante zu unterstützen⁵⁶. 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 ein String, und bevor Sie mit diesen Informationen ein Datumsobjekt 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 angesagt, und da helfen die String-Methoden, die wir auch beim Test auf ein gültiges E-Mail-Format beziehungsweise einen gültigen URL verwendet haben.

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.

Die einfachste Überprüfung ist der Monat. Es muss zwingend eine Zahl zwischen 1 und 12 sein.

55. Also den Tag zweistellig, dann ein Punkt, dann den Monat zweistellig, wieder ein Punkt und das Jahr mit vier Stellen. Sinnvoll ist es aber, bei einer einstelligen Eingabe von Tag und Monat einen führende Null per JavaScript zu ergänzen und einen Anwender bei einer Eingabe von beispielsweise 5.2.2004 nicht zu zwingen, seine Eingabe in der Form 05.02.2004 zu wiederholen.

56. Manche Anwender mögen vielleicht *TT/MM/JJ* oder *TT/MM/JJJJ* oder *TT.MM.JJ* oder was sonst so noch möglich ist – auch nach internationalem Format. Das wird ein Fass ohne Boden.

Die Überprüfung des Jahres ist ebenso recht einfach. Sie müssen nur selbst entscheiden, welche Jahresangaben für Ihr Projekt sinnvoll sind.

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 Datums
11         // 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(
```

Listing 254: Schematische Überprüfung auf ein gültiges Datum

```
32     (datum[2]%400 == "0") ? (1) : (
33         (datum[2]%100 == "0") ? (0) : (
34             (datum[2]%4 == "0") ? (1) : (0)
35         )
36     )
37 ) {
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 254: 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 hoch gezählt zu werden. In Zeile 3 wird ein Datenfeld angelegt (`datum = new Array();`). In Zeile 4 wird die Benutzereingabe an dem Punkt in Teilstrings aufgeplittet 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 unterschiedliches 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 Nachkomma-Anteil abgeschnitten. Das Ergebnis des Vergleichs kann also nur dann `true` liefern, wenn es sich bei `zahl` um eine Ganzzahl handelt. Falls dem 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 irgendein offensichtliches Problem in der Struktur der Datumsangabe bereits 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 von der Zeile 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 abzu prüfen. Der Test auf ein Schaltjahr erfolgt mit dem konditionalen Operator. Im Allgemeinen ist dessen Verwendung 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 von dem konditionalen Operator findet nicht statt.

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

Wenn die Auswertung von `(datum[2]%400 == "0")` `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 von dem konditionalen Operator findet nicht statt.

Wenn die Auswertung von `(datum[2]%100 == "0")` `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 von dem konditionalen Operator 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 stimmen.\n";
16         fehler++;
17     }
18     else{
19         if(
20             (datum[2] < 1900) ||
21             (datum[2] > 2100)
22         ){

```

Listing 255: Test auf die Eingabe eines gültigen Datums

```
23     meldung = meldung +
24         "Das Jahr ist ungültig. Geben Sie das bitte vierstellig 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     (datum[0] > tag)
69 ){
70     meldung = meldung +
71         "Der Tag ist ungültig.\n";
72     fehler++;
73 }
```

Listing 255: Test auf die Eingabe eines gültigen Datums (Forts.)

Grund-
lagen

Core

Formu-
lareAus-
nahme

Arrays

Strings

Datum

Platt-
form

DHTML

Events

```
74 }
75 if(fehler){
76     alert(meldung);
77     return false;
78 }
79 else {
80     meldung = "Die Datumsangabe kann nicht stimmen.\n";
81     fehler = 0;
82     return true;
83 }
84 }
85 //-->
86 </script>
87 <body>
88 <form name="meinForm" action="" method="POST">
89     Datum :
90     <input name="eins" onBlur="testDatum(this)">
91     <br>
92     <input type="submit" value="Ok">
93 </form>
94 </body>
95 </html>
```

Listing 255: 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 erfolgt wie beim Test einer E-Mail-Eingabe. Am Ende wird wieder entweder eine Fehlermeldung angezeigt oder aber das System zurückgesetzt.

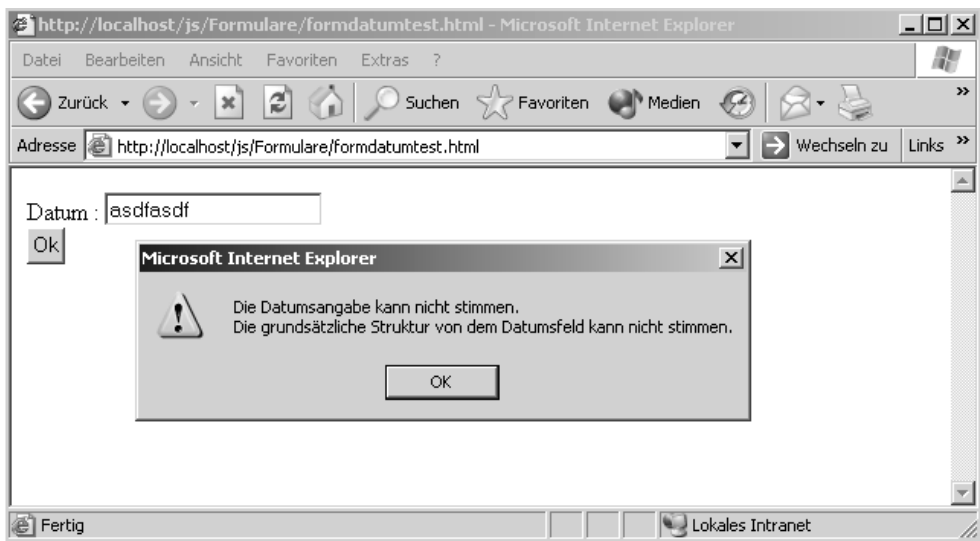


Abbildung 172: Ein grundsätzliches Problem in der Datumseingabe



Abbildung 173: Das Jahr 2003 war kein Schaltjahr – deshalb kann es keinen 29. Februar geben.

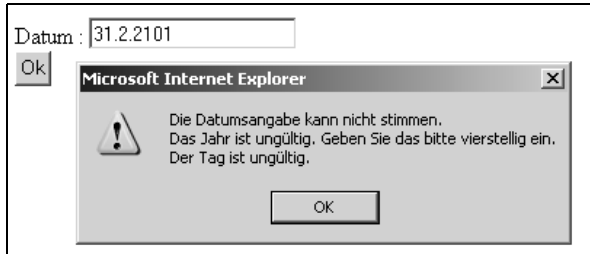


Abbildung 174: Sowohl der Tag als auch das Jahr verletzen Randbedingungen.

111 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 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 256: 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 Browserfenstern⁵⁷, und der Rechner friert ein. Mit einigem 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.)

57. Es besteht keine Chance, die Fenster in ähnlicher Geschwindigkeit zu schließen, wie neue aufpoppen ;-).

auch in ein Gästebuch oder ein Diskussionsforum auf HTML-Basis zu schreiben. Wenn sich dann ein anderer Besucher die Einträge in dem Gästebuch ansieht, wird dessen Browser das Skript interpretieren⁵⁸ und immer neue Instanzen des Browsers mit besagtem Effekt öffnen⁵⁹. Es ist also einem Betreiber von Webformularen, deren Eingaben nach dem Verschieken in fremdem 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 diesem Rezept grundsätzlich zum Einsatz kommt. Allerdings macht es beim Ersetzen bestimmten 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. Nur kann die Verhinderung diverser Inhalte zahlreiche Besucher 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 manchen Situationen oft sinnvoll (etwa um eine Datenbank konsistent zu halten).

So könnte eine schematische Funktion aussehen, die einen bestimmten Inhalt verhindert:

```
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 257: Suche nach verbotenen Inhalten

58. Es sei denn, die Interpretation von Skripten ist deaktiviert.

59. So etwas passiert natürlich auch, wenn ein solcher 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 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 kaum weiterer Argumente, die Verwendung von HTML in E-Mails auszuschalten.

Die Funktion ergreift im Fall von verbotenen Inhalten (der Vergleich liefert `true`) entsprechende Gegenmaßnahmen.

Tipp Im Kapitel über Stringmanipulationen besprechen wir den Umgang mit so genannten *regulären Ausdrücken*. Im Rezept »Wie kann ich mit regulären Ausdrücken in Strings suchen und sie ersetzen?« finden Sie über die hier besprochenen schematische 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" action="" method="POST">
20     Ihr Kommentar:
21     <textarea cols="50" rows="5" name="eins" onBlur="testInhalt(this,'<')">
22     </textarea>
23     <br>
24     <input type="submit" value="Ok">
25 </form>
26 </body>
27 </html>

```

Listing 258: 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 Eingabefelds 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 HTML-Tags eingeben kann! Damit ist natürlich auch die Eingabe von Skript-Containern verhindert.

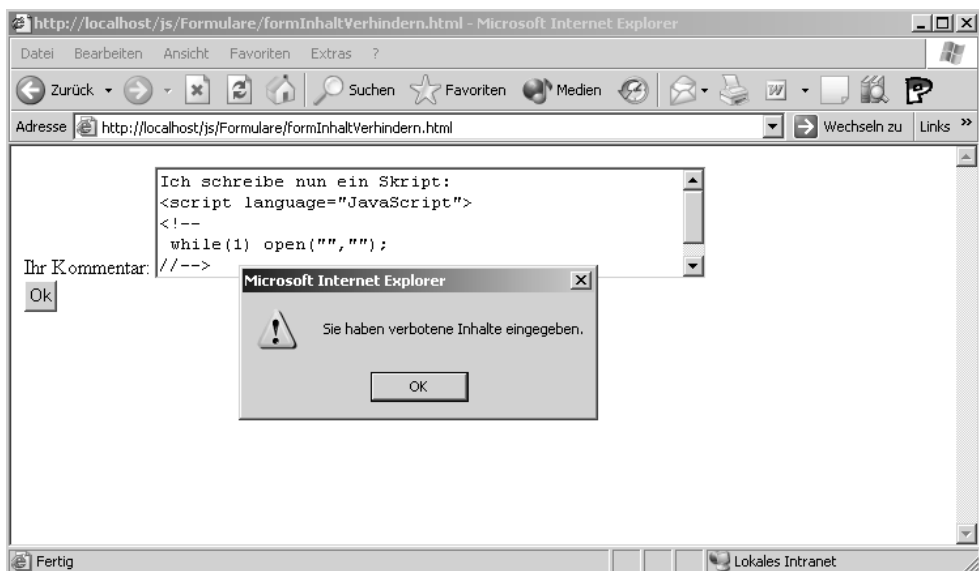


Abbildung 175: Gefahr erkannt, Gefahr gebannt.

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 Maskierung ein Zeichen einzugeben, dass 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 ISO 5589-1-Norm), in der jedes Zeichen eine eindeutige 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 ISO 5589-1-Norm und ihre zugehörige Maskierung im Rahmen von 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 `#` und Abschluss mit einem Semikolon (z.B. `!` für `!`). 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	SP	
33	!	
34	"	<code>&quot;</code>
35	#	
36	\$	
37	%	
38	&	<code>&amp;</code>
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	<	<code>&lt;</code>
61	=	
62	>	<code>&gt;</code>
63	?	

Tabelle 5: Maskierungen von Zeichen

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

Nummerncode	Zeichen	Characterentity
64	@	
65	A	
66	B	
67	C	
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	

Tabelle 5: Maskierungen von Zeichen (Forts.)

Nummerncode	Zeichen	Characterentity
99	c	
100	d	
101	e	
102	f	
103	g	
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	...	

Tabelle 5: Maskierungen von Zeichen (Forts.)

Nummerncode	Zeichen	Characterentity
134	†	
135	‡	
136	^	
137	‰	
138	Š	
139	◁	
140	CE	
141		
142		
143		
144		
145	'	
146	'	
147	"	
148	"	
149	o	
150	-	
151	-	
152	~	
153	™	
154	š	
155	›	
156	œ	
157		
158		
159	Ÿ	
160	(erzwungenes Leerzeichen)	
161	¡	¡
162	¢	¢
163	£	£
164	¤	¤
165	¥	¥
166		¦
167	§	§
168	¨	¨

Tabelle 5: Maskierungen von Zeichen (Forts.)

Nummerncode	Zeichen	Characterentity
169	©	©
170	ª	ª
171	«	«
172	¬	¬
173	-	­
174	®	®
175	-	¯
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	Ç	Ç
200	È	È
201	É	É
202	Ê	Ê
203	Ë	Ë

Tabelle 5: Maskierungen von Zeichen (Forts.)

Nummerncode	Zeichen	Characterentity
204	Ì	Ì
205	Í	Í
206	Î	Î
207	Ï	Ï
208	Ð	Ð
209	Ñ	Ñ
210	Ò	Ò
211	Ó	Ó
212	Ô	Ô
213	Õ	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

Tabelle 5: Maskierungen von Zeichen (Forts.)

Nummerncode	Zeichen	Characterentity
239	ï	ï
240	ð	ð
241	ñ	ñ
242	ò	ò
243	ó	ó
244	ô	ô
245	õ	õ
246	ö	ö
247	÷	÷
248	ø	ø
249	ù	ù
250	ú	ú
251	û	û
252	ü	ü
253	ý	ý
254	þ	þ
255	ÿ	ÿ

Tabelle 5: Maskierungen von Zeichen (Forts.)

Betrachten wir ein Beispiel, das bei einem verboten 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     }

```

Listing 259: Austausch kritischer Zeichen durch unkritische Zeichen

```

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" action="" method="POST" onSubmit="werteAnpassen()">
28         Ihr Kommentar:
29         <textarea cols="50" rows="5" name="eins">
30         </textarea>
31         <br>
32         <input type="submit" value="Ok">
33     </form>
34 </body>
35 </html>

```

Listing 259: Austausch kritischer Zeichen durch 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 Referenz⁶⁰. 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 neues 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(verboten) > -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(verboten);`). 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 = "";`).

60. Natürlich können Sie auch den Rückgabewert der `prompt()`-Methode oder jeden anderen String angeben.

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.



Abbildung 176: Auweia – ohne Maskierung kann 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.

Diese Information werden Sie in der Praxis dem Anwender natürlich nicht präsentieren.

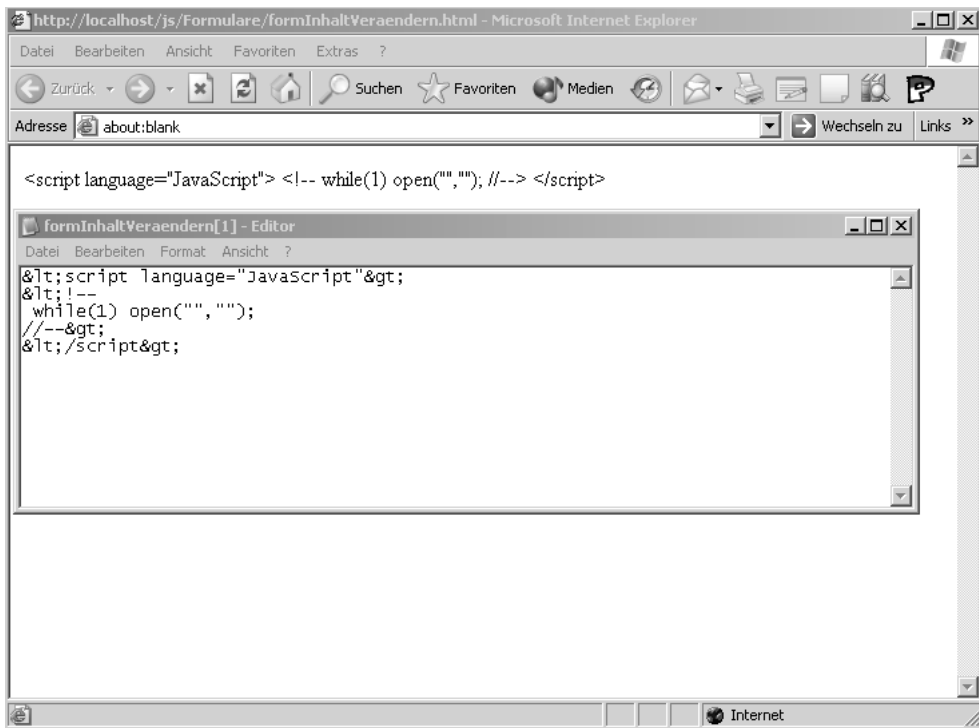


Abbildung 177: Der Zahn ist gezogen.

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 ungünstigsten 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 andere Zeichen beim Austausch diese Probleme übrigens nicht. Wie dem auch sei – das hier gezeigte Rezept umgeht diese Schwierigkeiten elegant.

112 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

trivial, und die wesentlichen Dinge betreffen nicht die Programmierung selbst. Plausibilisierung von einem Webformular ist nur als globales Konzept schlüssig, das auch die Weiterverwendung der Daten nach dem Versenden beinhalten muss.

113 Welche Problemstellungen müssen beachtet werden?

Bei der Plausibilisierung von einem Webformular 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 Benutzereingaben samt weiteren notwendigen Daten übermittelt. Ein Online-Shop besitzt zusätzlich meist eine angeschlossene Datenbank und 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 als 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 Webserver über Servertechnologien wie PHP) vielfach Programmierlaien⁶¹ versuchen, denen dann in der Regel auch kein professionelles Umfeld mit Konzeption, Test etc. zur Seite steht.

61. Das soll nicht abwertend gemeint sein. Es soll nur die Erfahrung im Umgang mit Programmierung und Plausibilisierungskonzepten bewerten. Selbstverständlich muss das nicht jedermann beherrschen.

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 Eingabe⁶²? 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 werden⁶³. Oder die Beziehung zwischen dem eingegebenen Name 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 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 auf einem Feld⁶⁴ der Anwender auf 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 vorkommen beziehungsweise programmiert werden kann, ist das ein Vorgehen, das seit der Einführung grafischer Oberflächen im Allgemeinen nicht mehr tolerierbar ist, weder bei einem Standalone-Programm⁶⁵, 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

62. Etwa für ein Passwort.

63. In vielen Fällen muss etwa das berechnete Alter größer als 18 Jahre sein.

64. Etwa die Eingabe von Text, wo eine Zahl verlangt wurde.

65. Obwohl es da Situationen geben kann, in denen das noch akzeptabel ist.

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. Allgemein ist es also sinnvoll, weder beim Fokussieren noch beim Verlassen von Formularfeldern zu plausibilisieren.

Aber wo denn dann? 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 Seite⁶⁶ oder gar überhaupt nicht beim Client. 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 Onlinegeschäft.

Die Variante, in der sich der Client vollständig um die Plausibilisierung kümmert, hat diverse Vorteile. Zwar ist HTML zu keinerlei nennenswerten Plausibilisierungen in der Lage⁶⁷, 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 formuliertes Formular überhaupt abgeschickt wird. Nachteile

66. Die Eingaben des Benutzers werden beispielsweise in versteckten Feldern oder globalen Skriptvariablen gemerkt.

67. 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.

clientseitiger Plausibilisierungen jenseits der eingeschränkten Möglichkeiten von 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 via einer Technik wie PHP oder ASP die ausschließliche Plausibilisierung übernimmt, kann man dort fast alle Plausibilisierungen durchführen⁶⁸. Die Nachteile sind jedoch gravierend. Hier sind drei der wichtigsten:

1. Das Laufzeitverhalten einer serverseitig plausibilisierten Webapplikation ist schlecht. Bis eine Antwort vom Server auf eine Anfrage zur Plausibilisierung von bestimmten Daten wieder beim Client ist, kann viel Zeit vergehen.
2. Es kommt bei einer serverseitig plausibilisierten Webapplikation zu einer starken Belastung der Kommunikationswege durch unnütz hin- und hergeschickte 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 unnütz Kapazität verbraucht. Ebenso werden die Kommunikationswege mit überflüssigen Daten verstopft, und eine Fehlermeldung benötigt unter Umständen immens 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 Anwender dort statt Zahlen Buchstaben einträgt⁶⁹, 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 dem Server zurück zum Client.

Nun stellt sich bei Applikationen mit Datenbankbindung 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 Programmier Technik 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.

68. Die wenigen Ausnahmen betreffen Einschränkungen der Servertechnik, die sich aufgrund fehlender Informationen dort ergeben, etwa kann man mit den meisten serverseitigen Programmier Techniken wie PHP nicht so einfach Informationen über den Bildschirm des Anwenders erhalten, da diese Informationen nicht implizit mit HTTP übertragen werden.

69. Etwa der klassische Flüchtigkeitsfehler, statt einer Null den Buchstaben 0 einzutragen.

Diese Empfehlung muss natürlich begründet werden. Erstens gilt bei der Plausibilisierung auf der Datenbank erst recht, dass die Performance unzumutbar 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 samt seinen Gehilfen wie dem Skriptinterpreter 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 weitergeleitet werden. Dazu ist es in der Regel ein Problem, dass die Datentypen einer Datenbank nicht ohne Konvertierung mit Datentypen aus Webeingaben umgehen können. HTML-Formulare übermitteln ausschließlich Text – auch beispielsweise 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 extrem schlechten Performance bestraft 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 HTML bietet und die von allen Browsern unterstützt werden. Alles Weitere programmieren Sie in einer Technik wie JavaScript. Ist das beim Client aufgrund logischer Probleme⁷⁰ 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, ASP⁷¹ oder JSP. Die Datenbank sollte danach nur mit verträglichen Werten gefüttert werden. Nur bei nicht einkalkulierten Situationen sollte die Datenbankplausibilisierung eine letzten Firewall darstellen. 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).

70. Etwa weil Sie Daten brauchen, die auf dem Client nicht vorhanden sind, zum Beispiel die Schlüssigkeit von Postleitzahl und Ort.

71. Bei ASP nutzen Sie naheliegendermaßen das 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('['FieldName']')` haben Sie bei ASP Zugang zu einem Formularfeld.

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 einsetzen) stehen Ihnen die zahlreichen Rezepte aus diesem Kapitel bereit. 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>
07 <td>
08 Kunde
09 </td>
10 <td>
11 <input name="Kd" maxLength=40>
12 </td>
13 <td>
14 Kundennummer
15 </td>
16 <td>
17 <input name="Kdnr">
18 </td>
19 </tr>
20 <tr>
21 <td>
22 Tee
23 </td>
24 <td>
25 <input type="Checkbox" name="tee" value="Tee">
26 </td>
27 <td>
28 Menge
29 </td>
30 <td>
31 <input name="teeanz" value=0>
```

Listing 260: Das reine Webformular

```

32 </td>
33 </tr>
34 <tr>
35 <td>
36 Kaffee
37 </td>
38 <td>
39 <input type="Checkbox" name="kaf" value="Kaffee">
40 </td>
41 <td>
42 Menge
43 </td>
44 <td>
45 <input name="kafanz" value=0>
46 </td>
47 </tr>
48 </table>
49 <hr><input type="Submit" value="Ok">
50 <input type="reset" value="Abbruch">
51 </form>
52 </body>
53 </html>

```

Listing 260: 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 HTML über den Parameter `maxLength` auf 40 beschränkt (Zeile 11). 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 von dem Rückgabewert `false`, dass das Formular überhaupt abgeschickt wird. Ist der Wert aber `true`, wird das angegebene Script (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.

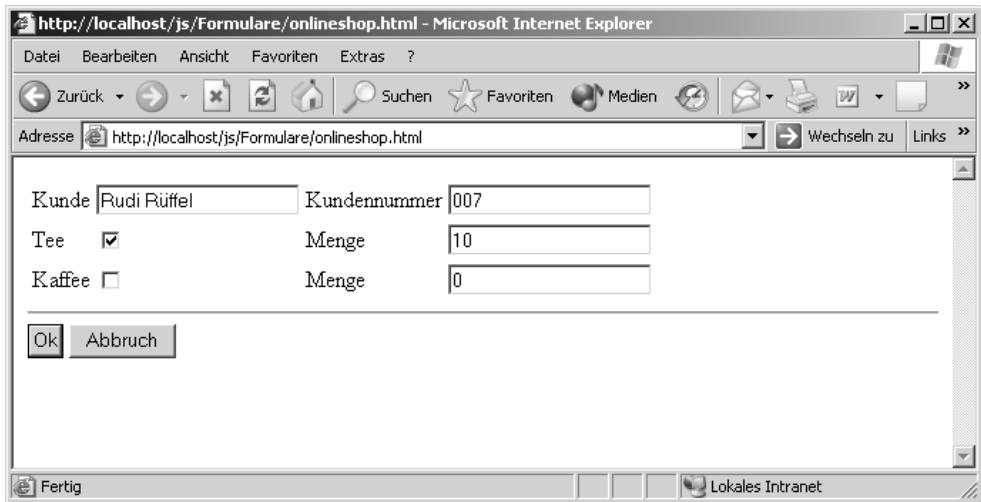


Abbildung 178: Das Webinterface von dem Online-Shop

Schauen wir uns nun den Skriptbereich mit den konkreten Plausibilisierungen an:

```

01 function kname() {
02   if(document.shop.Kd.value=="") {
03     alert("Sie müssen einen Kundennamen eingeben");
04     return false;
05   }
06   else{
07     return true;
08   }
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 teean () {
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)){

```

Listing 261: Die reine Plausibilisierung von dem Webformular

```
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() {
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 261: Die reine Plausibilisierung von dem Webformular (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 dem Rückgabe `false` ab. Wirft keine der `if`-Anweisungen `false` aus, liefert `plausi()` den Wert `true`. Mit anderen Worten – die Zeile 62 wird nur dann erreicht, wenn keine der für unser Modell relevanten Bedingungen verletzt sind. 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.

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

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 Kundenamen leer ist. Da der Kundename für unser Modell zwingend sein soll, wird eine Fehlermeldung angezeigt und `false` zurückgegeben, wenn der Kundennamen nicht angegeben wurde.



Abbildung 179: Der Kundename 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 1000 liegt. Das bedeutet indirekt, dass die Eingabe mindestens vierstellig ist. Damit kann bei dem Eingabefeld auf das HTML-Attribut `maxlength` verzichtet werden, aber in der Praxis in ein solches natürlich nie von Nachteil – eventuell auch zusätzlich. 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 180: 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 0 sein. Auch wird die Eingabe einer Bestellmenge größer 0 nur akzeptiert, wenn die zugehörige Checkbox selektiert ist⁷². Falls die Checkbox nicht ausgewählt wird, wird grundsätzlich die Bestellmenge auf den Wert 0 gesetzt. Die Funktion `teean()` in der Zeile 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 (`if(!isNaN(document.shop.teeanz.value))`). Die zweite Bedingung in Zeile 28 garantiert, dass der Wert nichtnegativ 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;`). 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 als auch die nachfolgende Anweisung in Zeile 37 (`return false;`) nicht über eine `if`-Bedingung abgesichert werden.

72. 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.



Abbildung 181: 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 einzelilem 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 Plausibilitätenmodell entsprechend geringfügig (*onlineshop2.html*).

```

01 meldung = "";
02 function kfname() {
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 () {
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) &&

```

Listing 262: Das modifizierte Plausibilitätenkonzept, in dem alle Fehler des Anwenders in einem Dialog angezeigt werden

```

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 (!kname()) 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 262: Das modifizierte Plausibilitätenkonzept, in dem alle Fehler des Anwenders in einem Dialog angezeigt werden (Forts.)

Im Unterschied zur ersten Variante des Plausibilitätenkonzepts sammelt diese Version alle Fehlermeldungen vor der Ausgabe in einer globalen Variablen. In Zeile 1 wird mit `meldung = ""`; eine entsprechende Variable global eingeführt. Diese wird beim Laden der Webseite als

Grund-
lagen

Core

Formu-
lareAus-
nahme

Arrays

Strings

Datum

Platt-
form

DHTML

Events

Leerstring initialisiert. Jede der einzelnen Plausibilisierungsfunktionen hängt nun – statt eine Meldung direkt auszugeben – im Fehlerfall eine individuelle Meldung an den Wert dieser Variable an (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 + 37` "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";`). 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 Varianten 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 auch in den ersten Varianten überwacht. Beachten Sie, dass hier jedoch der Rückgabewert `false` von einer der einzelnen Plausibilisierungsfunktionen nicht zum direkten return mit eben selbigen `false` von `plausi()` führt, sondern die Variable `fehler` wird erhöht (Zeile 60 – `if (!kdnr()) 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. Beachten Sie, dass beim Zusammensetzen der Fehlermeldung das Steuerzeichen `\n` am Ende jeder Einzelmeldung steht. Das sorgt für einen Zeilenumbruch in Fehlerfenster zwischen den einzelnen Meldungen.

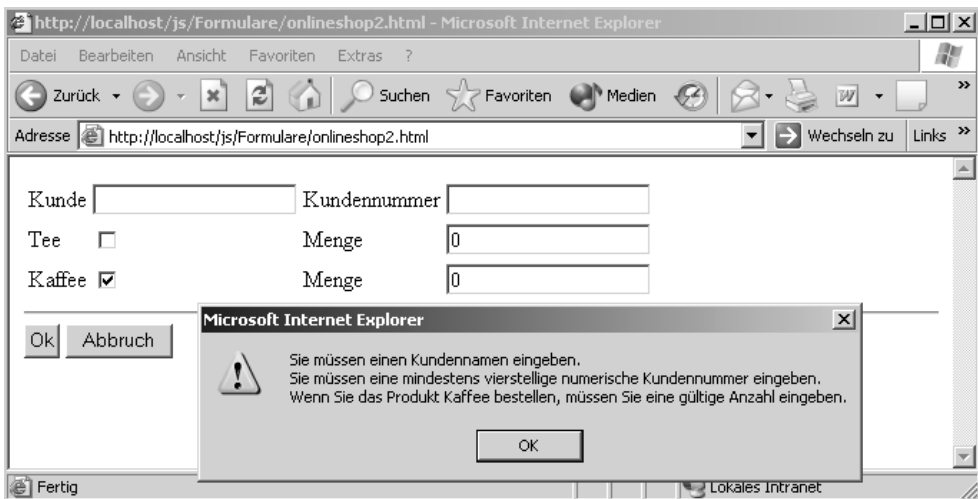


Abbildung 182: Alle Fehler des Anwenders werden beim Abschicken der Formulardaten in einem einzigen Dialog angezeigt.

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 teean () {
34     if(document.shop.tee.checked!=true) {
35         document.shop.teeanz.value=0;
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{

```

Listing 263: Das weiter modifizierte Plausibilitätenkonzept, in dem alle Fehler des Anwenders in der Webseite selbst angezeigt werden

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

```
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++;
87     if (!teean()) fehler++;
88     if (!kafan()) fehler++;
89     if(fehler == 0) {
90         return true;
91     }
```

Listing 263: Das weiter modifizierte Plausibilitätenkonzept, in dem alle Fehler des Anwenders in der Webseite selbst angezeigt werden (Forts.)

```

92 else {
93     return false;
94 }
95 }
96 //-->
97 </script>
98 <noscript></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 263: Das weiter modifizierte Plausibilitätenkonzept, in dem alle Fehler des Anwenders in der Webseite selbst angezeigt werden (Forts.)

Im Gegensatz zu der letzten Variante des Plausibilitätenkonzepts 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 das fehler-

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

HTML

Events

hafte 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 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` durch 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` durch 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 Abfangen der Versendung von den Formulardaten einzelne Fehler behoben wurden, werden die jeweiligen Fehleranzeigen wieder durch die Grafik `leer.gif` ausgetauscht.

The screenshot shows a Microsoft Internet Explorer window with the address bar displaying `http://localhost/js/Formulare/onlineshop3.html`. The form contains the following elements:

- A text input field labeled "Kunde".
- A text input field labeled "Kundennummer".
- A section for "Tee" with a checkbox and a text input field labeled "Menge" containing the value "0".
- A section for "Kaffee" with a checkbox and a text input field labeled "Menge" containing the value "0".
- At the bottom, there are two buttons: "Ok" and "Abbruch".

The browser's status bar at the bottom shows "Fertig" and "Lokales Intranet".

Abbildung 183: Das leere Formular

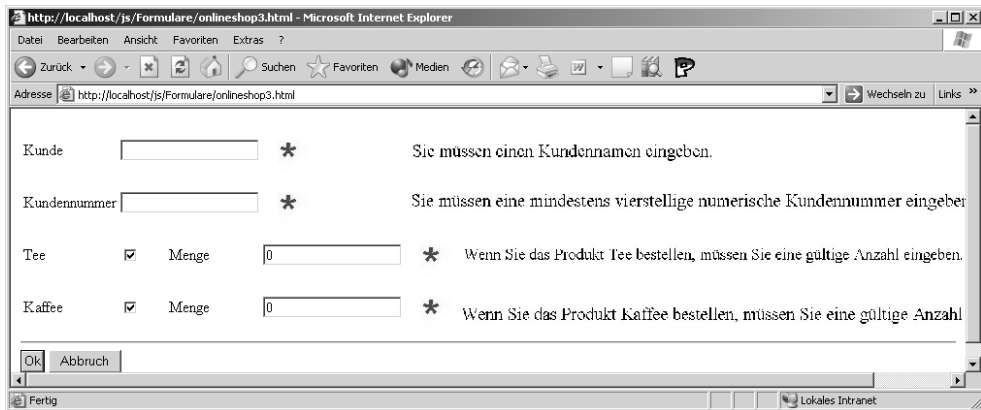


Abbildung 184: Alle Eingabemöglichkeiten des Formulars sind falsch.

Hinweis

Der Austausch der Grafiken per JavaScript funktioniert in den meisten Browsern sehr zuverlässig. Jedoch sollten Sie das auf jeden Fall in allen gewünschten Browsern testen. Insbesondere kann es zu Problemen mit der Anzeige der Grafiken kommen.

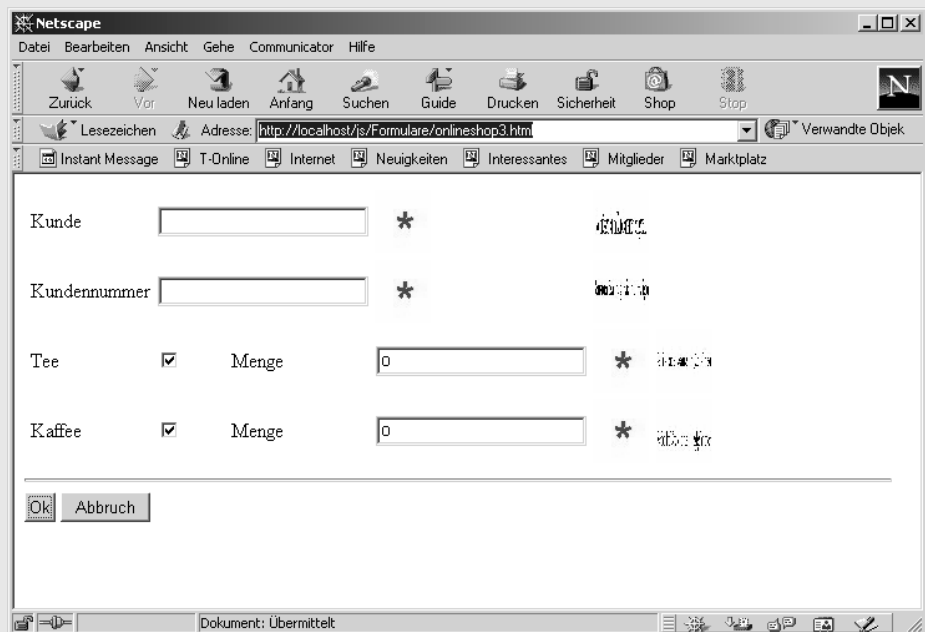


Abbildung 185: Im Netscape Navigator 4.7 laufen die Grafiken aus dem Ruder, wenn Grafiken ausgetauscht werden.

Im Zweifelsfall experimentieren Sie mit verschiedenen Grafiken und setzen 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 eine Abwandlung davon werden Sie in der Praxis jedoch 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 aufgrund 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 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 264: Die extrem kleine HTML-Datei onlineshop4.html für den plausibilisierten Online-Shop

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 kname() {
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) ||
```

Listing 265: Die externe JavaScript-Datei onlineshop4.js mit sämtlichen Funktionalitäten von dieser Variante des plausibilisierten Online-Shops


```

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 teean () {
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";
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] =

```

Listing 265: Die externe JavaScript-Datei onlineshop4.js mit sämtlichen Funktionalitäten von dieser Variante des plausibilisierten Online-Shops (Forts.)

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

```

62     "Wenn Sie das Produkt Kaffee bestellen, m&uuml;ssen Sie eine g&uuml;ltige
Anzahl eingeben.\n";
63     return false;
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><body><form
name="shop" action="bestellung.php" method="post" onSubmit="return
plausi()"><table><tr><td>Kunde</td><td colspan="3" align="right"><input name="Kd"
maxlength=40>');
81     if(fehler == 0) {
82         document.write('</td><td></td></tr>');
83     }
84     else {
85         document.write('</td><td>'+ meldung[0]+ '</td></tr>');
86     }
87     document.write('<tr><td colspan="3" align="right">Kundennummer</td><td><input
name="Kdnr"></td>');
88     if(fehler == 0) {
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 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 name="kafanz" value=0></td>');
102    if(fehler == 0) {
103        document.write('</td><td></td></tr>');
104    }

```

Listing 265: Die externe JavaScript-Datei onlineshop4.js mit sämtlichen Funktionalitäten von dieser Variante des plausibilisierten Online-Shops (Forts.)

```

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 265: Die externe JavaScript-Datei onlineshop4.js mit sämtlichen Funktionalitäten von dieser Variante des plausibilisierten Online-Shops (Forts.)

Die Funktion `generiereFormular()` erstreckt sich von Zeile 79 bis 110. Diese Funktion wird von der 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 wird⁷³ und im `<body>`-Tag kein `onLoad`-Aufruf für die Funktion `generiereFormular()` mehr stehen darf⁷⁴. Ansonsten enthält die Funktion im Grunde nur den Aufbau des Webformulars in einer Tabellenstruktur, 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></tr>'); } else { document.write('</td><td>'+ meldung[0]+ '</td></tr>'); }`). 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 generiert⁷⁵. 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 geschlossen 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.

73. Sonst würden alle folgenden Aufrufe von JavaScript-Funktionen nicht gefunden.

74. Sonst würde sich die Funktion ja beim Generieren der Webseite unmittelbar wieder selbst aufrufen.

75. Wenn Sie das Neuschreiben aus der Webseite selbst auslösen, überschreiben Sie damit auch die Variableninhalte von dem String-Array und der Fehlervariablen. Das ist – wie oben erwähnt – so, als würden Sie sich den Ast absägen, auf dem Sie sitzen.

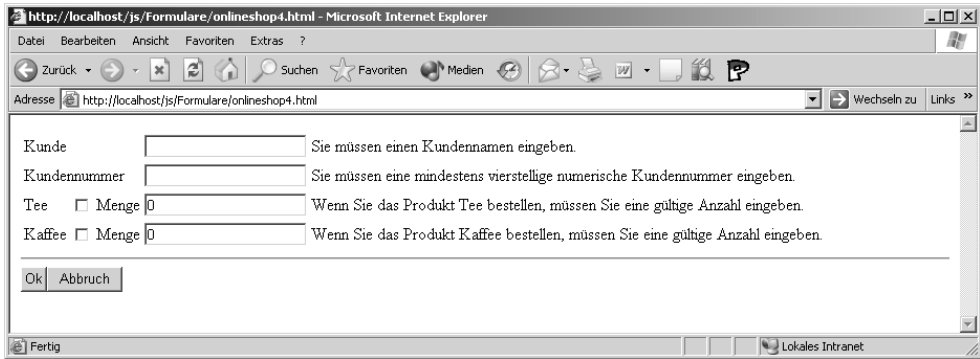


Abbildung 186: Die Fehlermeldungen werden direkt hinter der Fehlerstelle angezeigt.

Jetzt wird Ihnen auffallen, dass im Falle eines Fehlers die Webseite vollkommen neu geschrieben wird und die jeweiligen Fehlermeldungen angezeigt werden. Aber es sind 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.

Der `else`-Block zum Aufruf von `generiereFormular()` im Fehlerfall erstreckt sich von Zeile 73 bis 77 und sieht bisher so aus:

```

73  else {
74      generiereFormular();
75      fehler = 0;
76      return false;
77  }
```

Listing 266: 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 267: Die modifizierte Version von dem `else`-Block mit dem Aufruf von `generiereFormular()` im Fehlerfall

In dieser modifizierten Version merken Sie sich die Eingaben für alle interessanten Felder (hier nur der Kundename 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()` etwa 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 268: Die modifizierte Funktion `kdnr()`

In Zeile 6 wird im Fehlerfall der Wert von dem Eingabefeld geleert.

Achtung Diese Variante ist nicht für alle Browser geeignet. Einwandfrei funktioniert sie im Opera und dem Internet Explorer. Im 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.

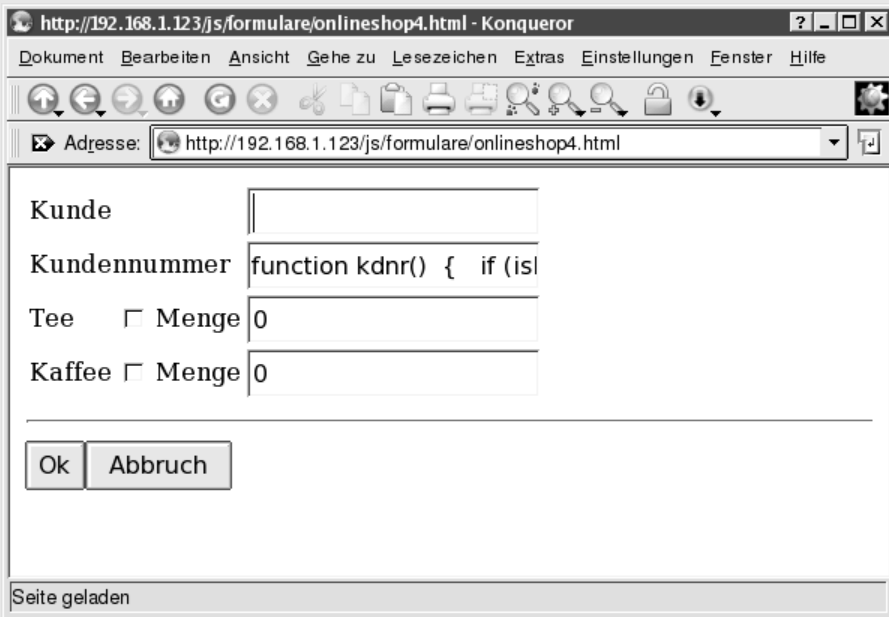


Abbildung 187: Nach einem Fehler zeigt der Konqueror einen Teil des Quelltextes an.

Im Netscape Navigator beziehungsweise Mozilla ist die Situation allerdings schlimmer. Es treten verschiedene Fehler auf, die sich verschiedenartig äußern, aber auf ähnlichen Gründen basieren. Der Netscape Navigator in neueren Versionen als auch Mozilla bauen die Webseite erst gar nicht auf. Wenn Sie sich die JavaScript-Konsole ansehen, ist die Variable `fehler` nicht definiert, und das Skript zum Generieren der Webseite bricht ab.

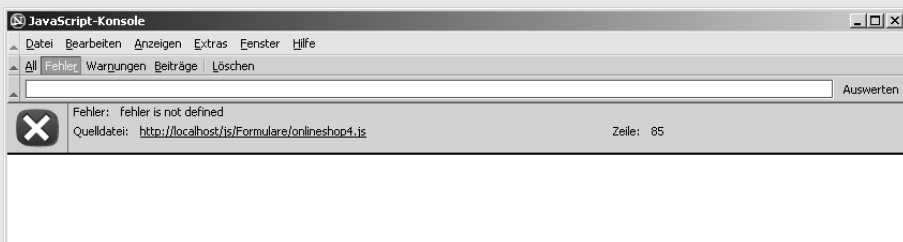


Abbildung 188: In der JavaScript-Konsole des Navigators 7.1 wird bereits beim ersten Schreiben der Webseite behauptet, die Variable `fehler` wäre nicht definiert.

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.

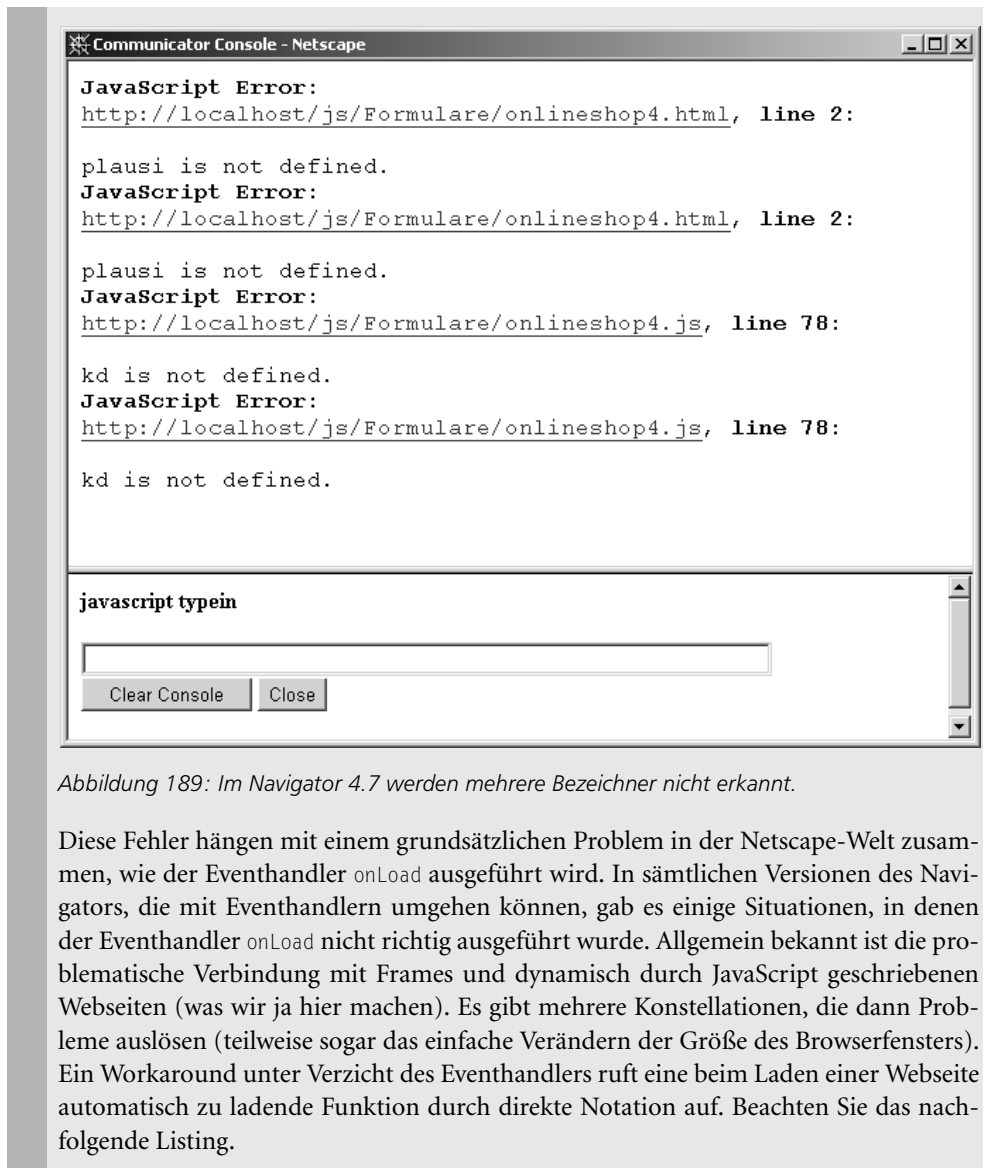


Abbildung 189: Im Navigator 4.7 werden mehrere Bezeichner nicht erkannt.

Diese Fehler hängen mit einem grundsätzlichen Problem in der Netscape-Welt zusammen, wie der Eventhandler `onLoad` ausgeführt wird. In sämtlichen Versionen des Navigators, 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 des Eventhandlers ruft eine beim Laden einer Webseite automatisch zu ladende Funktion durch direkte Notation 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 HTML-Datei ist wieder einfach und sieht so aus (*onlineshop4a.html*):

```
01 <html>
02 <script src="onlineshop4a.js"></script>
03 <script language="JavaScript">
```

Listing 269: Die extrem kleine HTML-Datei für den plausibilisierten Online-Shop (*onlineshop4a.html*)

```

04 <!--
05   generiereFormular("", "", "", "");
06 //-->
07 </script>
08 <body>
09 </body>
10 </html>

```

Listing 269: Die extrem kleine HTML-Datei für den plausibilisierten Online-Shop (onlineshop4a.html) (Forts.)

Die Datei ruft im Gegensatz zur ersten Variante die Funktion `generiereFormular()` nicht mit dem Eventhandler `onLoad` beim `<body>`-Tag auf, sondern als normalen JavaScript-Aufruf in einem zusätzlichen `<script>`-Container (Zeile 5). Beachten Sie, dass die Funktion `generiereFormular()` immer noch in dieser externen Datei definiert ist (diese heißt hier `onlineshop4a.js`). Sie hat aber in dieser neuen Version vier Übergabewerte.

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 }

```

Listing 270: Die neue externe JavaScript-Datei `onlineshop4a.js` mit sämtlichen Funktionalitäten von dieser Variante des plausibilisierten Online-Shops

```

29 function teean () {
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   }
45   meldung2 =
46     "Wenn Sie das Produkt Tee bestellen, m&uuml;ssen Sie eine g&uuml;ltige 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, m&uuml;ssen Sie eine g&uuml;ltige ↵
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 {

```

Listing 270: Die neue externe JavaScript-Datei onlineshop4a.js mit sämtlichen Funktionalitäten von dieser Variante des plausibilisierten Online-Shops (Forts.)

Grund-
lagen

Core

Formu-
lareAus-
nahme

Arrays

Strings

Datum

Platt-
form

DHTML

Events

```

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><body><form
name="shop" action="bestellung.php" method="post" onSubmit="return
plausi()"><table><tr><td>Kunde</td><td colspan="3" align="right"><input name="Kd"
maxlength=40>');
88     document.write('</td><td>'+ meldung0+ '</td></tr>');
89     document.write('<tr><td colspan="3">Kundennummer</td><td align="right"><input
name="Kdnr"></td>');
90     document.write('</td><td>'+ meldung1+ '</td></tr>');
91     document.write('<tr><td>Tee</td><td><input type="Checkbox" name="tee"
value="Tee"></td><td>Menge</td><td><input name="teeanz" value=0></td>');
92     document.write('</td><td>'+ meldung2+ '</td></tr>');
93     document.write('<tr><td>Kaffee</td><td><input type="Checkbox" name="kaf"
value="Kaffee"></td><td>Menge</td><td><input name="kafanz" value=0></td>');
94     document.write('</td><td>'+ meldung3+ '</td></tr>');
95     document.write('</table><hr><input type="Submit" value="Ok"><input type="reset"
value="Abbruch"></form></body></html>');
96     document.close();
97 }

```

Listing 270: Die neue externe JavaScript-Datei onlineshop4a.js mit sämtlichen Funktionalitäten von 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 88, 90, 92 und 94), 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 weitergehende Informationen, die in einer Erweiterung des Rezepts eventuell verwendet werden könnten. Sie können sich also je nach Notwendigkeit einen der beiden Wege herausuchen.

Das Beispiel ist jetzt auch in neueren Versionen des Navigators und dem Mozilla voll funktionsfähig.

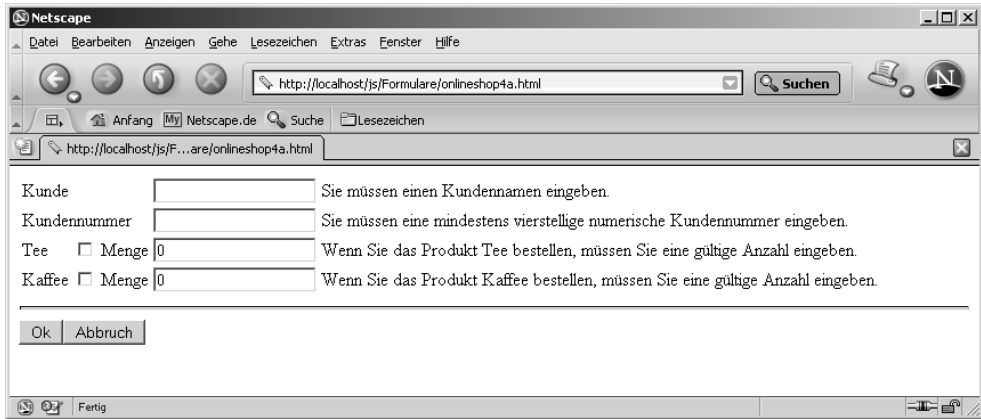


Abbildung 190: Jetzt kommen auch neuere Versionen des Navigators und des Mozilla zurecht.

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

Zum Abschluss soll unser Online-Shop die Fehlermeldungen noch mit Unterstützung von DHTML (Dynamic HTML) anzeigen. Dazu positionieren wir die Fehlermeldungen dynamisch mit Style Sheets und JavaScript. DHTML umfasst im Kern die Verbindung von HTML, Style Sheets und JavaScript beziehungsweise Aufgabenstellungen, die sich darum ranken. Diese Umsetzung hat einen sehr großen Reiz und ist auch äußerst elegant. Nur leider funktioniert sie nicht in allen Browsern⁷⁶ und benötigt im Grunde für eine zuverlässige Gestaltung des Layouts eine absolute Positionierung sämtlicher Bestandteile der Webseite mit Style Sheets. Soweit soll aber dieses kleine Beispiel nicht gehen. Wir wollen bei der Webseite auf ein reines HTML-Design zurückgreifen und nur die Fehlermeldungen mit DHTML positionieren. Dennoch – diese Lösung ist sehr elegant und wirkungsvoll sowie sehr flexibel.

Beispiel (*onlineshop4.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;

```

Listing 271: Ein Online-Shop mit Fehleranzeige über DHTML

76. Insbesondere der Netscape Navigator beziehungsweise Mozilla machen große Probleme. Opera, Konqueror oder Internet Explorer kommen aber damit hervorragend zurecht.

```
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";
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 teean () {
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         }
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></noscript>
108 <body>
109 <form name="shop" action="bestellung.php" method="post"
110 onSubmit="return plausi()">
111 <span class="p0" id="m1">
112     Sie müssen einen Kundennamen eingeben.
113 </span>

```

Listing 271: Ein Online-Shop mit Fehleranzeige über DHTML (Forts.)

Grund-
lagen

Core

Formu-
lareAus-
nahme

Arrays

Strings

Datum

Platt-
form

DHTML

Events

```

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
119 eingeben.
120 </span>
121 <span class="p0" id="m4">
122 Wenn Sie das Produkt Kaffee bestellen, m&uuml;ssen Sie eine g&uuml;ltige Anzahl
123 eingeben.
124 </span>
125 <table>
126 <tr>
127 <td>Kunde</td>
128 <td colspan=2><input name="Kd" maxlength=40></td>
129 <td></td>
130 <td colspan=2></td>
131 </tr>
132 <tr>
133 <td>Kundennummer</td>
134 <td colspan=2><input name="Kdnr"></td>
135 <td></td>
136 <td colspan=2></td>
137 </tr>
138 <tr>
139 <td>Tee</td>
140 <td><input type="Checkbox" name="tee" value="Tee"></td>
141 <td>Menge</td>
142 <td><input name="teeanz" value=0></td>
143 <td></td>
144 <td></td>
145 </tr>
146 <tr>
147 <td>Kaffee</td>
148 <td><input type="Checkbox" name="kaf" value="Kaffee"></td>
149 <td>Menge</td>
150 <td><input name="kafanz" value=0></td>
151 <td></td>
152 <td></td>
153 </tr>
154 <tr>
155 <td colspan=2><input type="Submit" value="Ok">
156 <input type="reset" value="Abbruch">
157 </td>
158 </tr>
159 </table>
160 </form>
161 </body>
162 </html>

```

Listing 271: 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 HTML 4.0 in HTML verankert 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 Sheets mit Positionsangaben. Dabei bezeichnen `.p0` bis `.p4` so genannte CSS-Klassen. Dies liegt an dem vorangestellten Punkt. Allgemein spezifizieren Style Sheets einen der existierenden HTML-Tags. Dann wird dieser ohne vorangestellten Punkt notiert. Wenn Sie einem Element eine Stilinformation zuordnen, schreiben Sie diese in geschweifte Klammern hinter das Element oder die Klasse. Das geht für eine HTML-Überschrift der Ordnung 1 beispielsweise so:

```
h1 { color : white }
```

Listing 272: Eine CSS-Definition für eine Überschrift 1. Ordnung

Die Syntax von einer CSS-Deklaration sieht immer gleich aus. Es gibt jedes Mal den folgenden Aufbau:

```
[name] : [wert]
```

Listing 273: Eine schematische CSS-Deklaration

Dabei gibt `name` das zu formatierende Detail an und `wert` die konkrete Stilinformation. Beachten Sie, dass dazwischen ein Doppelpunkt und kein Gleichheitszeichen steht! Bei Bedarf werden mehrere Deklarationen mit Semikolon getrennt.

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 274: 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 ist mit einem Minuszeichen versehen. 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 Webbrowser auch keine (unerwünschten) Bildlaufleisten an. Die Klasse `.p0` spezifiziert also eine Position, auf der Fehlertexte zwischengelagert werden können, ohne dass ein Anwender sie 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 HTML-Element erfolgt über die Erweiterung eines beliebigen HTML-Container-Tags um den Parameter `class = "[Name der Klasse]"`.

Wenn man beliebigen Inhalt einer Webseite in einen Container einpacken will, bietet sich ideal der `<span>`-Container an. Dieser zeichnet sich dadurch aus, dass er keinerlei HTML-Eigenwirkung hat. Sie können ihn sich als eine Art Klarsichthülle vorstellen, in die ein Teil der Webseite eingepackt wird. Die ausschließliche Bedeutung des `<span>`-Containers ist, Angriffspunkte für Fremdwirkungen wie Style Sheets bereitzustellen. Beachten Sie die Zeilen 111 bis 113 (`<span class="p0" id="m1">Sie müssen einen Kundennamen eingeben.</span>`). Das ist genau so ein `<span>`-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 `<span>`-Container.

Jetzt sollte Ihnen auffallen, dass die `<span>`-Container jeweils noch einen weiteren Parameter enthalten. Den `id`-Parameter, der im Mittelpunkt der dynamischen Veränderung von Webseiten über JavaScript steht. Darüber kann in den meisten Browsern JavaScript ein Element in einer Webseite identifizieren. Über `[Name der Klasse].className` 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 275: Die Funktion kdnr()

Der JavaScript-Zugriff über `className` steht in der Netscape-/Mozilla-Welt sowie einige Exoten und alten Browsern nicht zur Verfügung. Deshalb ist dieses Rezept explizit nicht für den Navigator oder den Mozilla geeignet. Die resultierenden Fehler unterscheiden sich zwar ja nach Version, aber die Details dazu nützen uns nichts. Allerdings ist es relativ einfach, mit `document.getElementById()` den Zugriff so umzubauen, dass er für diese Browser meistens funktioniert. Dazu finden Sie im Kapitel zu DHTML Rezepte.

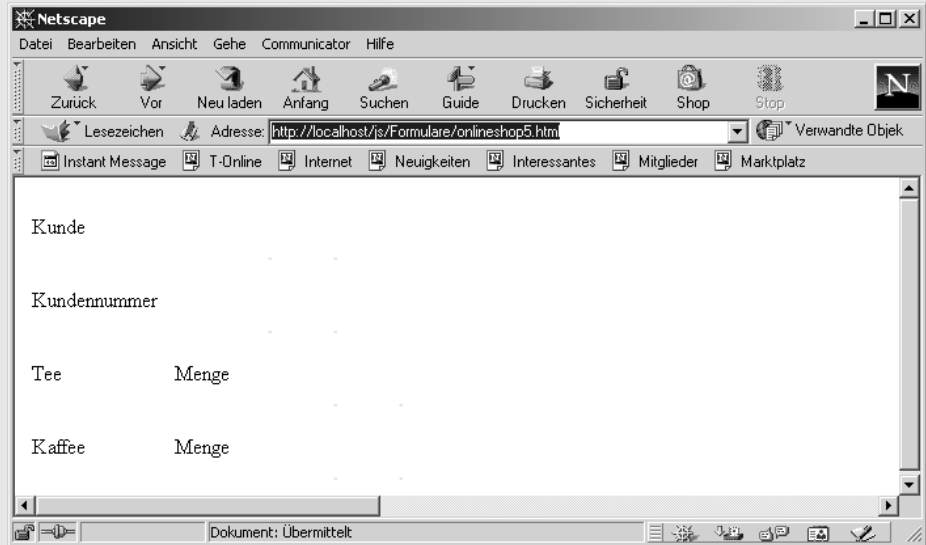


Abbildung 191: Der Navigator 4.7 verschluckt sogar das HTML-Formular.

Falls Sie diese Version nutzen wollen, müssen Sie zwingend eine Browserweiche voranstellen.

In Zeile 46 wird dem Element `m2` die neue CSS-Klasse `.p2` zugewiesen. Das bedeutet, der `<span>`-Container, der mit `id="m2"` gekennzeichnet ist (Zeile 114 bis 116 – `<span class="p0" id="m2">Sie müssen eine mindestens vierstellige numerische Kundennummer eingeben.</span>`), 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.

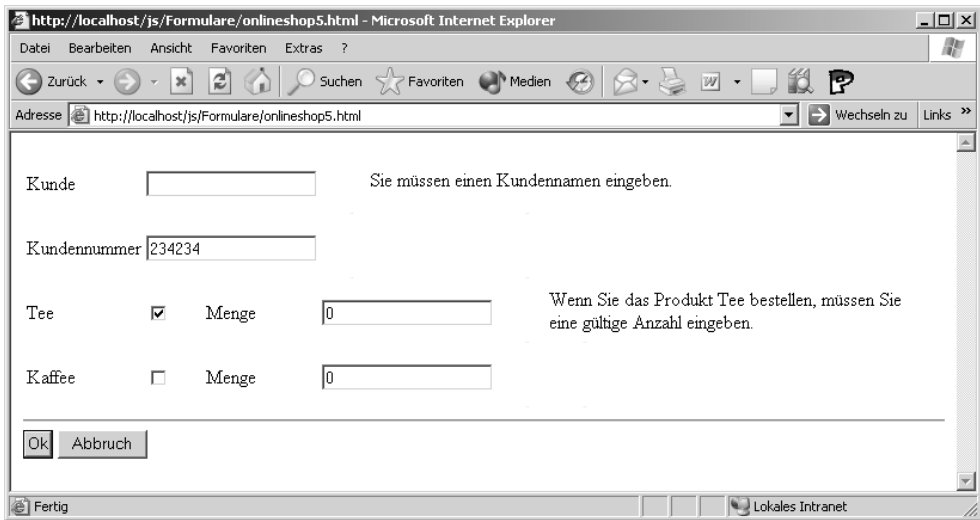


Abbildung 192: Fehleranzeige mit der Positionierung der Fehlermeldungen über DHTML – so wie es sein sollte

Achtung

Eine Positionierung mit Style Sheets erzwingt im Grunde, 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.

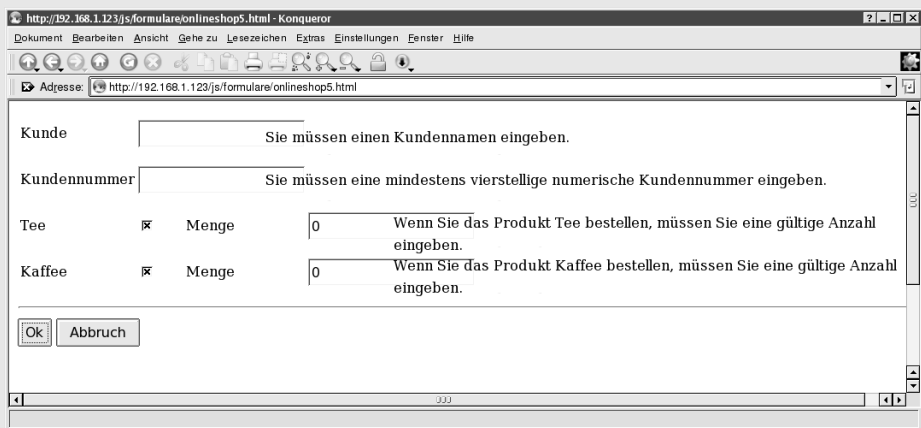


Abbildung 193: Das Layout im Konqueror, wenn dessen Grundkonfiguration nicht so eingestellt ist wie in dem Browser, in dem Sie das Layout getestet haben – die Funktionalität ist aber voll unterstützt.

114 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 hingewiesen⁷⁷ und dabei dennoch betont, dass man sich jedoch nicht auf die Plausibilisierung durch JavaScript verlassen kann. Immerhin gibt es noch (antiquierte) Browser ohne JavaScript-Unterstützung, und 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* (nahezu⁷⁸) 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 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 das Abschicken der Formulardaten nicht mit dem normalen Submit-Button erfolgen darf, sondern man muss 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.

Dynamisches Erstellen des Webformulars beim Laden einer Webseite

Der zweite Schritt in Hinsicht auf Erschwerung einer unplausibilisierten Absendung der Formulardaten ist das dynamische Schreiben des Webformulars beziehungsweise der gesamten Webseite mit JavaScript (per `document.write()`). Wenn beim Laden einer Webseite kein JavaScript aktiviert ist, bekommt ein Anwender gar kein Formular 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 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.

77. Der Umfang der Rezepte in diesem Kapitel zeigt dies ja ebenfalls.

78. Wenn ein Besucher einer Webseite richtig Ahnung hat und sich Zeit nimmt, lässt sich das 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.

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 Browsern⁷⁹ 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 haben Sie ja auch einen Halbexperten an der Hackerfront, dem dieser im Grunde einfache Schritt gar nicht bekannt ist. Auf jeden Fall wird der Aufwand weiter erhöht⁸⁰.

Verschleiern von Code

Die Sicherheit von Formularen wird auch dadurch erhöht, dass Sie möglichst kryptisch programmieren. Das machen einige Programmierer schon von Haus aus⁸¹, 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 aufgehoben⁸², 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 Interpretertechniken 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 ziehen, 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 etwa 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;
```

Listing 276: Einbinden einer externen Datei, deren Name berechnet wird

79. Etwa dem Internet Explorer

80. Wenngleich minimal, aber die Summe der Kleinigkeiten macht dem Hacker einfach Ärger.

81. ;-))

82. Etwa alle nicht unbedingt notwendigen Zeilenumbrüche, Leerzeichen, Tabulatoren etc. beseitigt.

```
05 a = 5 * ("1" + 111) + ++a + (a-- % 13);
06 document.write("<script src='e' + a + '.js'></script>");
07 //-->
08 </script>
09 <noscript></noscript>
10 <body text="#000000" bgcolor="#FFFFFF" link="#FF0000" alink="#FF0000"
vlink="#FF0000">
11 </body>
12 </html>
```

Listing 276: Einbinden einer externen Datei, deren Name berechnet wird (Forts.)

In Zeile 5 wird der zentrale Teil von dem Namen der externen Datei berechnet. Ist Ihnen der Name klar? Falls nicht, sehen Sie den Erfolg so einer Maßnahme, die natürlich vielfach komplexer sein kann.

Tip

Ein einfaches `alert(a)` zeigt natürlich den Wert von `a` und damit den fehlenden Teil des Dateinamens an. Aber in einer komplexeren Struktur diesem Verfahren erst einmal auf die Spur zu kommen, ist nicht trivial.

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 im URL in der Adresszeile im Browser⁸³, 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 übermittelnden Formulardaten aufgefüllt wird. Dann wird der generierte Pseudo-URL als String abgeschickt (siehe Seite 172 mit den Ausführungen zum Formularcontainer). Diesen String kann ein Anwender aber auch manuell in der Adresszeile des Browsers eingeben, ohne Ihr Formular zu verwenden⁸⁴. Dann kann er natürlich gewünschte Werte vollkommen unabhängig von irgendwelchen Plausibilisierungen eines Webformulars direkt in den 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 etwa 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) jedoch dieses Feature zur Absicherung auch in neuen Versionen wieder deaktivieren⁸⁵, und dann haben Sie auf

83. Das ist hier nur Nebensache.

84. 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 den Pseudo-URL.

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 Browser die Daten dennoch als indirekten GET-Versand zum Server schicken, und dieser nimmt sie einfach entgegen. Hier hilft nur, den Server selbst sicher zu machen⁸⁵ oder den Provider zu wechseln, wenn dieser so eine Sicherheitslücke nicht dicht macht.

Achtung

Jetzt weiß ein etwas gewiefterer Anwender, dass ein Formular natürlich gefakt 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). 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 schon ziemlich Ahnung 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 gefakten 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.

85. Um etwa die Programmierung in PHP etwas zu erleichtern – immerhin sind viele PHP-Programmierer Laien, denen das Sicherheitsfeature nur schwer erklärt werden kann. Zudem müssten bei einer Umstellung auf die sichere Variante sämtliche betroffenen Altskripte angepasst werden.

86. Wenn Sie dazu in der Lage sind.

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, die eine unmittelbare Reaktion erzwingen. Diese können entweder durch den JavaScript-Interpreter automatisch ausgeworfen werden oder aber von dem Programmierer selbst, wenn es notwendig erscheint.

115 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 eingeben. Sei es um auszuprobieren, was denn 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 im Kapitel zu Formularen 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 277: Ein Skript mit einem Problem in Zeile 4

In dem Skript werden mit der JavaScript-Toplevelfunktion `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. In Zeile 3 kommt nun ein vernünftiges Ergebnis heraus, und das wird im Anzeige-

-
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 :-)). Genauer ist im Buch »Per Anhalter durch die Galaxis« zu finden.

bereich 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 194: Nach der ersten Ausgabe entsteht ein Problem.

Solche Probleme in einem Skript müssen aber nicht zwangsläufig zum Abbruch eines Skriptes führen. 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 Programmsituationen 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. 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 ein Zugriffsversuch auf ein Objekt, das noch gar nicht erzeugt wurde. Aber auch selbst definierte Ausnahmesituationen sind denkbar. Ausnahmen sind allgemein ein leistungsfähiges und fle-

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.

4. Das muss keine negativ belastete Situation sein.

xibles Instrument, um bestimmte Reaktionen eines Skriptes unabhängig von einem vorgegebenen Zeitpunkt sicherzustellen. Sie wollen beispielsweise eine bestimmte Benutzereingabe erzwingen und das Skript anhalten, solange diese nicht erfolgt ist.

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 Programme sind 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)⁵. 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. Allerdings muss man beachten, dass die Ausnahmebehandlung in JavaScript nicht zu weit implementiert ist, wie es etwa in Java der Fall ist, und zudem die verschiedenen Browser, welche offiziell mit Ausnahmen zurechtkommen, teilweise sehr unterschiedlich damit umgehen.

Grundsätzlich ist eine Ausnahmebehandlung auf dem objektorientierten Konzept aufgesetzt, denn eine Ausnahme ist als Objekt zu verstehen. Wenn eine Ausnahme erzeugt wird, wird von dem Laufzeitsystem ein Objekt mit Informationen generiert, welches 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 suchen. 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.

Ausnahme	Beschreibung
<code>EvalError</code>	Die Toplevelfunktion <code>eval([Zeichenkette])</code> 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 meisten Browser nicht zuverlässig.

Tabelle 6: Typen von Ausnahmeobjekten in ECMAScript beziehungsweise JavaScript

5. Was sicher auch eine Erklärung dafür ist, dass sich das Ausnahmekonzept noch nicht in allen neuen Browsern etabliert hat. Viele kritische Aktionen wie der Zugriff auf Datenträger sind ja aus JavaScript heraus nicht möglich.

Ausnahme	Beschreibung
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 tritt auf, 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 6: Typen von Ausnahmeobjekten in ECMAScript beziehungsweise JavaScript (Forts.)

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.

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 278: Die Struktur von `try-catch`

Alle potenziell kritischen Anweisungen stehen in dem `try`-Zweig. Der Ausdruck im `catch`-Zweig ist das konkrete Fehlerobjekt, und in dem `catch`-Zweig stehen einfach innerhalb geschweifeter Klammern die Anweisungen, die beim Auftreten der Ausnahme durchgeführt werden sollen.

Hinweis

Was Sie nun im `catch`-Teil konkret tun, ist nicht weiter festgelegt. Sie können jede Ihnen sinnvoll erscheinende Maßnahme ergreifen, die das Skript sinnvoll 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 Fokussieren des kritischen Eingabefeldes.

Kritische Aktionen in einem JavaScript sollten also immer innerhalb des `try`-Blocks durchgeführt werden, sofern man sicherstellen kann, dass der Besucher einen Browser verwendet, der mit Ausnahmebehandlung zurechtkommt. Zur Sicherheit sollten Sie aber auf jeden Fall eine Browserweiche voranstellen.

Der Begriff *try* sagt übrigens auch bereits sehr treffend, was dort 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).

Hinweis

Das Auftreten einer Ausnahme innerhalb eines `try-catch`-Konstruktes führt nicht dazu, dass das Skript beendet wird. Es läuft mit den Anweisungen hinter dem Konstrukt weiter (was ja ein wesentlicher Zweck der Behandlung einer Ausnahme ist).

Das nachfolgende Beispiel zeigt den Fall, in dem eine Ausnahme auftritt (hier eine einfache Situation, wo 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. Im `catch`-Teil wird das dabei erzeugte `Error`-Objekt aufgefangen⁶. 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>");
```

Listing 279: Eine Ausnahme wird erzeugt und abgefangen.

6. Sie könnten auch das speziellere `EvalError`-Objekt auffangen.

```
06 document.write("Das ist noch im try-Block - nach der kritischen Situation");
07 }
08 catch(Error) {
09 alert(Error.toString());
10 alert(Error.message);
11 }
12 document.write("Das kommt trotz Ausnahme");
13 </script>
14 </html>
```

Listing 279: Eine Ausnahme wird erzeugt und abgefangen. (Forts.)

Die beiden `eval()`-Anweisungen sowie die reine Textausgabe in Zeile 4, 5 und 6 sind von einem `try`-Block umgeben⁷. 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 Ausgabe von Zeile 6 wird nicht ausgeführt. Der Programmfluss verzweigt unmittelbar in den `catch`-Block. In Zeile 9 wird der Wert von dem generierten `Error`-Objekt angezeigt.

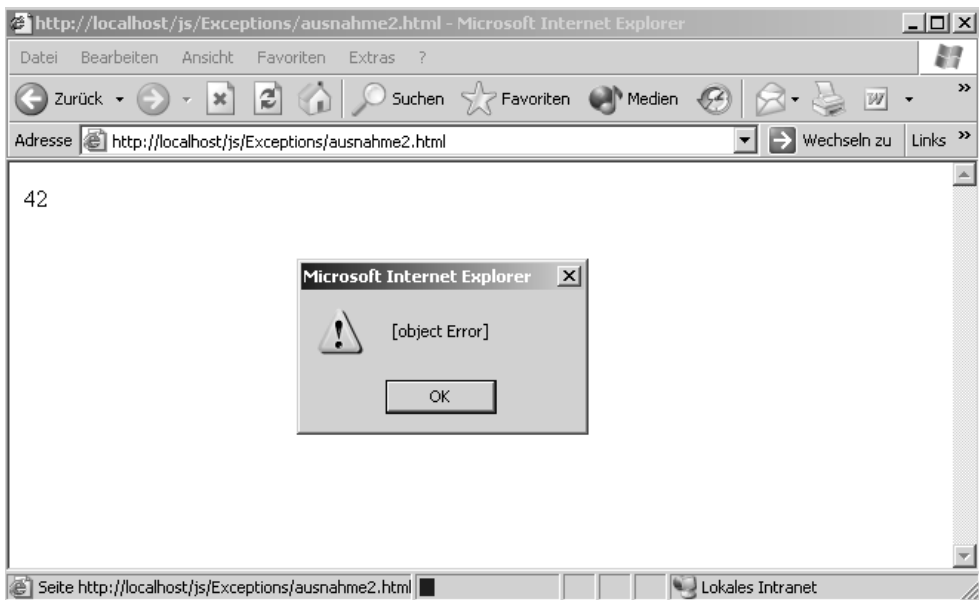


Abbildung 195: Die Ausnahme ist aufgetreten und wird abgefangen. Mit `toString()` kann das Objekt selbst ausgegeben werden.

7. 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`-Konstruktes stehen, soll nur zeigen, dass natürlich auch unkritische Anweisungen in einem `try`-Block stehen können und zudem das Verhalten nach Auftreten einer Ausnahme demonstrieren.

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.



Abbildung 196: Das zeigt der Navigator an.

In Zeile 10 wird die integrierte Fehlermeldung von dem `Error`-Objekt ausgegeben.



Abbildung 197: Die Standardmeldung des Ausnahmeobjekts – mit der Eigenschaft `message` abgefragt

Wenn der `catch`-Block abgearbeitet wurde, läuft das Skript mit der nächsten Anweisung hinter dem `try-catch`-Konstrukt weiter.

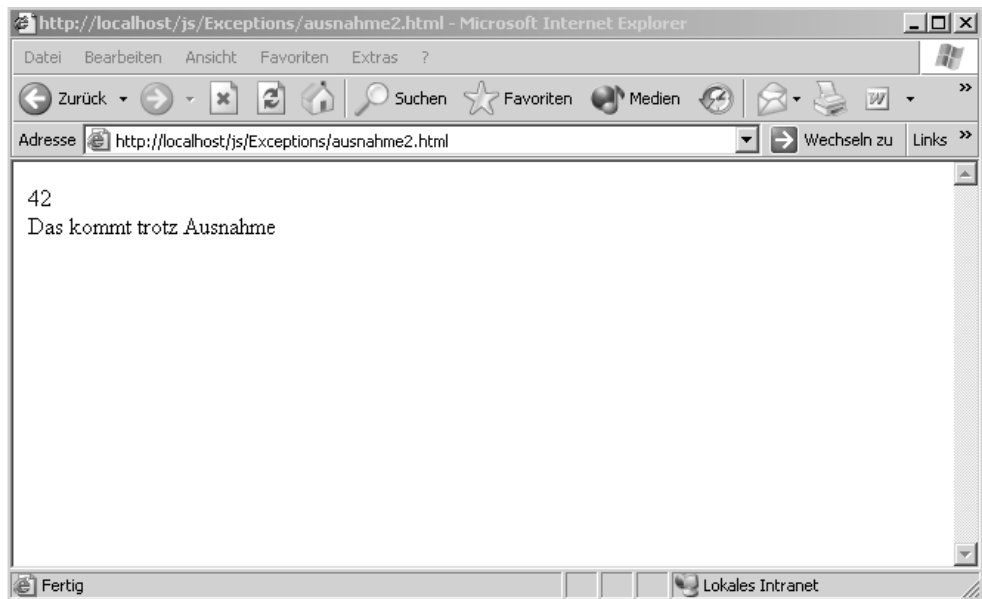


Abbildung 198: Nach der Ausnahme läuft das Skript weiter.

Wenn Sie ein Skript mit Ausnahmebehandlung in einen Browser laden, der diese Technik nicht unterstützt, wird der Skriptinterpreter mit einem Fehler reagieren.

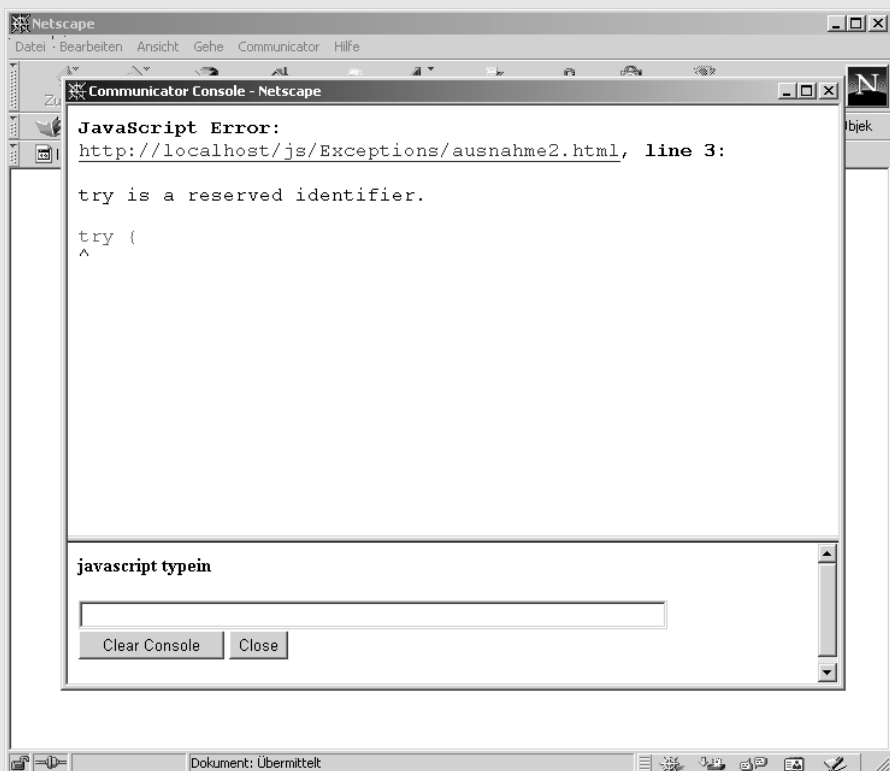


Abbildung 199: Der Navigator 4.7 meldet einen JavaScript-Fehler, da er keine Ausnahmebehandlung kann.

Die Verwendung von 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 JavaScript1.4 angeben, und damit blockieren Sie beispielsweise den Internet Explorer 6 und auch den Opera 7, obwohl diese Browser hervorragend mit Ausnahmebehandlung zurechtkommen. 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.

Wenn ein `try`-Block notiert wird, muss zwingend ein `catch`-Block folgen, der diese auffängt. Während die ersten Browser mit Unterstützung des Ausnahmekonzeptes noch eine Fehlermeldung beim Fehlen des `catch`-Blocks anzeigten, zeigen neuere Browser dummerweise gar nichts mehr an. Der `try`-Block wird einfach ignoriert.

116 Wie kann ich mehrere Ausnahmen behandeln?

Ein Verschachteln von `try-catch`-Strukturen ist möglich (allerdings in JavaScript nicht so oft nötig wie in Techniken wie Java), wobei dann auch außerhalb angesiedelte `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 Routine vorhanden, wird die Suche in der Aufrufhierarchie noch oben fortgesetzt. 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 potenzielle Probleme umgebenden, Struktur.

Achtung

Die andere Version der Behandlung von mehreren Ausnahmen notiert einfach auf der gleichen Ebene mehrere `catch`-Blöcke. Das sieht schematisch so aus:

```
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 280: 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.

117 Wie kann ich selbst definierte Ausnahmen erzeugen und verwenden?

Man kann auch in JavaScript daraus selbst definierte Ausnahmen erzeugen. `Error` kann sowohl als Funktion über `Error()` als auch als Konstruktor über `new Error()` verwendet werden und erzeugt damit ein neues `Error`-Objekt.

Grundlagen

Core

Formulare

Ausnahmen

Arrays

Strings

Datum

Plattform

DHTML

Events

Beispiel:

```
a = new Error();
```

Listing 281: Erzeugen einer selbst definierten Ausnahme als Ableitung von Error

Interessant ist dabei, dass man mit `Error([Meldung])` eine selbstdefinierte 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 282: Erzeugen einer selbstdefinierten Ausnahme mit Meldung und spätere Ausgabe mit alert()

Allerdings ist es mit der Erzeugung von selbstdefinierten Ausnahmen alleine nicht getan. Diese müssen auch zu einer spezifizierten Situation ausgeworfen werden. Und auch das müssen Sie selbst angeben⁸.

Ausnahmen können also nicht nur automatisch vom System ausgeworfen werden. Sie sind ebenfalls in der Lage, selbst in einer Funktion eine Ausnahme auswerfen, 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 Ausnahme-Objekt. Verwandt ist die Situation zu einer `return`-Ausweisung.

Die Syntax sieht so aus:

```
throw [Ausdruck];
```

Listing 283: 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 einzig sinnvolle Weg.

Beispiel:

```
throw new Error("Na, na, na");
```

Listing 284: Auswerfen einer Ausnahme

Wenn Sie nun eine oder mehrere selbstdefinierte Ausnahmen von `Error` ableiten, können Sie allgemein `Error` abfangen (wenn man sehr allgemein reagieren will) oder aber jedes davon

⁸ Darauf beruht auch der gesamte Nutzen von selbstdefinierten Ausnahmen.

abgeleitete Objekt. Das nächste Beispiel, was 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 285: 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 Funktio-

Grund-
lagen

Core

Formu-
lare

Aus-
nahme

Arrays

Strings

Datum

Platt-
form

DHTML

Events

nen die selbstdefinierten 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.



Abbildung 200: Die selbstdefinierten Ausnahmen sind nicht aufgetreten, aber die Ausnahme vom Typ `Error` wurde ausgeworfen.

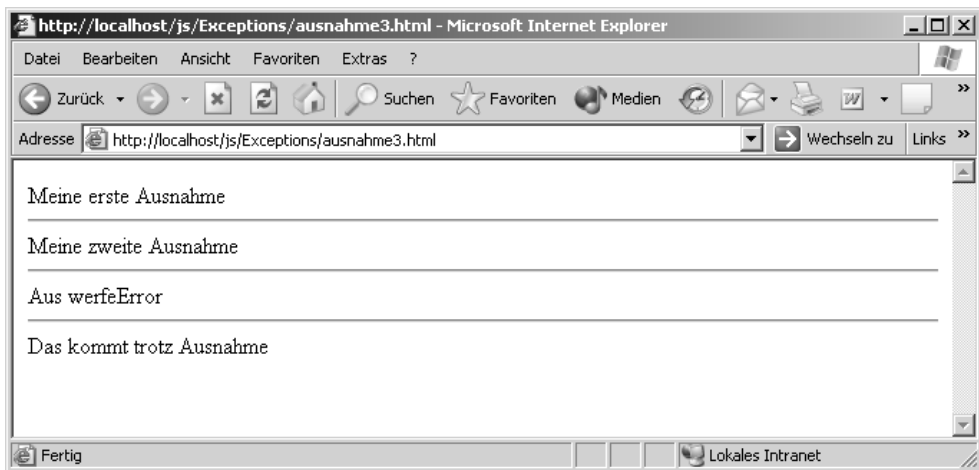


Abbildung 201: Hier liegt genau der umgekehrte Fall vor – beide selbstdefinierten Ausnahmen wurden ausgeworfen, aber nicht die Ausnahme vom Typ `Error`.

Auch wenn Sie mit einer selbstdefinierten Ausnahme (zum Beispiel mit einer eigenen Fehlermeldung) arbeiten, können Sie diese pauschal über die Superklasse (`Error`) abfangen. Das lässt den Code erheblich verkürzen, wenn das Auftreten irgendeiner Ausnahme sofort zu einem Behandeln veranlassen soll.



Abbildung 202: Alle drei Ausnahmen wurden ausgeworfen – hier läuft das Beispiel im Navigator 7.1.

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 286: Die Funktionen werfen verschiedene Ausnahmen aus, aber es gibt nur ein try-catch.

Grundlagen

Core

Formulare

Ausnahmen

Arrays

Strings

Datum

Plattform

DHTML

Events

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.

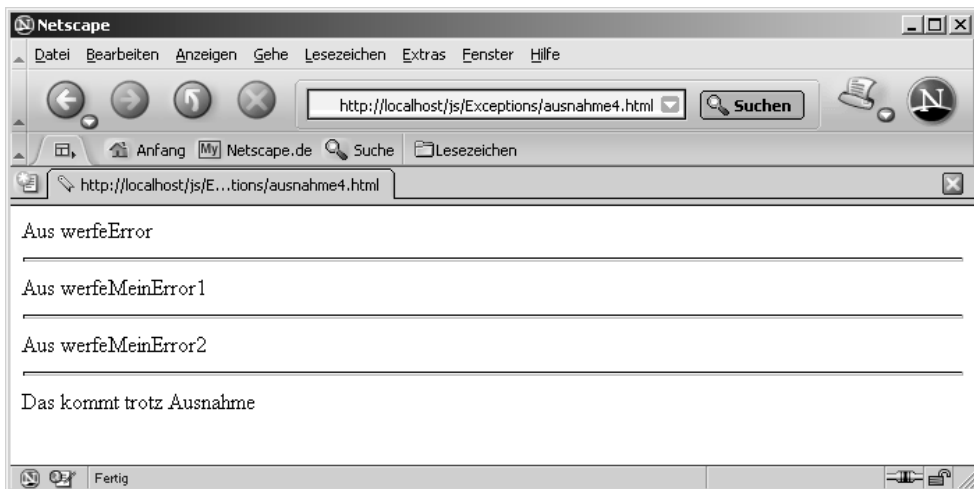


Abbildung 203: Keine Ausnahme – alle Meldungen nach dem optionalen `throw` sind zu sehen.

Wenn jedoch eine Ausnahme auftritt, werden die noch folgenden Funktionsaufrufe im `try`-Block nicht erreicht.

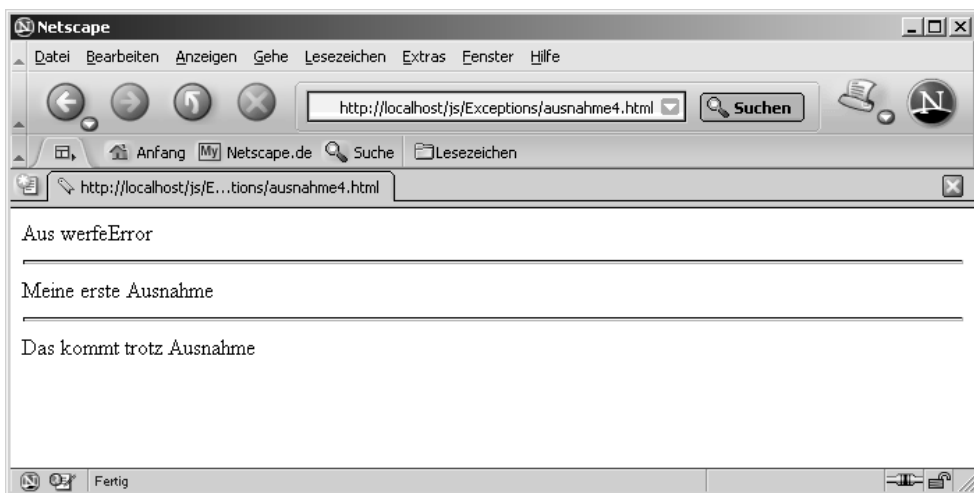


Abbildung 204: Die erste selbst definierte Ausnahme wurde ausgeworfen.

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. Das nachfolgende Beispiel soll nun auch mit verschiedenen Situationen arbeiten, die unterschiedliche Reaktionen bewirken. In einer Situation soll keine Ausnahme ausgeworfen werden, in zwei Situationen werden explizite Ausnahmeobjekte ausgeworfen und in einem Fall ein Literal. Um diese Situationen zu trennen, wird in dem Beispiel explizit der Operator `instanceof` angewandt, der eine Neuerung von JavaScript 1.4 beziehungsweise 1.5 ist und den Typ eines Objekts spezifiziert.

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 287: Je nach Situation werden verschiedene Ausnahmen ausgeworfen und entsprechend gehandhabt.

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

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 Ausnahmen 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 einen von den 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.

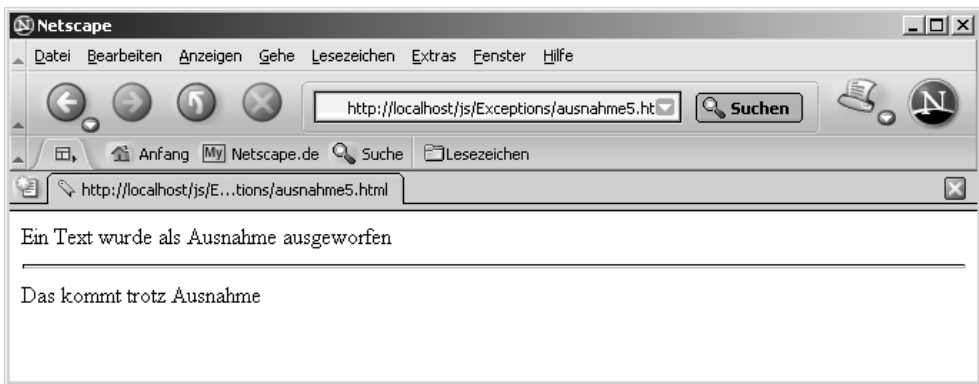


Abbildung 205: Ein String wurde ausgeworfen.

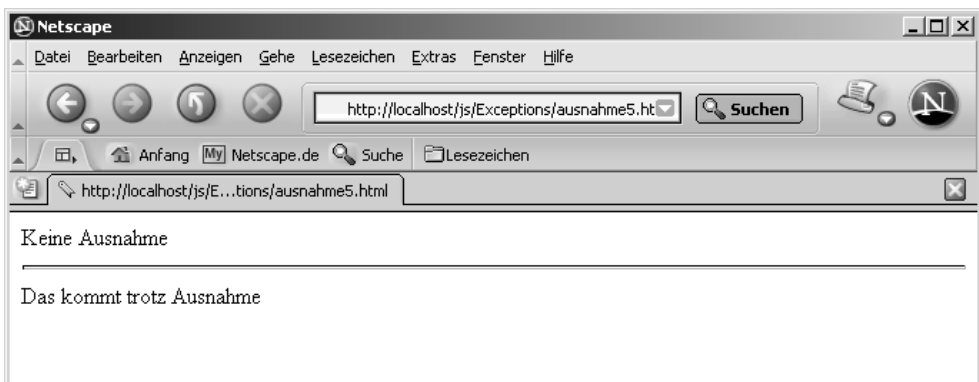


Abbildung 206: Es ist keine Ausnahme ausgeworfen worden.

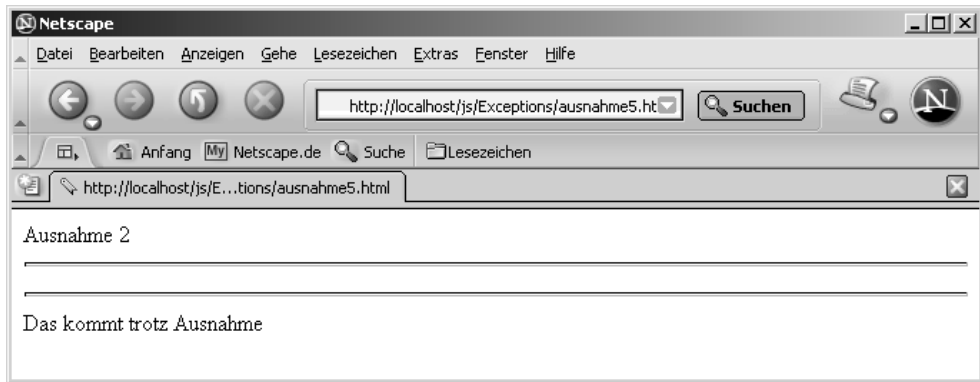


Abbildung 207: Die selbstdefinierte Fehlermeldung beim Auftreten von Ausnahme 2

118 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 trägt sicher auch zu der langsamen Verbreitung von Ausnahmebehandlungen in JavaScript 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. Die wahrscheinlich in der Praxis sinnvollste Anwendung von Ausnahmen ist jedoch 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.

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){
```

Listing 288: Eine selbstdefinierte Ausnahme zum Abfangen einer falschen Benutzereingabe in einem Webformular

```

16     document.f1.z1.focus();
17     alert(e.message);
18 }
19 }
20 //-->
21 </script>
22 <noscript></noscript>
23 <body text="#000000" bgcolor="#FFFFFF" link="#FF0000" alink="#FF0000"
vlink="#FF0000">
24 <form name="f1">
25 <h1>Geben Sie bitte zwei Zahlen ein</h1>
26 <table >
27 <tr>
28 <td>
29 Zahl 1
30 </td>
31 <td>
32 <input type="Text" name="z1">
33 </td>
34 </tr>
35 <td>
36 Zahl 1
37 </td>
38 <td>
39 <input type="Text" name="z2">
40 </td>
41 <tr>
42 <td>
43 Berechnung
44 </td>
45 <td>
46 <input type="button" value="ok" onClick="multipliziere()">
47 </td>
48 </tr>
49 </table>
50 </form>
51 </body>
52 </html>

```

Listing 288: Eine selbstdefinierte Ausnahme zum Abfangen einer falschen Benutzereingabe in einem Webformular (Forts.)

In diesem Webformular (Zeile 24 bis 50) 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()` finden Sie von Zeile 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 selbstdefinierte 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 208: Eine Ausnahme wird ausgeworfen.



Abbildung 209: Das Ergebnis der Multiplikation wird angezeigt.

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 289: 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 der Netscape Navigator diese Variante einwandfrei versteht.



Abbildung 210: Der Internet Explorer kommt mit der Syntax nicht zurecht.



Abbildung 211: Keine Probleme im Netscape Navigator

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

Datenfelder

In diesem Kapitel geht es um das Thema *Datenfelder* beziehungsweise *Arrays*. Ein Array oder Datenfeld ist eine Sammlung von Variablen beliebigen Inhalts¹ mit einem gemeinsamen Namen. Die einzelnen Elemente eines Datenfelds können über einen Index angesprochen werden. Objektfelder haben in JavaScript eine besondere Bedeutung, denn sie sind einmal Datenfelder, aber auch als Objekte zu verstehen, und 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 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 ein 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 einige 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, welche 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.
<code>links</code>	Die im Objektfeld enthaltenen Elemente repräsentieren eine Liste aller Hyperlinks in einer Webseite.

Tabelle 7: Einige wichtige Objektfelder, die unter JavaScript nutzbar sind

1. Das können primitive Datentypen als auch Objekte sein.

Objektfeld	Beschreibung
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 7: Einige wichtige Objektfelder, die unter JavaScript nutzbar sind (Forts.)

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.

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. Erst damit werden Dinge möglich, wie man sie beispielsweise bei der Nutzung von Objektfeldern in einer Webseite einsetzt.

119 Wie kann ich ein Datenfeld anlegen?

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 täuschen, wenn Sie in JavaScript viele Objekte scheinbar ohne Erzeugung verwenden können. Zahlreiche Objekte als 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 (beispielsweise die Anwendung der `split()`-Methode bei einem `String`-Objekt). Sie können aber auch ein Datenfeld selbst erstellen. Ein solches Datenfeld wird mit Hilfe des dafür zuständigen JavaScript-Schlüsselworts erzeugt – `new`. Nachgestellt wird der so genannte *Konstruktor*³ der `Array`-Klasse notiert. Und zwar mit folgender Syntax:

2. Auch das kann passieren. Etwa ist das in Java mit dem `String`-Objekt möglich.

3. Das ist eine spezielle Methode in der OOP, deren Aufgabe die Erzeugung von Objekten ist.

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

Listing 290: 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 291: Anlegen eines Datenfelds mit fünf 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 292: Ein Test, ob ein Datenfeldelement existiert

Alternativ können Sie die Datenfeldelemente bei der Erzeugung bereits vorbelegen:

```
[Datenfeldbezeichner] = new Array([Vorbelegungswerte]);
```

Listing 293: Anlegen eines Datenfelds mit einem Vorbelegungswert

Beispiel:

```
a = new Array(3, 4, "zu gleich");
```

Listing 294: Anlegen eines Datenfelds mit drei 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 295: Anlegen eines Datenfelds mit Elemente

Beispiel:

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

Listing 296: Anlegen eines Datenfelds ohne Elemente

120 Wie greife ich auf ein Datenfeld zu?

Wenn Sie beim Anlegen eines Datenfelds eine Referenz auf dieses einer Variablen 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 297: Anlegen eines Datenfelds mit fünf 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) die Eigenschaft `length`.

Beachten Sie 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 477.



Abbildung 212: Zugriff auf das Array

121 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 darauf eine Referenz in einer Variablen zu legen. 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 den Konstruktor 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 298: Anlegen eines anonymen Datenfelds mit fünf Elementen und Ausgabe der Größe

In Zeile 3 legen wir ein Datenfeld an und geben direkt über einen anonymen Zugriff die Größe aus.

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 etwa in mächtigeren Programmiersprachen wie Java hat.

122 Wie greife ich auf die Elemente in einem Datenfeld zu?

Wenn Sie beim Anlegen eines Datenfelds eine Referenz auf dieses einer Variablen zuweisen, kommen Sie über diese Variable an das Datenfeld selbst. Darüber haben Sie Zugang zu den Eigenschaften und Methoden eines Datenfelds. Aber damit haben Sie noch nicht den Zugang

zum Inhalt des Datenfelds – den *Datenfeldelementen*. Allgemein kennzeichnet ein Datenfeldelement ein Bezeichner und in eckigen Klammern eine eindeutige Kennung über einen Index.

123 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++;
12 }
13 </script>
14 </html>
```

Listing 299: Wertzuweisung von Datenfeldelementen mit anschließendem Auslesen

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 Datenfeldelementinhalt, und der Inhalt des jeweiligen Elements wird mit `document.write()` in der Webseite ausgegeben. Als Abbruchkriterium der Schleife wird die Größe des Datenfelds verwendet (`i < test.length`). Zeile 11 erhöht die Zählvariable.

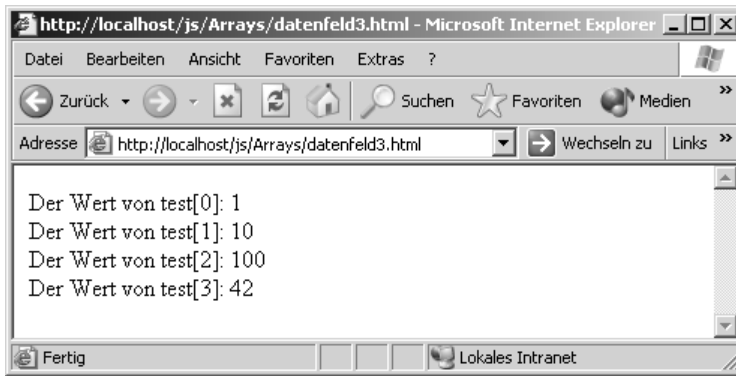


Abbildung 213: Lese- und Schreibzugriff auf Datenfeldelemente

Ein Datenfeld kann in JavaScript auch gleichzeitig verschiedene Datentypen aufnehmen. Das können viele andere Programmiersprachen nicht⁴. 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();
06 adr[0] = "Hauptstrasse";
07 adr[1] = 42;
08 adr[2] = 12345;
09 adr[3] = "Hinterdemon";
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 300: Ein Adressdatenfeld mit verschiedenen Datentypen

In dem Beispiel wird in Zeile 4 ein leeres Datenfeld mit dem Namen `adr` angelegt. In den Zeilen 4 bis 11 werden sieben Werte dem Datenfeld hinzugefügt (mit unterschiedlichen Datentypen – Ganzzahlen und Texte), so dass danach das Datenfeld aus sieben Elementen besteht.

4. Hier zeigt sich die lose Typisierung von JavaScript.

In Zeile 13 bis 16 wird auf die einzelnen Felder von dem Datenfeld zugegriffen (beachten Sie, dass das nicht in chronologischer Reihenfolge geschieht), um die Werte in Verbindung mit festen Strings auszugeben.

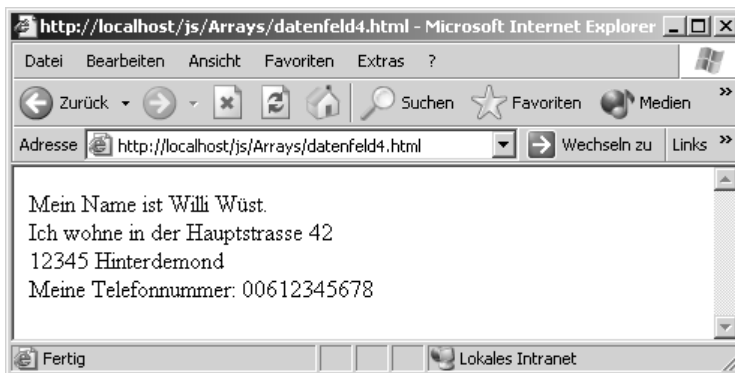


Abbildung 214: Es lassen sich in einem Datenfeld verschiedene Typen von Werten speichern.

124 Wie erfolgt der Zugriff über einen String-Index? Assoziative Datenfelder

So genannte *assoziativ* 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 eben kein numerischer Index angegeben. Sinn und Zweck eines assoziativen Datenfelds ist im Wesentlichen, einen Quelltext besser lesbar zu machen. Wenn Sie das letzte Beispiel mit den Adressinformationen in einem Datenfeld betrachten, wird Ihnen sicher klar, dass sich die einzelnen Bestandteile der Adressinformationen kaum aufgrund des Index schließen 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 " ,

```

Listing 301: Die Verwendung von assoziativen Datenfeldern

```

14 person["strasse"], " ", person["hausnr"],
15 "<br>", person["plz"], " ", person["ort"],
16 "<br>Meine Telefonnummer: ", person["telefon"]);
17 </script>
18 </html>

```

Listing 301: Die Verwendung von assoziativen Datenfeldern (Forts.)

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.

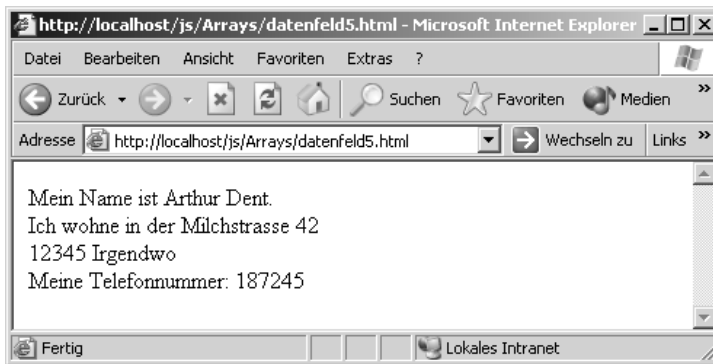


Abbildung 215: Die Verwendung von assoziativen Datenfeldern ist oft besser lesbar.

Achtung

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 302: Ein gemischtes Datenfeld

Sie sollten allerdings von solchen Konstruktionen dringend Abstand nehmen. Der Quelltext wird sehr unübersichtlich, und es gibt kaum einen sinnvollen Grund, für diesen 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:

```
17 alert(person.length);
```

Listing 303: Ergänzung des Beispiels um die Ausgabe der Datenfeldgröße

Hinweis

Die Eigenschaft `length` hat bei einem rein assoziativen Datenfeld immer den Wert 0.

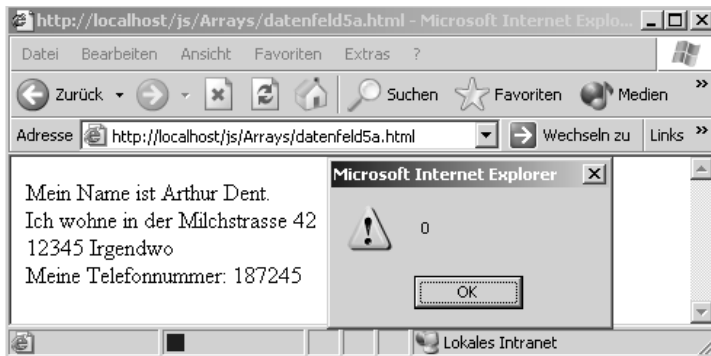


Abbildung 216: Das assoziative Datenfeld hat sieben Einträge, aber `length` liefert den Wert 0.

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 304: 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;  
07 person["ort"] = "Irgendwo";  
08 person["strasse"] = "Milchstrasse";
```

Listing 305: Der Versuch, auf ein assoziatives Datenfeld mit einem Index zuzugreifen

```

09 person["hausnr"] = 42;
10 person["telefon"] = 005555555;
11 for (i = 0; i<7 ;i++) {
12   document.write(person[i] + "<br>");
13 }
14 </script>
15 </html>

```

Listing 305: 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`.

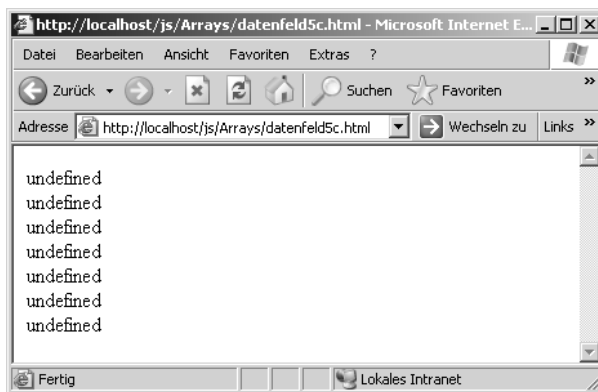


Abbildung 217: Das ist nicht der Inhalt des assoziativen Datenfelds.

Der Grund ist, dass ein assoziatives Datenfeld zwar ebenfalls chronologisch, aber eben nicht mit einem numerischen Index verwaltet wird. Das ist in den meisten Programmieretechniken 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.

125 Wie erfolgt der Zugriff auf alle Elemente eines Datenfelds über for...in?

Trotz des fehlenden numerischen Index 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 306: 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 verwenden. Das gilt sowohl für Datenfelder mit numerischem Index als auch 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 307: 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.

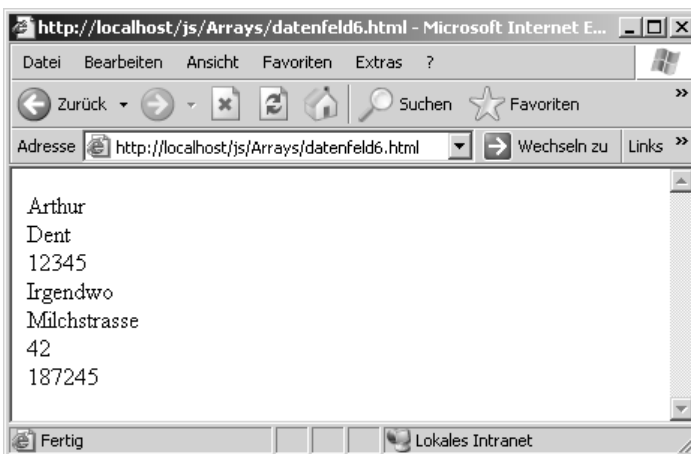


Abbildung 218: Auch bei assoziativen Datenfeldern geht die `for...in`-Schleife – Sie erkennen, dass auch das assoziative Array einen chronologischen Aufbau hat.

126 Wie kann ich die Datenfeldgröße bzw. 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 308: 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) die Eigenschaft `length`.



Abbildung 219: Die Anzahl der Elemente in dem Datenfeld wird mit `length` abgefragt.

Aber hat das Datenfeld jetzt wirklich 100 Elemente?

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. Da ist einmal die Technik der assoziativen Datenfelder. Für ein assoziatives Datenfeld steht die Größe des Datenfelds über die Eigenschaft `length` nicht zur Verfügung (siehe das Rezept »Zugriff über einen String-Index« auf Seite 472). Sie können aber die Größe indirekt bestimmen, indem Sie die `for...in`-Schleife verwenden. Etwa so:

```
01 function groesseArray(a) {
02   g=0;
```

Listing 309: Berechnung der Größe eines Datenfelds

```
03  for (i in a) {  
04      g++;  
05  }  
06  return g;  
07  }
```

Listing 309: Berechnung der Größe eines Datenfelds (Forts.)

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 310: 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 220: Die Bestimmung der Größe eines assoziativen Datenfelds

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:</td><td>"
16   + a.length + "</td></tr>");
17 document.write(
18 "<tr><td>Der Wert von groesseArray(a) nach dem Anlegen des Datenfelds:</td><td>"
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]:</td><td>"
26   + groesseArray(a) + "</td></tr>");
27 a[100] = 1;
28 document.write(
29 "<tr><td>Der Wert von a.length nach dem Anlegen von a[100]:</td><td>" + a.length +
30   "</td></tr>");
31 document.write(
32 "<tr><td>Der Wert von groesseArray(a) nach dem Anlegen von a[100]:</td><td>"

```

Listing 311: Der Unterschied zwischen `length` und der tatsächlichen Anzahl der enthaltenen Elemente

```

32 + groesseArray(a) + "</td>");
33 </script>
34 </tr>
35 </table>
36 </body>
37 </html>

```

Listing 311: Der Unterschied zwischen `length` und der tatsächlichen Anzahl der enthaltenen Elemente

In dem Beispiel wird ein Datenfeld mit numerischem Index angelegt. Zeile 13 definiert 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 sicher nicht verwunderlich, 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, abweichen. Während `a.length` den Wert 101 liefert, wird `groesseArray(a)` nur den Wert 2 liefern.

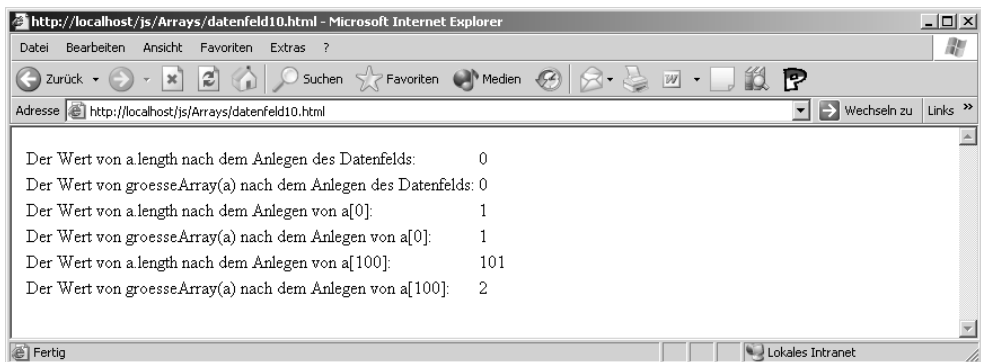


Abbildung 221: Die letzten beiden Werte weichen voneinander ab.

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 maximal bereits 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 Datenfелеlement erzeugt wird und vor der die Elemente noch nicht existieren, mit `length` eine Schleife über das Datenfeld durchlaufen wird (*datenfeld11.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 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><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 312: 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 Zeile 21 bis 23 durchläuft aber dessen ungeachtet auch diese Elemente.

Das Verhalten von `length` ist jetzt keine Schwäche, sondern Sie können das Verhalten natürlich 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.

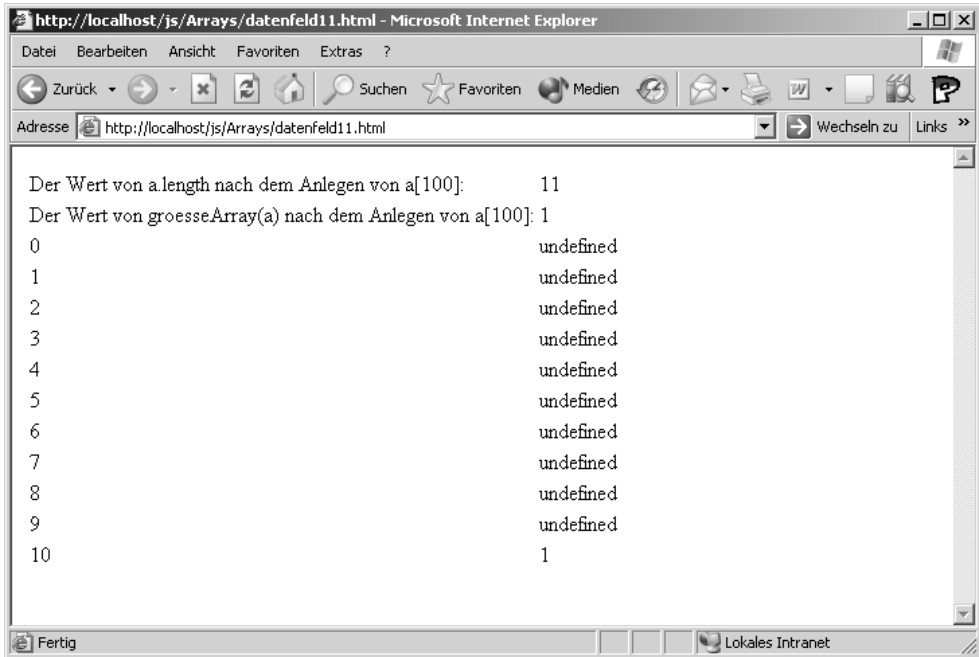


Abbildung 222: Die ersten zehn Elemente des Datenfelds sind nicht definiert.

Hinweis

Natürlich können Sie diese Funktion `groesseArray()` auch bei Datenfeldern mit numerischem Index ohne undefinierte Elemente verwenden. Dieser Umstand ist dort jedoch überflüssig.

127 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 313: 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 314: Verändern der Größe eines Datenfelds

Die anfängliche Größe des Datenfelds wird in Zeile 3 auf 100 festgelegt. Die Ausgabe in Zeile 4 bis 5 belegt 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 223: 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 477).

Dieses Verhalten von Datenfeldern in JavaScript und der eigentlichen 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(

```

Listing 315: Festlegung der Größe des Arrays mit `length`

```

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: "
09   + a.length + "<br>");
10 </script>
11 </html>

```

Listing 315: 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` direkt ein Wert zugewiesen, was auch ohne Fehler funktioniert. Die zweite Ausgabe zeigt es.

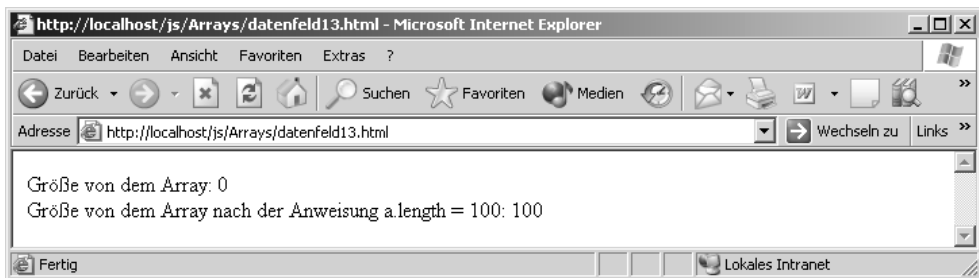


Abbildung 224: Der Eigenschaft `length` wurde explizit mit Wertzuweisung der Wert 100 zugewiesen.

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.

128 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. 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 316: Anlegen eines multidimensionalen Datenfelds

Dabei verwenden Sie als Elemente des äußeren Datenfelds den Konstruktoraufruf 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]`⁵ einfach weitere 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 317: 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.

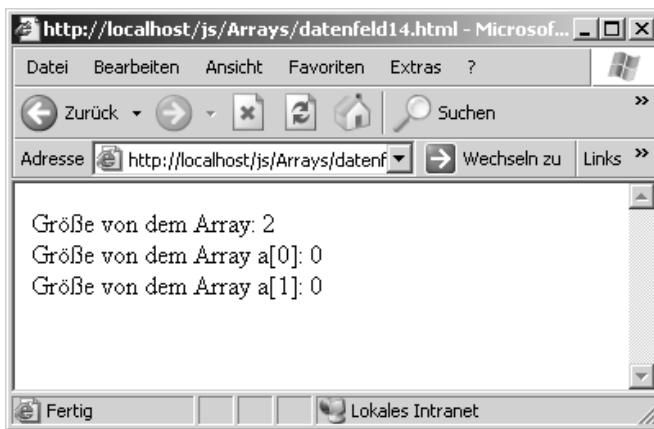


Abbildung 225: Die Größenangaben von einem zweidimensionalen Datenfeld

Natürlich können Sie auf diese Weise auch tiefer verschachtelte Datenfelder erzeugen.

5. Der Wert `n` steht für die Anzahl der angelegten Datenfelder.

Beispiel (*datenfeld15.html*):

```

01 <html>
02 <script language="JavaScript">
03 a = new Array(new Array(new Array(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 document.write("Größe von dem Array a[0][0]: " + a[0][0].length + "<br>");
08 document.write("Größe von dem Array a[0][0][0]: " + a[0][0][0].length + "<br>");
09 document.write("Größe von dem Array a[1][0]: " + a[1][0].length + "<br>");
10 </script>
11 </html>

```

Listing 318: Ein multidimensionales Datenfeld mit unterschiedlichen Dimensionen

In Zeile 2 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 soll⁶. In der Praxis machen solche Gebilde jenseits der zweiten Dimension kaum Sinn, aber 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.

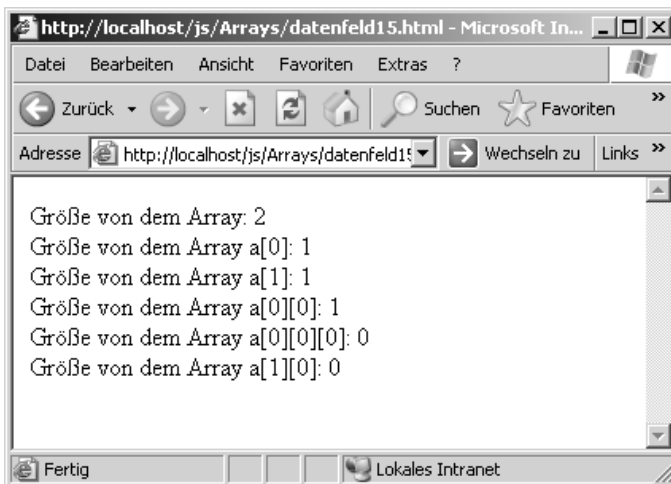


Abbildung 226: Ein krummes Array mit vier beziehungsweise drei Dimensionen

6. ;-)

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";
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] = "Hinterdemon";
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], "<br>");
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], "<br>");
27 //-->
28 </script>
29 <body>
30 </body>
31 </html>

```

Listing 319: Eine Adressliste als zweidimensionales Array

In Zeile 4 wird mit `var adr = new Array(new Array(), new Array());` ein zweidimensionales 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.

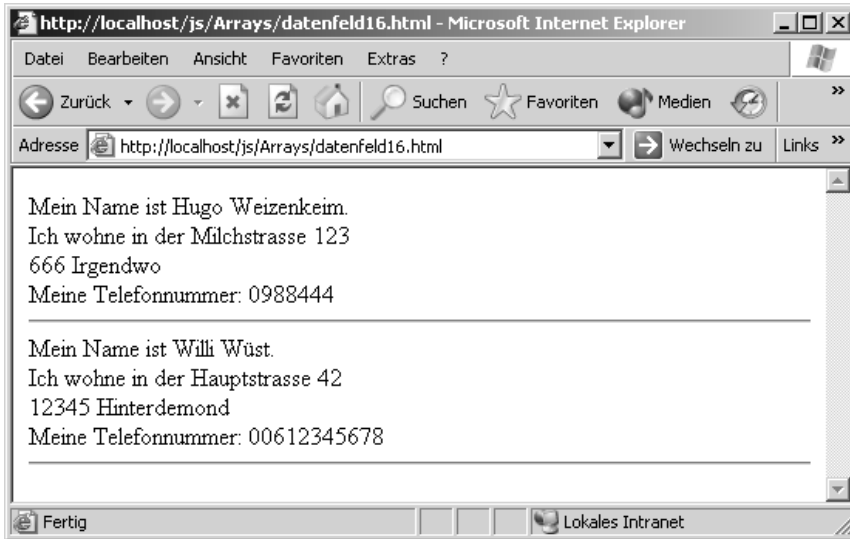


Abbildung 227: Eine kleine Adressdatenbank als zweidimensionales Datenfeld

Wie schon im Fall 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"],

```

Listing 320: Der Einsatz von einem assoziativen Index in der zweiten Dimension

```

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 320: Der Einsatz von einem assoziativen Index in der zweiten Dimension (Forts.)

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"] = "Hinterdemon";
18 adr["willi"]["telefon"] = "00612345678";
19 adr["willi"]["nname"] = "Wüst";
20 adr["willi"]["vname"] = "Willi";
21 document.write(
22 "Mein Name ist " ,adr["hugo"]["vname"] , " " ,adr["hugo"]["nname"] ,
23 ".<br>Ich wohne in der ",adr["hugo"]["str"], " " ,adr["hugo"]["hausnr"],
24 "<br>",adr["hugo"]["plz"]," " ,adr["hugo"]["ort"],

```

Listing 321: Ein rein assoziatives Datenfeld mit zwei Dimensionen

```

25 "<br>Meine Telefonnummer: ",adr["hugo"]["telefon"],"<hr>");
26 document.write(
27 "Mein Name ist " ,adr["willi"]["vname"] , " " ,adr["willi"]["nname"] ,
28 ".<br>Ich wohne in der ",adr["willi"]["str"], " " ,adr["willi"]["hausnr"],
29 "<br>",adr["willi"]["plz"]," " ,adr["willi"]["ort"],
30 "<br>Meine Telefonnummer: ",adr["willi"]["telefon"],"<hr>");
31 //-->
32 </script>
33 <body>
34 </body>
35 </html>

```

Listing 321: Ein rein assoziatives Datenfeld mit zwei Dimensionen (Forts.)

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, einen String-Index dort 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.

129 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.

Wenden wir zuerst einmal die Methode ganz naiv 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>");

```

Listing 322: Anwendung von `sort()` auf ein Datenfeld

```
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 322: Anwendung von sort() auf ein Datenfeld (Forts.)

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`-Objekts⁷ 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 ganzzahlig. Mit `Math.round()` schneiden wir den Nachkommateil ab und erhalten damit die gewünschten ganzen Zahlen.

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] + "<br>");}`). Wir werden die Funktion vor der Sortierung und nach selbiger 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 einsortiert zu werden. Aber das gilt scheinbar nicht für alle Zahlen.

7. Ein Built-In-Objekt von JavaScript

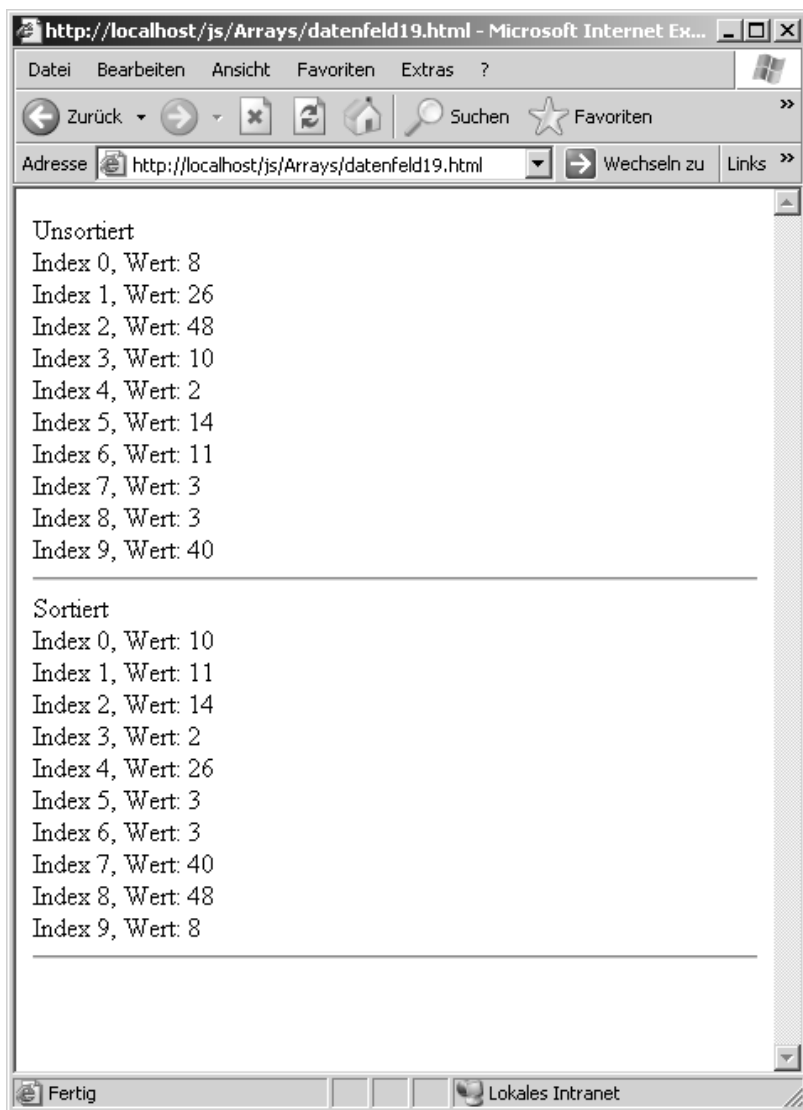


Abbildung 228: Seit wann ist 8 größer als 48?

Aber wenn Sie genauer hinschauen, erkennen Sie wahrscheinlich das Prinzip. Die Sortierung erfolgt nach dem ersten Zeichen in einem Datenfeldelement von klein nach groß. 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 879999999.

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 503).

Hinweis

Beachten Sie, dass sich die Sortierung von numerischen Datenfeldinhalten von der Sortierung bei 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 den Wert von dem jeweiligen ASCII-Code, wobei zuerst das erste Zeichen, dann das zweite Zeichen usw. eine Rolle spielen. Damit kann man nicht nur Zahlen sortieren, sondern der Sortieralgorithmus funktioniert sowohl bei Zahlen als auch Buchstaben und Sonderzeichen⁸. 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 323: 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`. Wenn Sie nun Zahlen nach ihrem tatsächlichen numerischen Wert sortieren wollen, hilft ein kleiner Trick. Beachten Sie das nachfolgende Listing (*datenfeld19b.html*):

8. Bei ganzen Texten gibt es einige Feinheiten zu beachten, die in Kürze folgen.

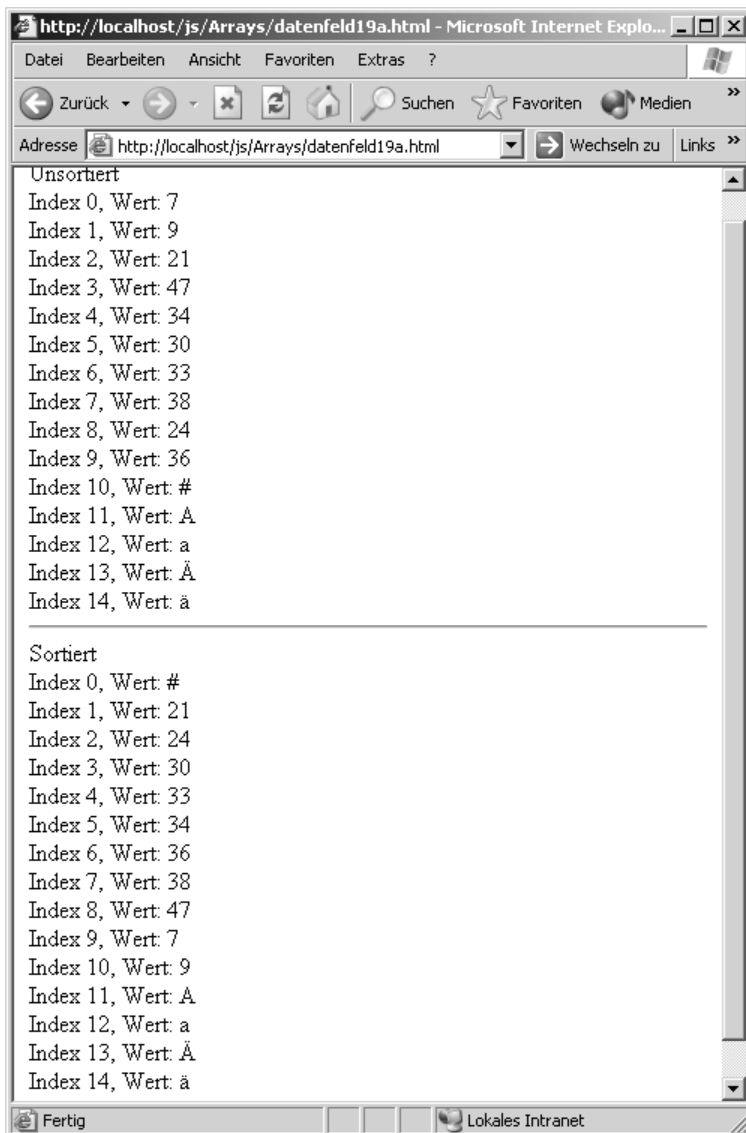


Abbildung 229: Buchstaben folgen nach Zahlen.

```

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;

```

Listing 324: Sortierung nach numerischem Wert

```
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 324: Sortierung nach numerischem Wert (Forts.)

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 von dem Multiplikator. 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 geben.

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` von diesem String abfragen, und das ist die Anzahl der Stellen (`l = (" " + 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 identisch zu der Länge des Multiplikators 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;`). Dass wir hier mit dem Stringformat arbeiten, ist wichtig. Andernfalls würden führende Nullen von JavaScript durch interne Optimierung wieder beseitigt, und wir hätten nichts gewonnen. In Zeile 8

berechnen wir danach erneut die Länge von der angepassten Zufallszahl⁹ (`t = (" " + temp).length;`). Wenn die angepasste Zufallszahl die gleiche Länge wie der Multiplikator hat, wird der Wert als String (!) zurückgegeben (Zeile 10 – `return temp;`).

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 wurden. 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 Stringformat stellt sicher, dass interne Optimierungen führende Nullen nicht wegoptimieren und unsere ganzen Anstrengungen verpuffen lassen.

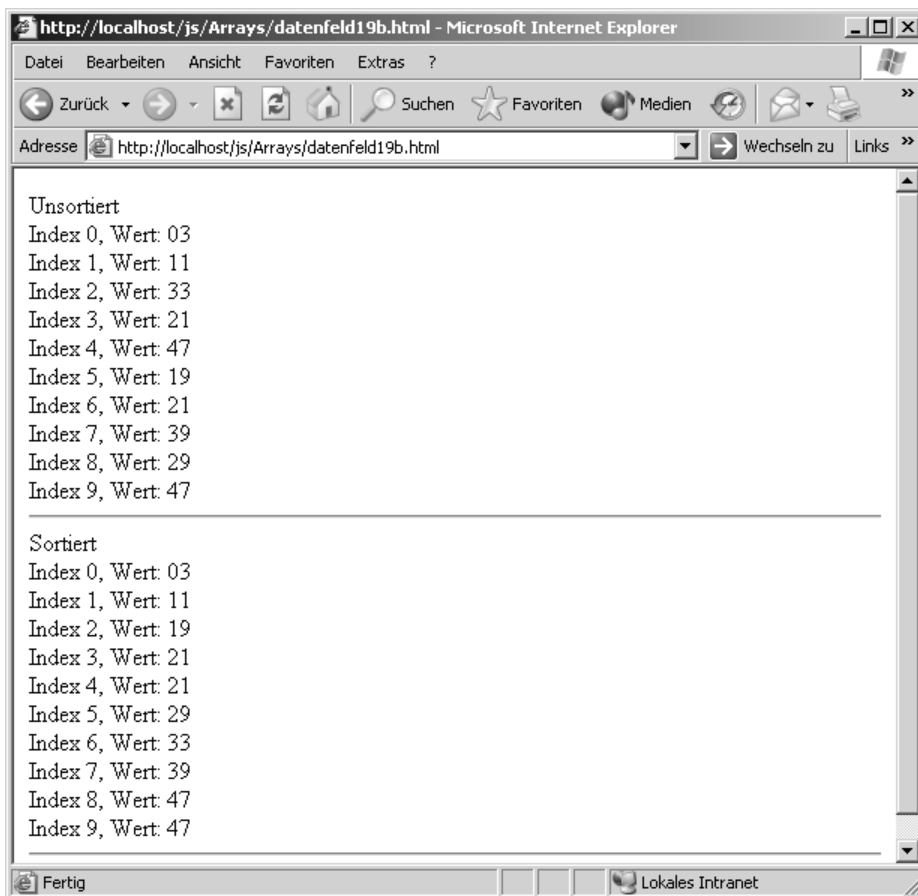


Abbildung 230: Die Sortierung erfolgt nach der numerischen Größe.

Jetzt kann es natürlich sein, dass Ihnen in der Darstellung die führenden Nullen nicht gefallen. Aber eine kleine Anpassung lässt diese verschwinden.

9. Genauer – dem String mit dem numerischen Wert der Zufallszahl als Inhalt.

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 325: 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 Datenfелеlement vor der Ausgabe (!) in eine Zahl. Damit optimiert JavaScript intern alle führenden Nullen weg. Was uns bei der Sortierung ein Problem bereitet hätte, nutzen wir hier gezielt aus.

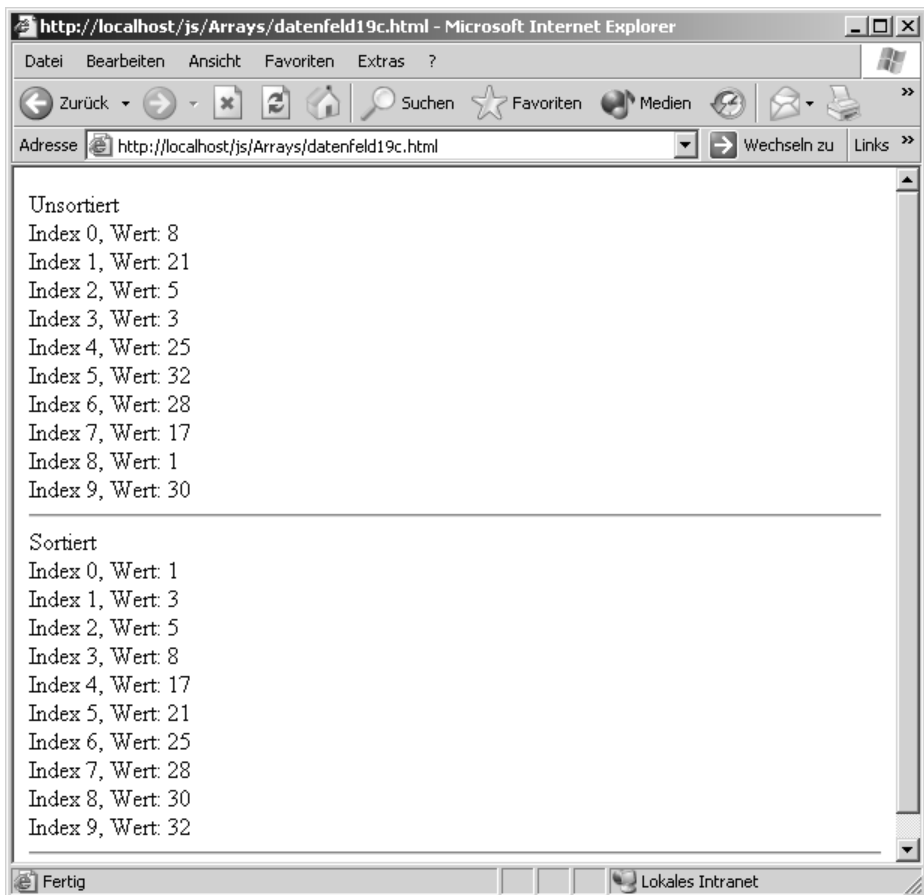


Abbildung 231: Einstellige Zahlen werden vor zweistelligen Zahlen ausgegeben.

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 Hand voll Feinheiten und vor allem Unterschiede zu dem Verfahren bei numerischen Inhalten beachtet werden. Zuerst wenden wir einfach die `sort()`-Methode ohne besondere Vorkehrungen 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 326: Einfache Sortierung von Texten

Außer dem Aufbau des Datenfelds (ein Textdatenfeld mit unterschiedlichen langen Texte 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. Das war bei 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 entsprechend wird es dahinter einsortiert.

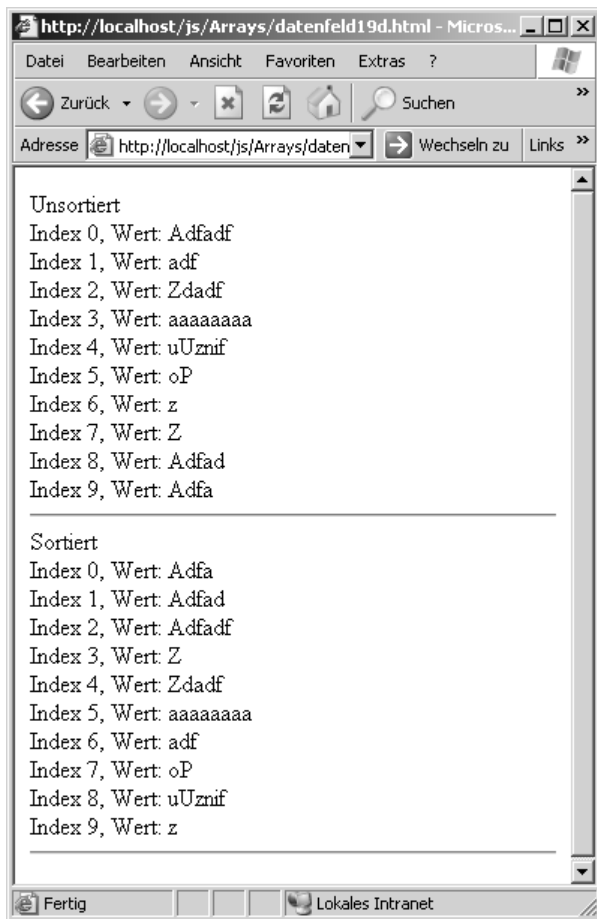


Abbildung 232: Die natürliche Sortierung von Texten unterschiedlicher Länge mit Berücksichtigung von Groß- und Kleinschreibung

Hinweis

Das Sortierverfahren kommt auch in der Praxis bei zahlreichen Sortierroutinen zur Verwendung (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 vorkommt (von kurz nach lang), können Sie folgendes Rezept anwenden, das alle Text auf die gleiche Länge konvertiert (*datenfeld19e.html*):

```
01 <html>
02 <script language="JavaScript">
03 laenge = 0;
```

Listing 327: Auffüllen der Texte auf gleiche Länge

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

HTML

Events

```

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) {
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 42 </html>

```

Listing 327: 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 Datenfeldinhalts »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 werden¹⁰, 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;`).

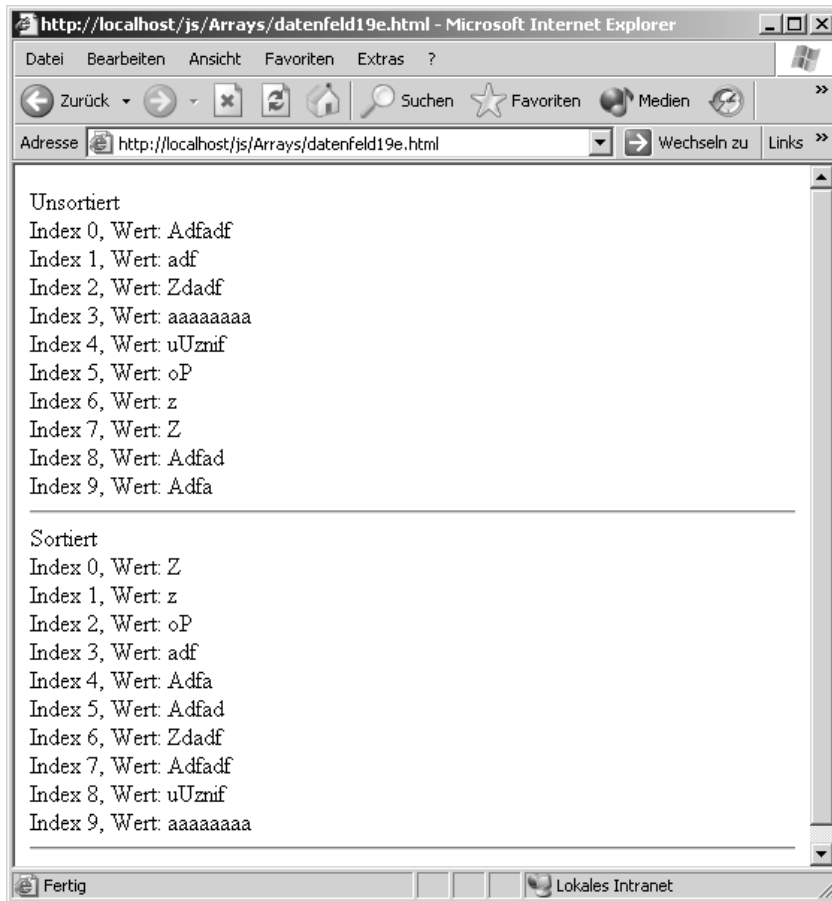


Abbildung 233: Sortierung von kurz nach lang

Wenn Sie sich die Ausgabe ansehen, werden die führenden Leerzeichen fehlen. Hat JavaScript diese bei Strings wie führenden 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 wegzaubert. Der Browser lässt Leerzeichen bei der Darstellung weg, wenn sie mehrfach vorkommen. Es handelt sich also um einen reinen Darstellungseffekt des Browsers.

10. Es ist keine Konvertierung mit einem vorangestellten Leerstring notwendig.

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*):

```
document.write(" <pre>Index " + i + ", Wert:" + a[i] + "</pre>");
```

Listing 328: Verhinderung 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 234: Die Leerzeichen sind schon da.

Hinweis

Natürlich kann die Sortierung von Texten auch unabhängig von Groß- und Kleinschreibung erfolgen. Das ist trivial, 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 Originaltexts 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.

130 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 490*).

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 329: 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 ganzzahligen 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. 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] + "<br>");}`).

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.

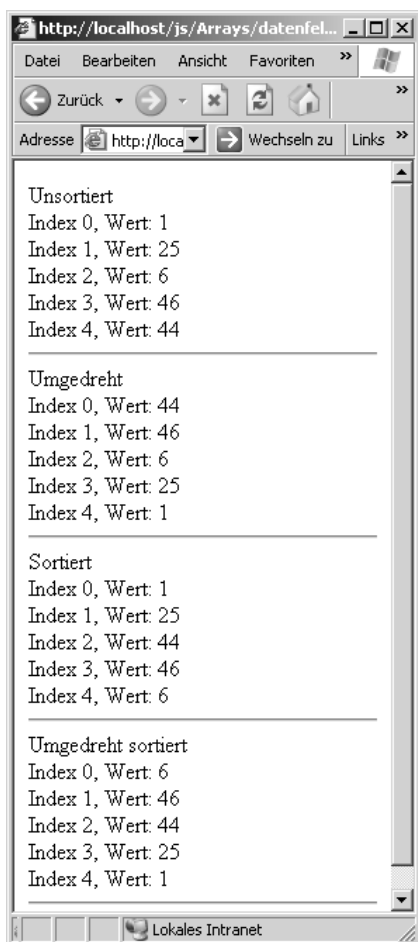


Abbildung 235: Unsortiert, umgedreht, sortiert und wieder umgedreht

Beachten Sie die Erklärungen zu dem Sortieralgorithmus im Rezept »Wie kann ich ein Datenfeld sortieren?« auf Seite 490.

131 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 zusammengeführten Datenfeldelementinhalte in dem resultierenden String abgrenzt und bei Bedarf dazu verwendet werden kann, mit `split()` den String 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 330: Verbinden eines Datenfelds zu einem String

In der Zeile 3 wird ein Datenfeld ohne Elemente erzeugt und durch die Schleife in Zeile 5 bis 8 mit den Rückgaben der `prompt()`-Methode aufgebaut.



Abbildung 236: Der Anwender gibt Werte zur Erzeugung des Datenfelds ein.

In Zeile 9 erfolgt die Ausgabe der Größe des Datenfelds (`document.write("Größe von dem Array: " + a.length + "<br>");`), und in Zeile 10 wird das Datenfeld zu einem String verbunden und dieser ausgegeben. Als Trennzeichen fungiert das Zeichen `%` (`document.write(a.join("%"));`).

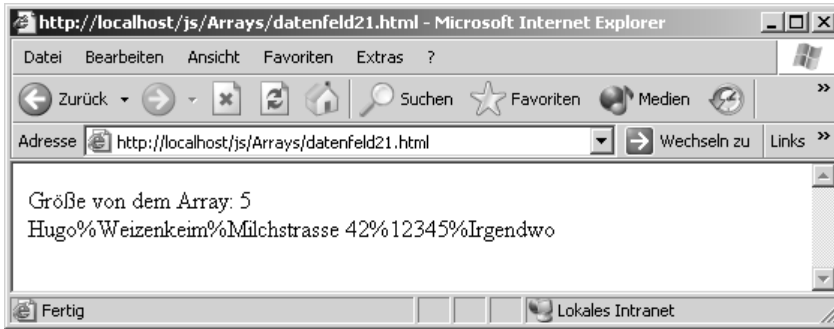


Abbildung 237: Die Ausgabe der Datenfeldgröße und des aus dem Datenfeld generierten Strings

132 Wie kann ich ein Datenfeldelement aus dem 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 zwar das letzte Element in einem Datenfeld entfernen und mit `shift()` das erste Element. Ebenso funktioniert es zum 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 angesagt, 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 nachfolgenden 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 nachfolgenden 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++){

```

Listing 331: Löschen eines Elements aus einem Datenfeld mit Verkleinerung von `length`

```
10  t2.shift();
11  }
12  return t1.concat(t2);
13 }
```

Listing 331: Löschen eines Elements aus einem Datenfeld mit Verkleinerung von length (Forts.)

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 332: Löschen eines Elements aus einem Datenfeld ohne Verkleinerung von length

Der erste Übergabewert der Funktion ist auch das Originaldatenfeld und der zweite Parameter der Index des Felds, das beseitigt werden soll. In Zeile 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 `f2` 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 15 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, etwas umfangreicheren Beispiel durch (*datenfeld22.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(
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();
```

Listing 333: Verschiedene Techniken zum Entfernen von Datenfeldelementen


```
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++){
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 " + a.length);
73 a.pop();
74 document.write(
75   "Größe von dem Array nach einmal pop(): "
76   + a.length + "<br>");
77 document.write(
78   "Anzahl der tatsächlich existierenden Elemente: "
79   + groesseArray(a) + "<br>");
80 ausgabe(a, "Inhalt nach einmal pop()");
81 a.shift();
82 document.write(
83   "Größe von dem Array nach einmal shift(): "
84   + a.length + "<br>");
85 document.write(
86   "Anzahl der tatsächlich existierenden Elemente: "
87   + groesseArray(a) + "<br>");
88 ausgabe(a, "Inhalt nach einmal shift()");
89 a = loescheAusMittel(a,2);
90 document.write(
91   "Größe von dem Array nach Aufruf von loescheAusMittel(a,2): "
92   + a.length + "<br>");
```

Listing 333: Verschiedene Techniken zum Entfernen von Datenfeldelementen (Forts.)

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

```

93 document.write(
94   "Anzahl der tatsächlich existierenden Elemente: "
95   + groesseArray(a) + "<br>");
96 ausgabe(a, "Inhalt nach Aufruf von loescheAusMitte1(a,2)");
97 a = loescheAusMitte2(a,2);
98 document.write(
99   "Größe von dem Array nach Aufruf von loescheAusMitte2(a,2): "
100   + a.length + "<br>");
101 document.write(
102   "Anzahl der tatsächlich existierenden Elemente: "
103   + groesseArray(a) + "<br>");
104 ausgabe(a, "Inhalt nach Aufruf von loescheAusMitte2(a,2)");
105 </script>
106 </html>

```

Listing 333: Verschiedene Techniken zum Entfernen von Datenfeldelementen (Forts.)

Das Beispiel benötigt neben den Funktionen `loescheAusMitte1()` (Zeile 18 bis 30) und `loescheAusMitte2()` (31 Zeile bis 45) zwei Hilfsfunktionen, die so oder ähnlich auch in anderen Rezepten in diesem Kapitel Verwendung finden:

- ▶ Die Funktion `groesseArray()` in Zeile 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 ausgegeben, das als erster Parameter übergeben wird. 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 334: Die Ausgabe nach der reinen Erzeugung

In der `for`-Schleife von Zeile 55 bis 57 wird das Datenfeld mit Zahlen gefüllt. Anschließend werden wieder 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 335: 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 336: 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 337: 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 geschoben 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 338: 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 339: 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 340: 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 angeben¹¹.

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 }
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 + "<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 341: Anwendung von splice() zum Eliminieren von Datenfeldelementen

In Zeile 26 eliminieren wir mit `a.splice(2,3)`; ab dem Index 2 drei Elemente aus dem Datenfeld.

11. Geben Sie jedoch weitere Parameter an, werden die bisherigen Inhalte des Datenfelds ab dem Start-index durch diese Inhalte ersetzt.

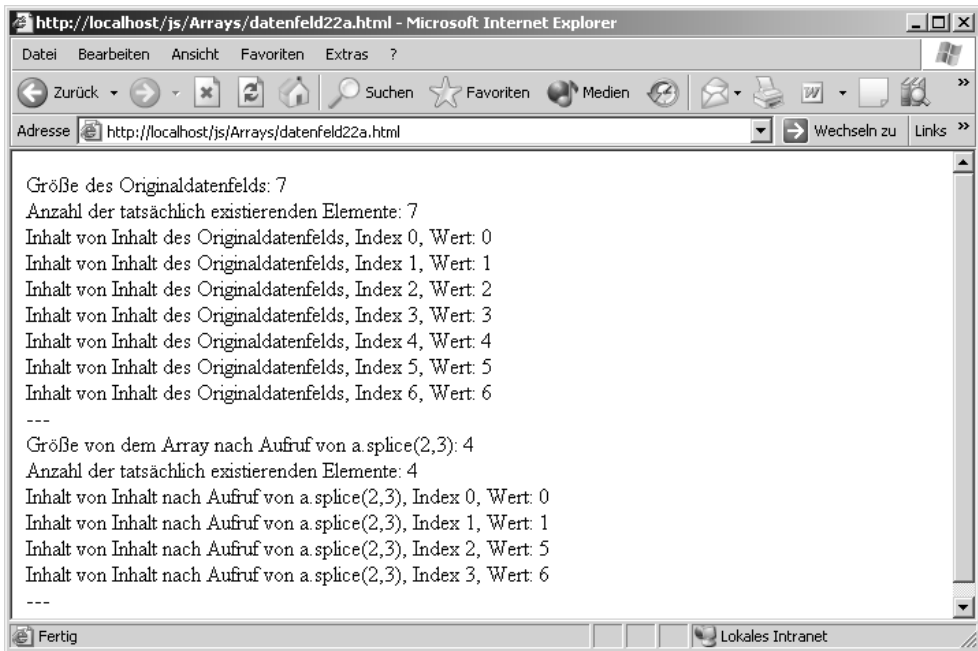


Abbildung 238: Löschen von Datenfeldelementen mit *splice()*

133 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 von dem zu vergrößernden Datenfeld 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 }

```

Listing 342: Manuelles Anfügen von Elementen an das Ende eines Datenfelds

```
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 342: Manuelles Anfügen von Elementen an das Ende eines Datenfelds (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 ausgegeben, das als Parameter übergeben wird. 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 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 drei neue Elemente dem Datenfeld 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.

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 343: Quellcodepassage aus `datenfeld23.html`

durch die folgende Codezeile auszutauschen (*`datenfeld23a.html`*):

```
17 a.push(5, 6, 7);
```

Listing 344: Äquivalenter Effekt mit der `push()`-Methode

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

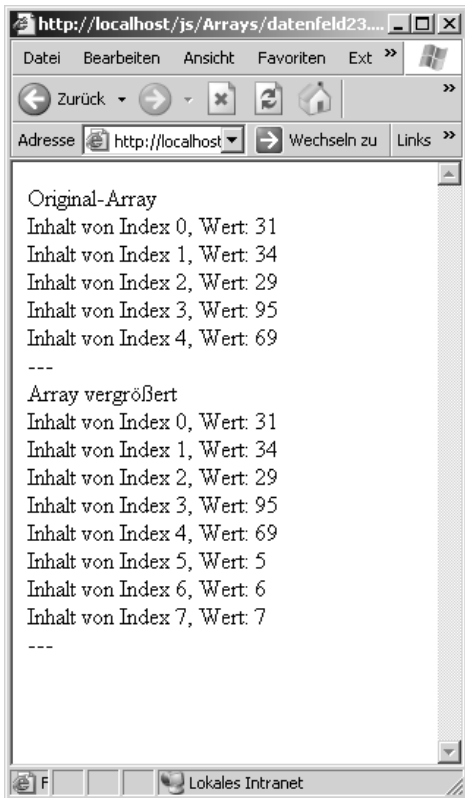


Abbildung 239: Das Originaldatenfeld und das vergrößerte Datenfeld

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.

134 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">
```

Listing 345: Manuelles Extrahieren eines Teils von einem Datenfeld


```
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 345: Manuelles Extrahieren eines Teils von einem Datenfeld (Forts.)

In der Funktion `ausgabe()` (Zeile 3 bis 10) werden mit einer `for`-Schleife die Inhalte aller Elemente in dem Datenfeld ausgegeben, das als Parameter übergeben wird. 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 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 Faktor 2 beim Index vom Originaldatenfeld `a` ist der Startwert der Extrahierung (Zeile 16 – `t[i] = a[i + 2];`). Der Endindex ergibt sich aus der Zahl der extrahierten Elemente.

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 von dem ersten zu extrahierenden Element und als zweiten Parameter den Index des letzten Elements.

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

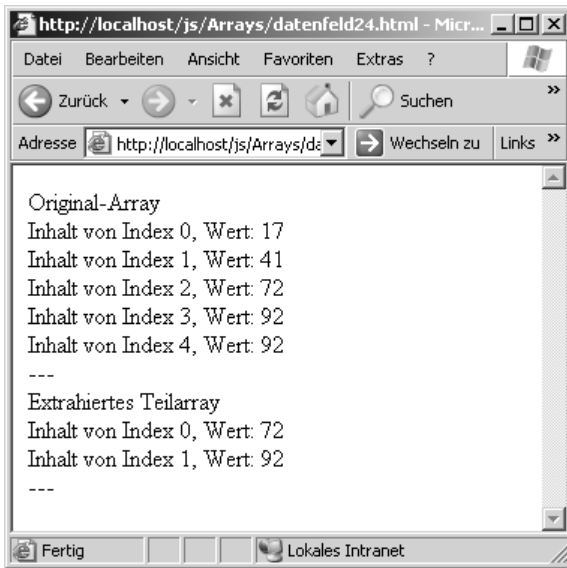


Abbildung 240: Extrahieren eines Teils von einem Datenfeld

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 346: Quellcodepassage aus datenfeld24.html

durch die folgende Codezeile auszutauschen (*datenfeld24a.html*):

```

17 t = a.slice(2,4);

```

Listing 347: 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 Datenfeld-elements rückwärts gezählt. Tauschen Sie zur Verdeutlichung die `slice()`-Methode im letzten Beispiel wie folgt aus (*datenfeld24b.html*):

```
17 t = a.slice(2,-1);
```

Listing 348: Äquivalenter Effekt mit der slice()-Methode und negativem Index

Das bedeutet bei einem Datenfeld mit fünf Elementen, dass vom Index 2 bis zum Index (5 - 1) extrahiert werden soll. Das ist also bei einem Datenfeld mit fünf 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.

135 Wie kann ich zwei Datenfelder zusammenfügen?

Sie können natürlich von Hand an das Ende eines Datenfelds die Elemente eines anderen Datenfelds anfügen, indem Sie die Größe von dem ersten Datenfeld mit der `length`-Eigenschaft ermitteln und explizit Wert für Wert des zweiten Datenfelds zur Erzeugung weiterer Datenfeldelemente ab dem nächsten Index 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);
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);
```

Listing 349: Manuelles Verbinden von zwei Datenfeldern

Grund-
lagen

Core

Formu-
lare

Aus-
nahme

Arrays

Strings

Datum

Platt-
form

DHTML

Events

```
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 349: 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 ausgegeben, das als Parameter übergeben wird. 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 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 von dem Datenfeld `a`.

In Zeile 18 erzeugen wir ein zweites Datenfeld `b` und füllen es analog 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 von dem zweiten Datenfeld 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 von dem zweiten Datenfeld `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 bereits für diese Tätigkeit die Methode `concat()` mit. Diese erwartet als Parameter den Namen von dem Datenfeld, 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 350: Quellcodepassage aus `datenfeld25.html`

durch die folgende Codezeile auszutauschen (`datenfeld25a.html`):

```
24 a = a.concat(b);
```

Listing 351: Äquivalenter Effekt mit der `concat()`-Methode

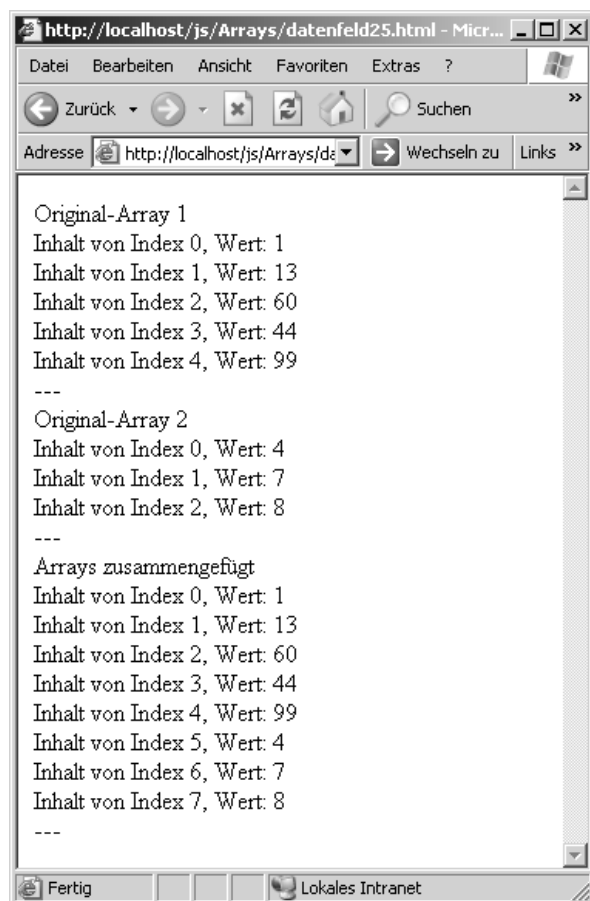


Abbildung 241: Zwei Datenfelder wurden manuell zusammengefügt.

Hinweis

Trotz der bequemen Vorgehensweise mit `concat()` ist die Beschäftigung mit der manuellen Vorgehensweise nützlich für das grundlegende Verständnis im Umgang mit Datenfeldern.

136 Wie kann ich am Anfang eines Datenfelds Elemente einfügen?

Sie können natürlich von Hand am Anfang eines Datenfelds Elemente 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 352: 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 ausgegeben, das als Parameter übergeben wird. 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 wird das Datenfeld als Parameter übergeben. Die Ausgabe ist die Größe und der Inhalt des ersten Datenfelds, der im folgenden Schritt am Anfang 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 von dem Datenfeld `b`. In Zeile 23 wird auch dieses Datenfeld ausgegeben.

Mit der folgenden `for`-Schleife werden die einzelnen Elemente von dem zweiten Datenfeld 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.

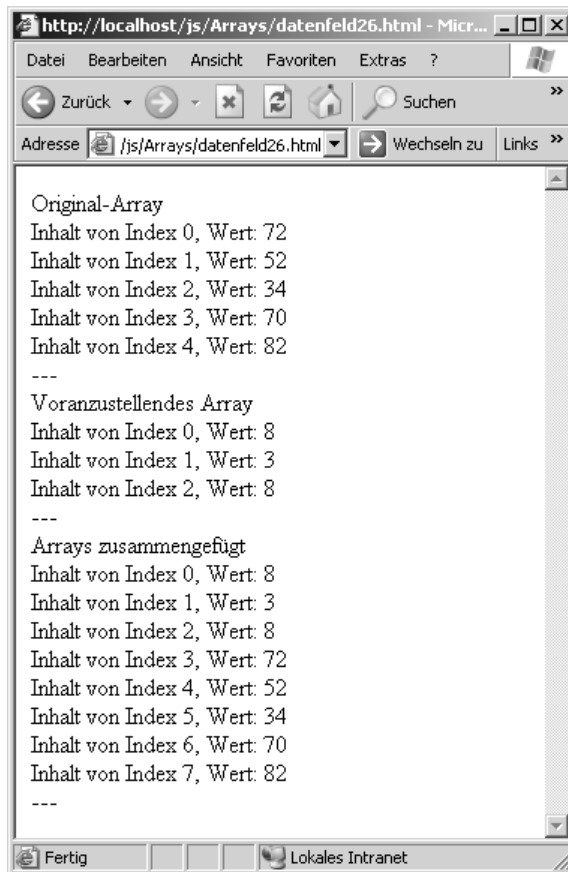


Abbildung 242: Voranstellen von Datenfeldelementen

Dieses manuelle Voranstellen eines zweiten Datenfelds funktioniert zuverlässig, aber noch eleganter ist die Verwendung der Methode `concat()`. Diese erwartet als Parameter den Namen von dem Datenfeld, 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 353: Quellcodepassage aus datenfeld26.html

durch die folgende Codezeile auszutauschen (*datenfeld26a.html*):

```
24 a = b.concat(a);
```

Listing 354: 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 hinzufügenden Elemente als Parameter.

In Beispiel *datenfeld26.html* brauchen Sie für die gleiche Wirkung wieder die Zeilen

```
24 for(i = 0; i < a.length; i++){
25   b[g + i] = a[i];
26 }
```

Listing 355: Quellcodepassage aus datenfeld26.html

durch die folgende Codezeile auszutauschen (*datenfeld26b.html*):

```
24 a.unshift(b[0], b[1], b[2]);
```

Listing 356: 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 von dem neuen Datenfeld.

Hinweis

Trotz der bequemen Vorgehensweise mit `unshift()` ist die Beschäftigung mit der manuellen Vorgehensweise nützlich für das grundlegende Verständnis im Umgang mit Datenfeldern.

137 Wie kann ich an einer beliebigen Stelle eines Datenfelds Elemente einfügen?

Sie können natürlich an einer beliebigen Stelle eines Datenfelds Elemente einfügen, indem Sie manuell die Inhalte dort ersetzen. Das erfolgt einfach über Zuweisung eines neuen Werts. Die zweite Alternative ist, die neuen Elemente einzufügen und die bisherigen Inhalte dort 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;
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 357: Manuelles Einfügen von Elementen in einem Datenfeld ohne und mit Verschieben der Originalinhalte

In der Funktion `ausgabe()`, die von Zeile 3 bis 10 definiert ist, werden mit einer `for`-Schleife die Inhalte aller Elemente in dem Datenfeld ausgegeben, das als Parameter übergeben wird. In Zeile 11 wird ein erstes Datenfeld `a` der Größe 5 angelegt, das direkt mit Vorgabewerten

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

initialisiert wird. In Zeile 13 erfolgt der erste Aufruf der Funktion `ausgabe()`. Dieser 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 sehen Sie das Resultat.

Tip

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 von Datenfeld `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 von dem Datenfeld `a` zugewiesen. Beachten Sie, dass der Index von einem Element aus `a` um den Wert 1 gegenüber dem Index von einem Datenfeldelement aus `b` reduziert ist. Das bewirkt die gewünschte Verschiebung. Das Feld `a[3]` wird also `b[4]` zugewiesen, `a[4]` `b[5]` 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` das Datenfeld `a` wieder, und wie gewünscht wurde an dem Index `neu` ein neues Element eingefügt.

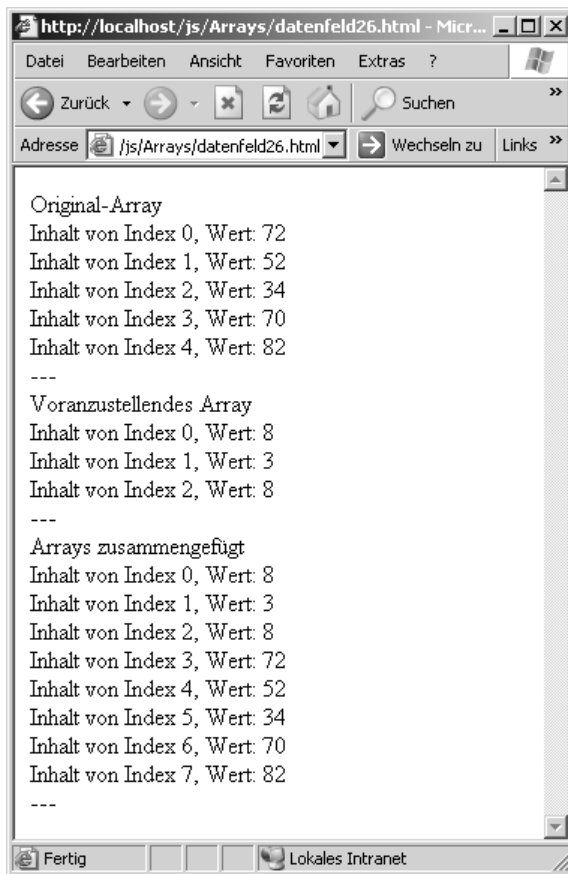


Abbildung 243: Einfügen mit und ohne Überschreiben

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.

138 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 358: 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 359: 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 bereits der Wert in einem vorherigen Element 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 360: 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 maximal in dem Datenfeld vorkommenden Wert 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 von dem Element mit dem maximal in dem Datenfeld vorkommenden Wert 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;
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 361: Suche nach dem Minimum in einem Datenfeld

Der wesentliche Unterschied zu der Suche nach dem Maximum ist der Vergleich in Zeile 5 auf kleiner (`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   }

```

Listing 362: Die Suche nach dem Maximum und dem Minimum in einem Datenfeld

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

HTML

Events

```
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.<br>");
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(
35     "Der Minimumwert des Arrays ist " + minimumwert
36     + " und ist an dem Index " + minimumindex
37     + " zu finden.<br>");
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 362: 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 ausgegeben, das als Parameter übergeben wird. In den folgenden Funktionen `sucheMax()` und `sucheMin()` ist nur die Reaktion auf die Ermittlung von dem Maximum und dem Minimum 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 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.

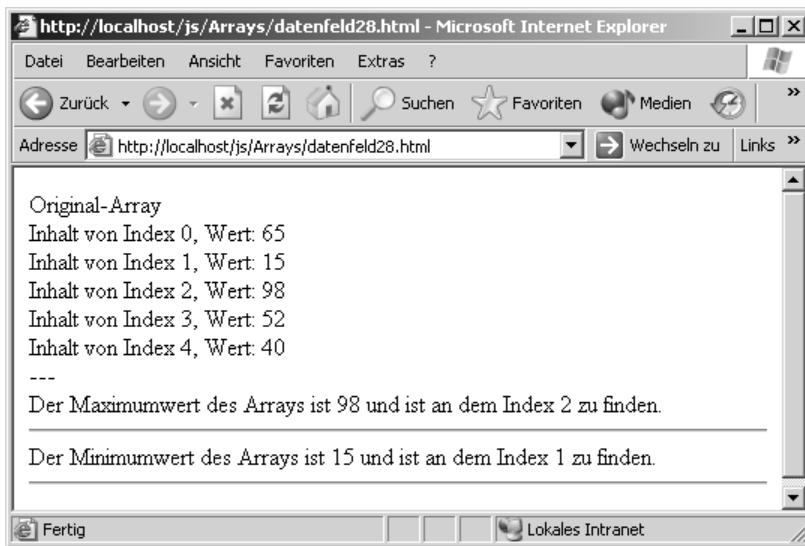


Abbildung 244: Ermittlung des Maximums und des Minimums samt jeweiligem Index

139 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 Lottospiel. 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 eine Zufallszahl zwischen 0 und 1 zu erzeugen. Aber das ist natürlich eine Zahl, die deterministisch an den Randbedingungen des Computerezustands 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++){
  
```

Listing 363: Mischen der Werte in einem Datenfeld

```
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 363: Mischen der Werte in einem Datenfeld (Forts.)

Die Funktion bekommt das zu mischende Datenfeld als Parameter `a` übergeben. In Zeile 2 erzeugen wir ein weiteres Datenfeld `b` mit genau der gleichen Größe wie die Größe 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`¹² 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];`).

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 ausgeführt¹³. 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.

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.

12. Es ist im Grunde unnötig, zwischen dem Datenfeld `a` und `b` zu unterscheiden, denn beide sind gleich groß.

13. Auf ein Neues :-).

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 (einzeln) hinein wollen. 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 statisch wird die Zeit zur Abarbeitung der Funktion nicht vorhersehbar sein, und je weiter die Durchläufe voranschreiten, desto länger dauert es bis zum Erfolg¹. 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.

1. Rein hypothetisch könnte bei unserer Programmierung sogar eine Endlosschleife entstehen. Das wäre der Fall, dass ein potenzieller Schwimmer beispielsweise endlos an vier verschlossenen Türen rüttelt und die fünfte freie Kabine endlos 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 Zählvariablen durchlaufen wird, bis ein Erfolg eintritt. Ob nun zehn Mal versucht wird, ein freies Datenfeld zu finden, oder 1000 Mal oder gar 100000 Mal spielt für einen Anwender keine spürbare Rolle. Lassen Sie einfach das Beispiel `datenfeld29a.html` laufen, in dem der folgende 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.

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));
```

Listing 364: Mischen der Einträge in einem Datenfeld

```
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 364: Mischen der Einträge 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 ausgegeben, das als Parameter übergeben wird. In den folgenden Zeilen 11 bis 23 In Zeile 24 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 29 erfolgt der erste Aufruf der Funktion `ausgabe()`. Dieser 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 = mische(a);`). Danach wird das gemischte Datenfeld ausgegeben und erneut gemischt und wieder das Resultat ausgegeben.

140 Wie kann ich eine Lottoziehung realisieren?

Unsere Funktion `mische()` kann nun auch dazu verwendet werden, ein Lottospiel zu simulieren. Aber beachten Sie erst einmal ein Lottospiel, das einfach ein Datenfeld mit sieben¹⁴ zufällig ermittelten numerischen Einträgen zwischen 1 und 49 aufbaut.

14. Sechs Zahlen und die Zusatzzahl.

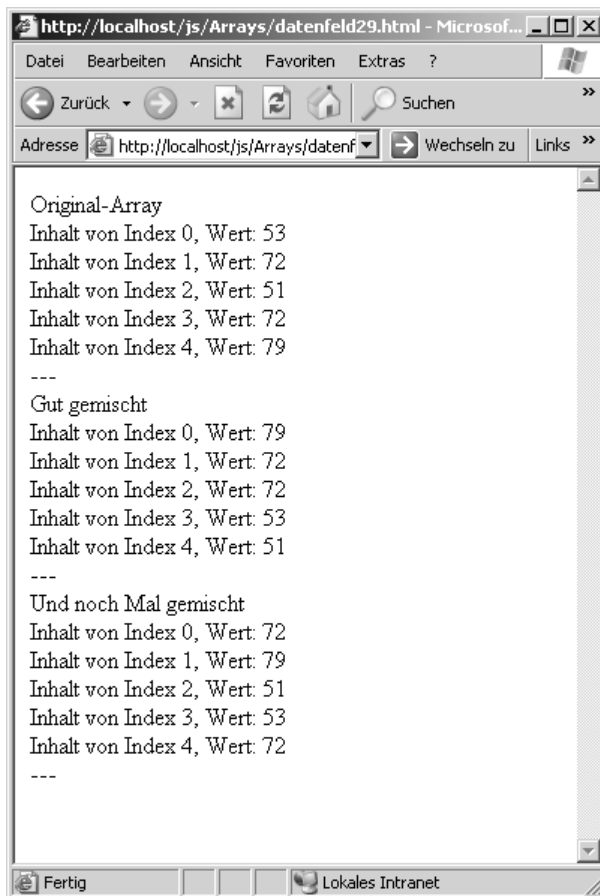


Abbildung 245: Einmal das Original und zwei Mal die gemischten Versionen

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);

```

Listing 365: Ein erster Versuch einer Lottoziehung

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

```
14 }  
15 ausgabe(a);  
16 </script>  
17 </html>
```

Listing 365: Ein erster Versuch einer Lottoziehung (Forts.)

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()`.

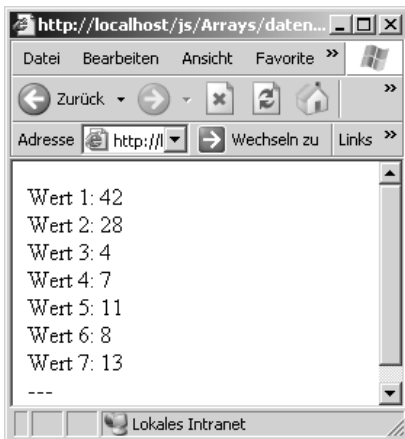


Abbildung 246: Die Lottozahlen von nächster Woche – wenn die gezogen werden und Sie gewinnen, will ich aber einen Anteil :-)

Das Beispiel scheint ganz gut zu funktionieren, aber lassen Sie die Ziehung nun mehrfach laufen.

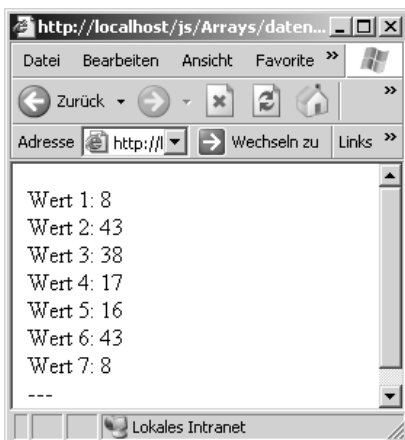


Abbildung 247: Na so was – die 8 wurde gleich zwei Mal gezogen.

Es wird Ihnen auffallen, dass bei manchen Ziehungen Zahlen mehrfach gezogen werden können. Und das passt natürlich nicht zu der simulierten Ziehung der Lottozahlen. 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     else {
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 366: Eine funktionierende Simulation einer Lottoziehung ohne Mehrfachziehung einer Zahl

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 so programmtechnisch 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 zwingenden 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 genau so viele Elemente aus dem Datenfeld gelöscht werden, wie der zweite Parameter angibt, sofern Sie keine weiteren Parameter angeben¹⁵. Wir kappen also das Datenfeld `a` auf sieben Elemente, was auch programmtechnisch ein echtes »Vergessen« bedeutet. Damit kann die Ausgabefunktion in Zeile 30 wie gewohnt das vollständige Datenfeld übergeben bekommen.

15. Geben Sie jedoch weitere Parameter an, werden die bisherigen Inhalte des Datenfelds ab dem Startindex durch diese Inhalte ersetzt.

Stringmanipulation

Die Manipulation von Texten gehört sicher zu den wichtigsten Aktionen bei komplexeren Skripten. Gerade im Zusammenhang mit Formulareingaben, denn diese werden rein als Strings weiter verarbeitet. Aber auch bei vielen anderen Vorgängen wie dem dynamischen Schreiben von Webseiten etc. Dem 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 Texte Objekte der Klasse `String` sind, können Sie mit 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 ein so genanntes Built-In-Objekt handelt. Eine Zuweisung 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 notwendig¹.

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 können.

141 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 komplexer Funktionen herauszufinden. 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.

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

HTML

Events

Beispiel (*str1.html*):

```
01 <html>
02 <body>
03 <script language="Javascript">
04   document.write(
05     "Wie lange ist der Text : " +
06     "Wie lange 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 367: Abfrage der Länge eines Strings

In Zeile 6 fragen wir unmittelbar die Länge von einem Stringliteral per Punktnotation ab. In Zeile 8 wird hingegen eine Stringvariable 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.

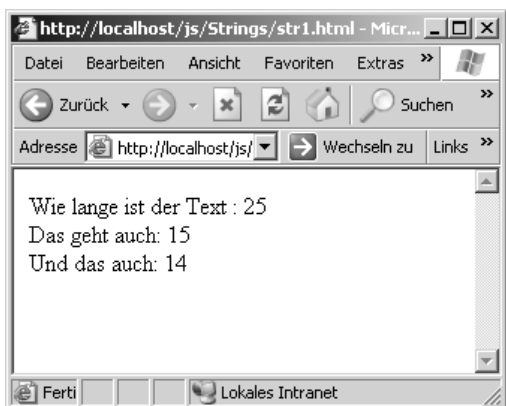


Abbildung 248: Die Länge von Strings

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 ein Setzen der Eigenschaft über eine Wertzuweisung keine Wirkung. Leider reagieren die Browser auf so eine unsinnige Anweisung nicht mit einer Fehlermeldung, sondern sie ignorieren sie einfach.

142 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 `ß`) 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 Sonderzeichen 123 #** ö ä??<br>");
08   document.write(
09     "Ein String mit Groß- und Kleinschreibung und Sonderzeichen 123 #** ö
    ä??<br>".toUpperCase());
10 </script>
11 </body>
12 </html>
```

Listing 368: 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. Zeile 9 zeigt die Wirkung auf Zeichen, die keine Buchstaben sind. Sie bleiben unverändert.

Hinweis

Beachten Sie auch das verwandte Rezept »Wie kann ich die Buchstaben in einem String in Kleinbuchstaben konvertieren?« auf Seite 542.

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

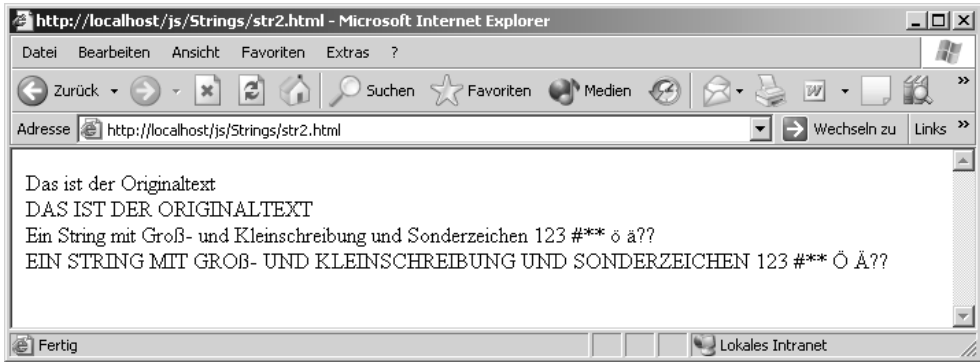


Abbildung 249: Die Wirkung von `toUpperCase()`

143 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 Benutzereingaben², die Sie mit einem vorgegebenen Text vergleichen wollen. In der `String`-Klasse finden Sie die Methode `toLowerCase()`, um alle Buchstaben (inklusive deutscher Umlaute) in einem vorangestellten String in Kleinbuchstaben umzuwandeln. Zeichen, die keinen Buchstaben sind, bleiben unverändert.

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 369: 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 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 Zahlen unverändert bleiben.

2. Zum Beispiel Formulareingaben.

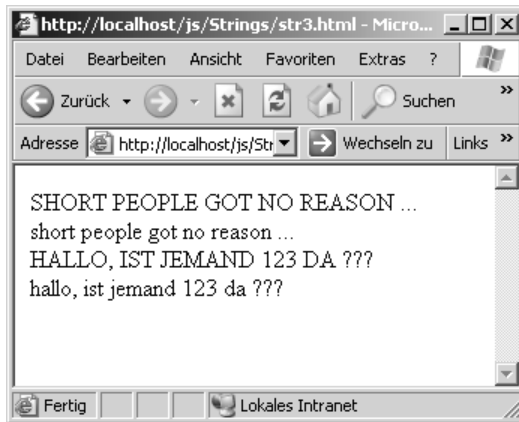


Abbildung 250: Die Wirkung von toLowerCase()

Hinweis

Beachten Sie auch das verwandte Rezept »Wie kann ich die Buchstaben in einem String in Großbuchstaben konvertieren?« auf Seite 541.

144 Wie ermittle ich das Zeichen, das in einem String an einer bestimmten Stelle steht?

Für verschiedene Anwendungen ist es gewünscht, bei Texten ein Zeichen an einer bestimmten Stelle abzufragen, ob beispielsweise 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 370: Einsatz von charAt()

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

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 hinaus greift.

Hinweis

Beachten Sie auch das verwandte Rezept »Wie ermittle ich die Unicode-Kodierung von dem Zeichen, das in einem String an einer bestimmten Stelle steht?« auf Seite 544.

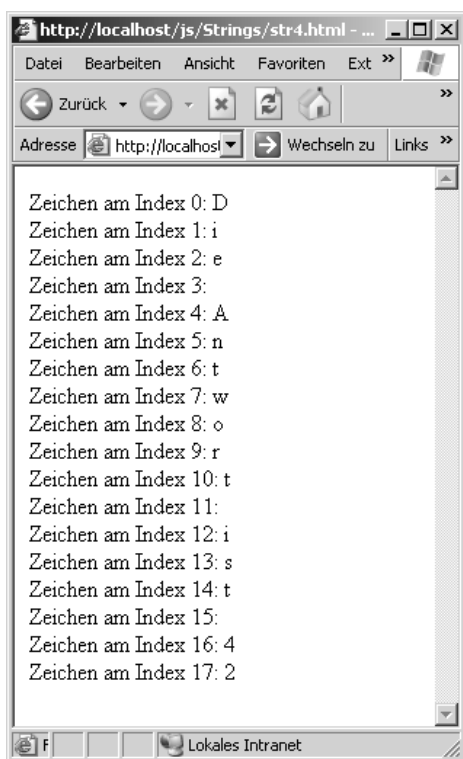


Abbildung 251: Das jeweilige Zeichen an dem angegebenen Index wird ausgegeben.

145 Wie ermittle ich die Unicode-Kodierung von einem bestimmten Zeichen, das in einem String 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.

Beachten Sie auch das verwandte Rezept »Wie ermittle ich das Zeichen, das in einem String an einer bestimmten Stelle steht?« auf Seite 543.

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 371: 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 Anweisung `t.charCodeAt(i)` liefert bei jedem Schleifendurchlauf die Unicode-Kodierung des jeweiligen Zeichens.

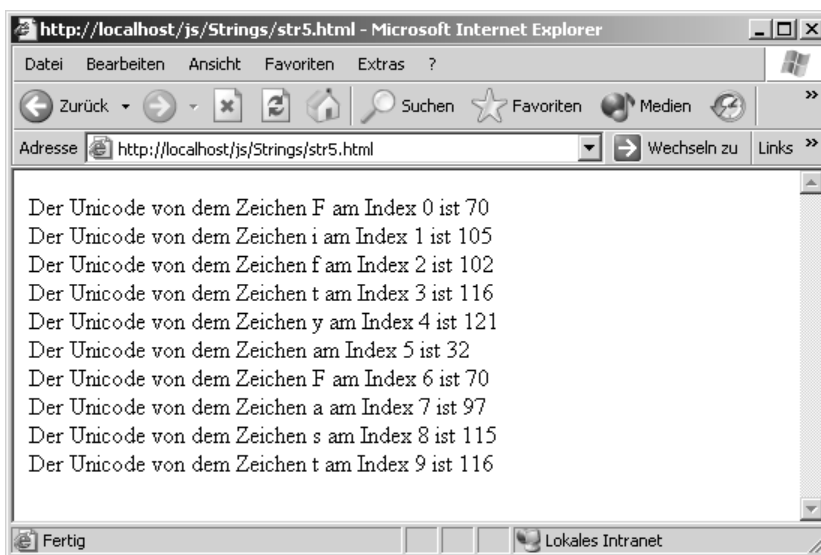


Abbildung 252: Das jeweilige Zeichen an dem angegebenen Index wird ausgegeben.

Für Buchstaben des deutschen Alphabets und Zahlen stimmt die Unicode-Kodierung mit der ASCII-Kodierung überein.

146 Wie verbinde ich zwei oder mehr Strings zu einem neuen String?

Zum Verbinden von Strings haben Sie in JavaScript (mindestens) zwei sinnvolle Möglichkeiten. Sie können beispielsweise den Plusoperator zum Verketteten von Strings verwenden. Dieser hat den zusätzlichen Effekt, dass eventuell mit einem String verkettete primitive Datentypen automatisch vor der Verbindung in Strings verwandelt werden. Alternativ können Sie die Methode `concat()` verwenden, die zwei oder mehr Strings zu einem neuen String verbindet. Genau genommen werden der oder die als Parameter angegebenen Strings³ 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 372: String-Verkettungen

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 `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.

3. Mehrere Parameter werden wie üblich mit Kommata getrennt.

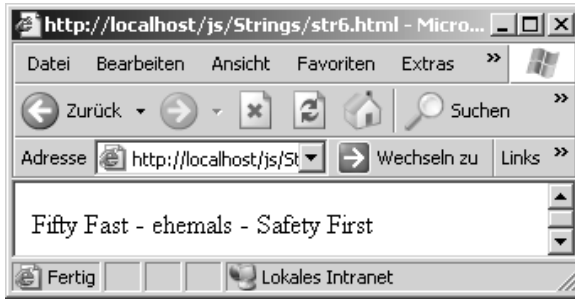


Abbildung 253: Nur ein String

147 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 in der Verarbeitung von Strings, denn Sie stellen beim Aufruf keinen String als Objekt voran, sondern explizit den Namen der Klasse `String`⁴. 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 373: Die Ausgabe des ASCII-Zeichensatzes mit `fromCharCode()`

Das Beispiel gibt die Unicode- beziehungsweise ASCII-Codes von 0 bis 255 aus und jeweils das zugehörige Zeichen, das über `fromCharCode(i)` als String ermittelt wird.

4. So etwas nennt man dann eine Klassenmethode. In JavaScript ist die Verwendung von solchen Klassenmethoden nicht so üblich, wie es etwa in Java der Fall ist.

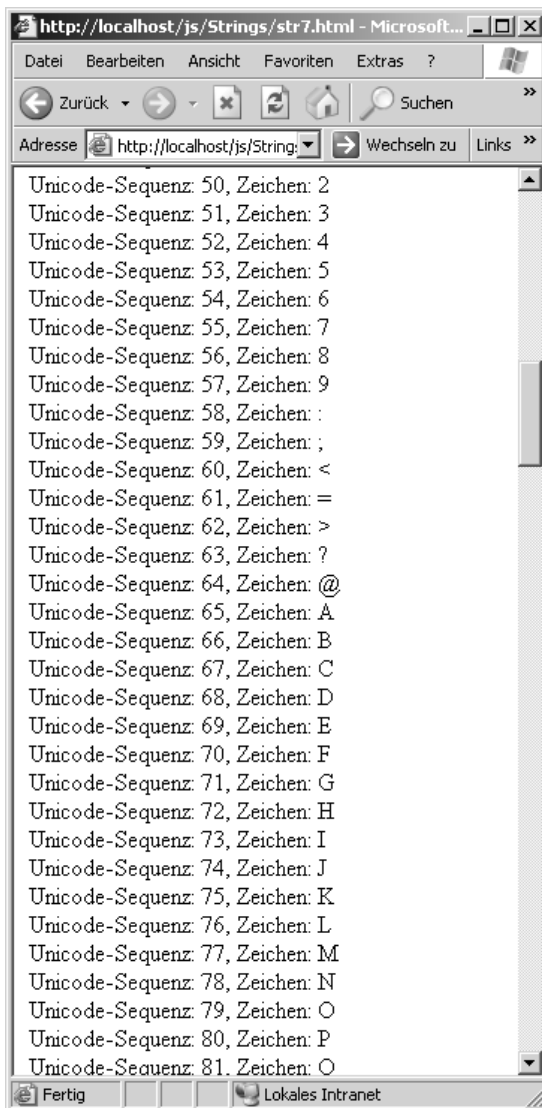


Abbildung 254: Die Unicode- beziehungsweise ASCII-Sequenz und das zugehörige Sonderzeichen, soweit der Browser es darstellen kann

Lassen Sie uns noch ein weiteres Beispiel durchspielen, was ganz unterhaltsam ist. Kennen Sie das Experiment, in dem 1000 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 Affen-ergüssen auftaucht. Das zweite Beispiel zur Demonstration der Methode `fromCharCode()` simuliert die Affenhorde.

Beispiel (*str7a.html*):

```
01 <html>
02 <body>
03 <script language="Javascript">
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 374: Ausgabe von zufälligen Buchstaben und darstellbaren Sonderzeichen

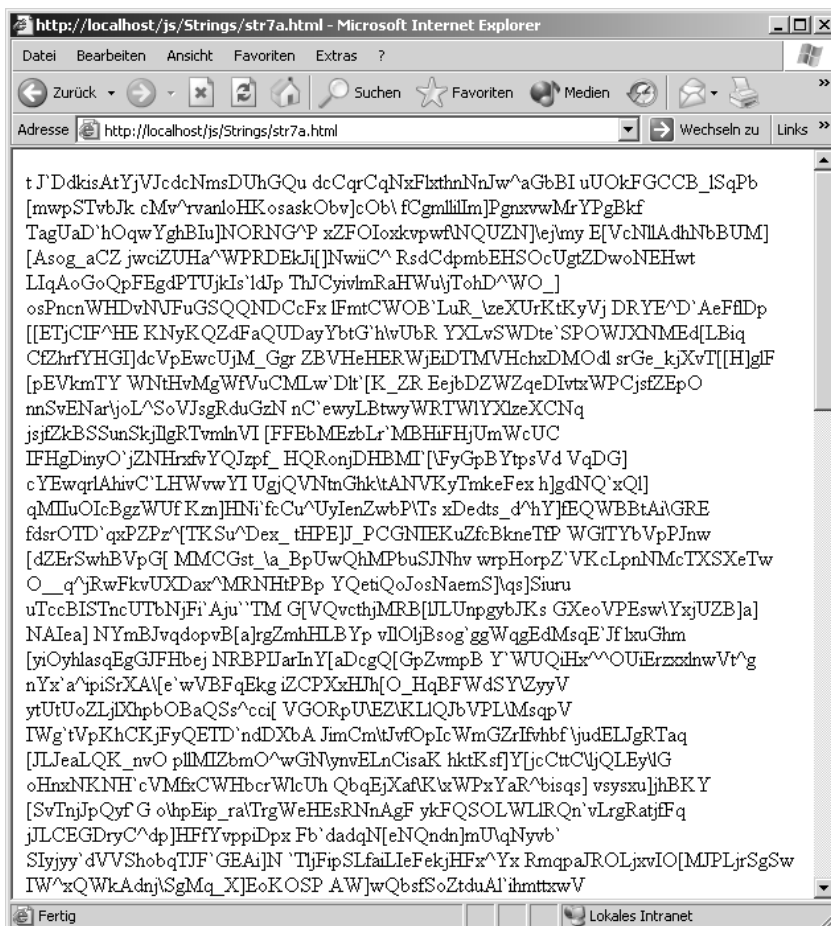


Abbildung 255: Was hat Ihre Affenhorde geschrieben?

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`-Objekts⁵ 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 ganzzahlig. Mit `Math.round()` schneiden wir den Nachkommateil ab und erhalten damit die notwendigen ganzen Zahlen. 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 mit in Zeile 6 mit `document.write(String.fromCharCode(z))`; das Zeichen ausgegeben, das dem ermittelten Code entspricht. Das Einfügen von einem Leerzeichen nach jeweils 25 Zeichen soll nur die Gestaltung etwas verbessern⁶.

148 Wie kann ich in Strings HTML-Formatierungen vornehmen?

Typischerweise werden Sie mit JavaScript Texte generieren, die in einem Webbrowser angezeigt werden sollen. In vielen Fällen sollen diese mit HTML formatiert 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>
```

Listing 375: Manuelles Einfügen von HTML-Tags in Strings

-
5. Ein Built-In-Objekt von JavaScript
 6. Die Affen werden sicher nicht mitzählen und nach 25 Zeichen regelmäßig die Leertaste drücken ;-).

```
17 </body>
18 </html>
```

Listing 375: Manuelles Einfügen von HTML-Tags in Strings (Forts.)

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.

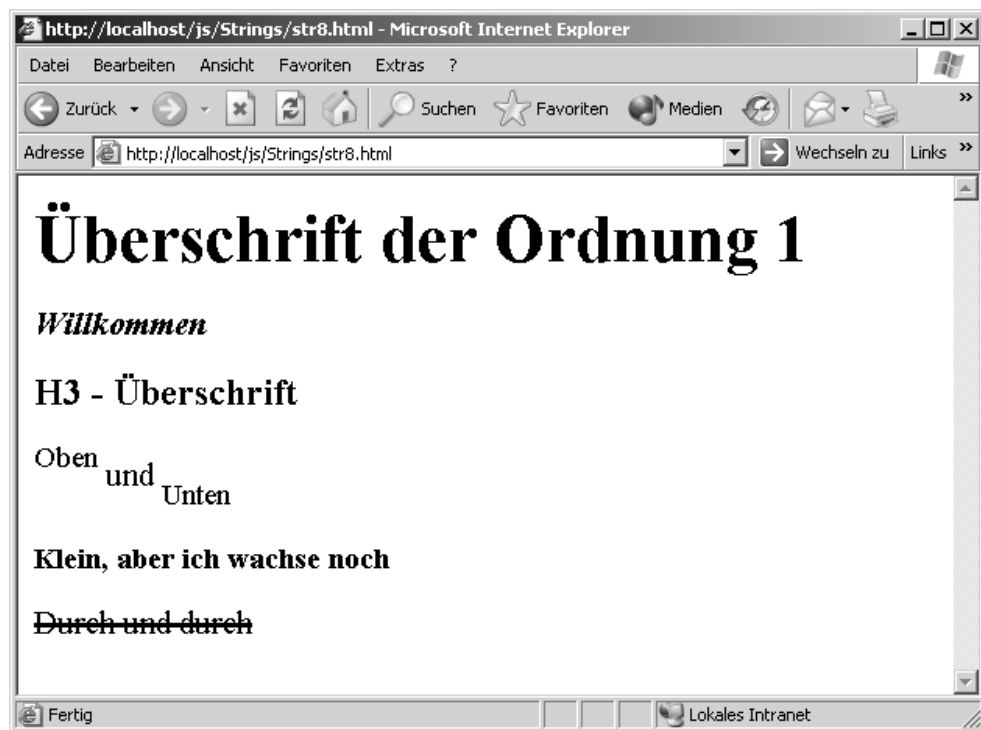


Abbildung 256: Mit JavaScript generierte HTML-Anweisungen und Text

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
<code>anchor()</code>	Diese Methode generiert ein Verweisziel (einen Anker) innerhalb einer HTML-Datei, der über einen Hyperlink angesprochen werden kann (einen Container der Form <code>[Text]</code>). Als Parameter wird ein Name für den Anker übergeben, der dann bei einem Hyperlink mit vorangestelltem Gatterzaun (#) als Sprungziel angegeben wird.
<code>big()</code>	Diese Methode setzt einen vorangestellten String in den HTML-Container <code><big></code> . Dieser bewirkt in den meisten Browsern eine größere Schrift.
<code>blink()</code>	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 diesen HTML-Tag nicht, und es ist merkwürdig, dass sich diese Methode in der String-Klasse gehalten hat.
<code>bold()</code>	Diese Methode setzt einen vorangestellten String in den HTML-Container <code></code> . Dieser bewirkt in den meisten Browsern eine fette Schrift.
<code>fixed()</code>	Diese Methode setzt einen vorangestellten String in den HTML-Container <code><tt></code> . Dieser bewirkt in den meisten Browsern eine dicktengleiche (gesperrte) Schrift.
<code>fontcolor()</code>	Diese Methode setzt einen vorangestellten String in den HTML-Container <code> ... </code> . Damit legen Sie die Schriftfarbe entsprechend fest. Die gewünschte Farbe wird der Funktion als String-Parameter übergeben, entweder in hexadezimaler Schreibweise oder als Farbname.
<code>fontsize()</code>	Diese Methode setzt einen vorangestellten String in den HTML-Container <code> ... </code> . 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 sehr kleine Schrift, 3 ist Normalschrift und 7 ist sehr große Schrift.
<code>italics()</code>	Diese Methode setzt einen vorangestellten String in den HTML-Container <code><i></code> . Das bewirkt eine kursive Schrift.
<code>link()</code>	Die Methode erzeugt einen Hyperlink (einen Container der Form <code>[Verweistext]</code>). Der vorangestellte String ist die in der Webseite sichtbare Beschriftung des Hyperlinks. Das Verweisziel wird als Parameter übergeben.
<code>small()</code>	Diese Methode setzt einen vorangestellten String in den HTML-Container <code><small></code> . Dieser bewirkt in den meisten Browsern eine kleinere Schrift.

Tabelle 8: Die String-Methoden zur HTML-Formatierung

Methode	Beschreibung
<code>strike()</code>	Diese Methode setzt einen vorangestellten String in den HTML-Container <code><strike></code> . Dieser bewirkt in den meisten Browsern einen durchgestrichenen Text.
<code>sub()</code>	Diese Methode setzt einen vorangestellten String in den HTML-Container <code><sub></code> . Dieser bewirkt in den meisten Browsern eine tiefer gestellte Schrift.
<code>sup()</code>	Diese Methode setzt einen vorangestellten String in den HTML-Container <code><sup></code> . Dieser bewirkt in den meisten Browsern eine hochgestellte Schrift.

Tabelle 8: 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();
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 376: Die Anwendung von String-Methoden zur HTML-Formatierung

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.

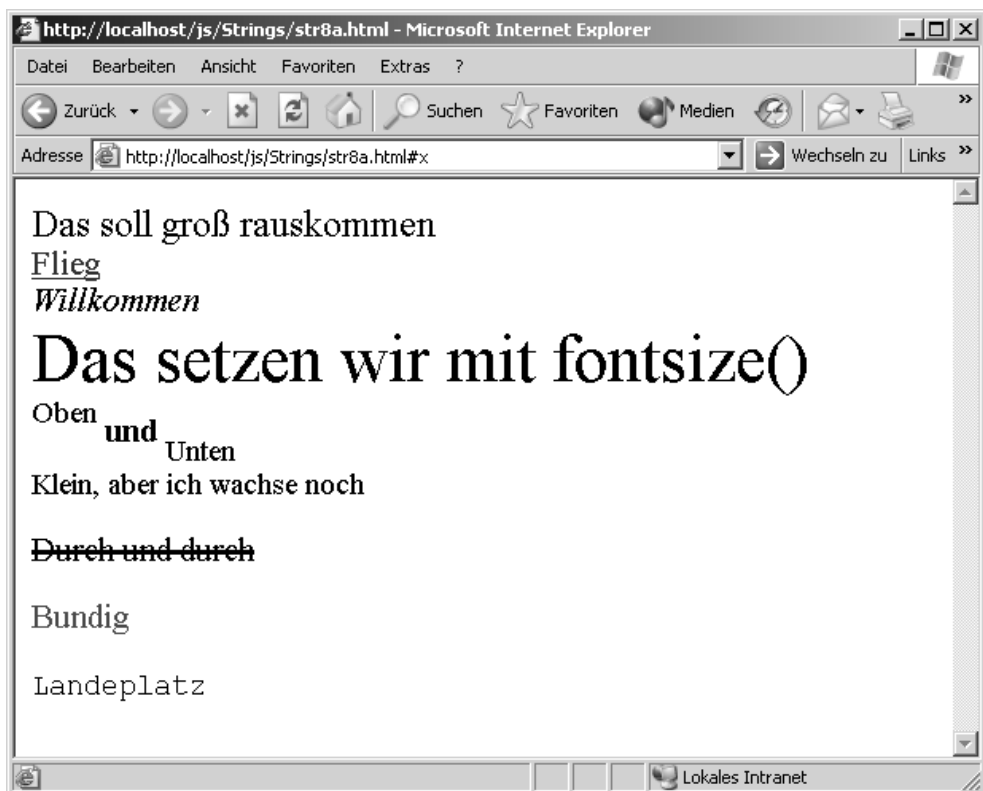


Abbildung 257: Die Auswirkungen der per JavaScript generierten HTML-Formatierungen

Tipp

Leider hat es sich in modernen Browsern durchgesetzt, dass auch ein reiner Anwender mit der Ansicht des Quelltexts über die browsereigene Anzeige⁷ genau den Code sieht, den der Ersteller der Seite auch eingegeben hat. Also im Fall von JavaScript sieht jeder Anwender auf Wunsch den vollständigen JavaScript-Code, wenn die Seite in seinem Webbrowser geladen wurde.

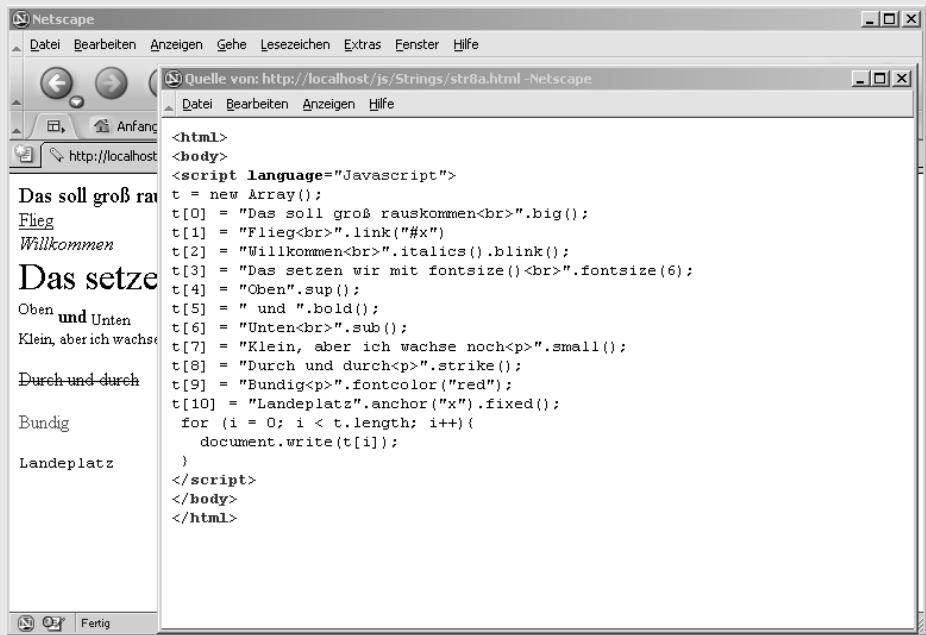


Abbildung 258: Beim Ansehen des Quellcodes sieht ein Anwender in den modernen Browsern auch den erzeugenden JavaScript-Code – hier im Navigator 7.1.

Das war nicht immer so, denn beispielsweise im Netscape Navigator vor der Version 6 konnte ein Anwender bei der Ansicht des Quelltextes einer Webseite mit dem browser-eigenen 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 per Skript auf Clientseite erst generiert wurde.



Abbildung 259: 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.

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 in dem 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 aufgepfropft. Und dieser ist halt ein dummer Texteditor und kann nicht mit entsprechenden Einschränkungen sicher gemacht werden. Durch die Marktdominanz von Microsoft und die damit aufgebauten Erwartungen des Anwenders haben die anderen Browserhersteller leider nachgezogen, und deren Anzeigemodule für den Quelltext zeigen mittlerweile ebenfalls den echten Quelltext an. Das hat indessen neben der leichteren Möglichkeit zu 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 einen Netscape Navigator 4.7 installiert, um so ein Feature zur Verfügung zu haben. Leider kommt aber diese Version mit nicht allen neueren Webtechniken zurecht.

Hinweis

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.

149 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 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]` anzuweisen, erst ab der spezifizierten Stelle in der Zeichenkette mit der Suche zu beginnen. Ziemlich anlog 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.

7. Im Internet Explorer beispielsweise mit dem Befehl `[Ansicht->Quelltext]` und im Netscape Navigator mit `[Anzeigen->Seitenquelltext]`.

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 377: Suche mit indexOf() und search()

In den Zeilen 7 und 10 wird mit der Methode `indexOf()` nach dem Vorkommen von 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. Diese sollen in der Webseite mit ausgegeben werden und müssen deshalb maskiert werden.



Abbildung 260: Die Suche nach Strings

Beachten Sie auch die verwandten Rezepte »Wie kann ich mit regulären Ausdrücken in Strings suchen und sie ersetzen?« auf Seite 560 und »Wie kann ich das letzte Vorkommen eines Zeichens oder einer Zeichenkette innerhalb eines Strings ermitteln?« auf Seite 558.

150 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 Benutzereingaben⁸. Sehr nützlich ist dabei die Methode `lastIndexOf([Suchstring], [Nummer])`. Mit der Methode durchsuchen Sie den vorangestellten String nach dem letzten 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 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 378: Suche nach dem letzten Vorkommen von dem Buchstaben 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.

8. Ein Beispiel ist, wenn Sie die Struktur von einem DNS-Namen analysieren wollen und den Toplevel suchen. Dieser wird nach dem letzten Punkt zu finden sein.

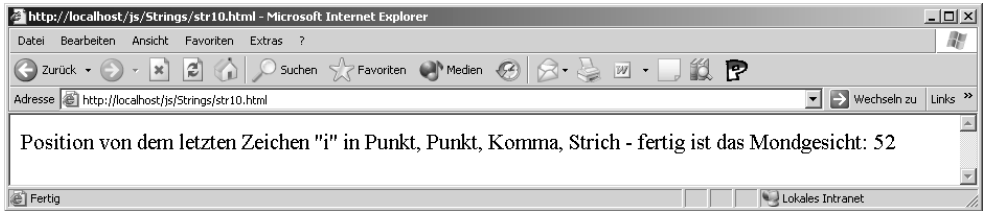


Abbildung 261: Die Suche nach dem letzten Vorkommen von einem Zeichen

Hinweis

Beachten Sie auch die verwandten Rezepte »Wie kann ich mit regulären Ausdrücken in Strings suchen und sie ersetzen?« auf Seite 560 und »Wie kann ich das erste Vorkommen eines Zeichens oder einer Zeichenkette innerhalb eines Strings ermitteln?« auf Seite 556.

151 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 379: Ersetzen von Textpassagen

In Zeile 4 wird ein String eingeführt, und in Zeile 6 ersetzen wir einen Teil der String-Variablen `t` mit der `replace()`-Methode.



Abbildung 262: Und schon hat man einen neuen Namen.

152 Wie kann ich mit regulären Ausdrücken in Strings suchen und sie ersetzen?

Über die `String`-Klasse stehen Ihnen mehrere Techniken bereit, 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 wollen. *Beachten Sie dazu die Rezepte »Wie kann ich innerhalb eines Strings einen Text ersetzen?« auf Seite 559, »Wie kann ich das letzte Vorkommen eines Zeichens oder einer Zeichenkette innerhalb eines Strings ermitteln?« auf Seite 558 und »Wie kann ich das erste Vorkommen eines Zeichens oder einer Zeichenkette innerhalb eines Strings ermitteln?« auf Seite 556.*

Die Möglichkeiten zur Suche in Texten gehen aber weiter, wenn Sie so genannte *reguläre Ausdrücke* einsetzen. 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 damit ist jedoch nicht ganz trivial. 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 auch für die Suche nach gewöhnlichen Strings zu verwendenden `search()`- sowie `replace()`-Methode der `String`-Klasse unterstützt als 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()`⁹, `exec()` und `test()` zur Verfügung stellt.

9. Mit dieser Methode können Sie ein bereits erzeugtes `RegExp`-Objekt kompilieren und damit die Performance verbessern. Wir werden die Methode aber nicht weiter verfolgen.

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 380: Eine Variable, die einen regulären Suchausdruck in Schrägstrichen zugewiesen bekommt

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 ebenfalls das `RegExp`-Objekt 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 381: 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 bis zu neun solcher Suchergebnisse pro Abfrage so speichern.

Die folgende Tabelle gibt die wichtigsten Steuerzeichen an, mit denen Sie einen regulären Ausdruck zusammensetzen können. Die Steuerzeichen können in einem Ausdruck kombiniert werden, sofern keine Widersprüche ausgelöst werden und mit dem Pipe-Symbol `|` als Oder-Verbindung mehrerer Ausdrücke in einem Pattern verwendet werden.

Steuerzeichen	Beispielpattern	Beschreibung
	/in/	Wenn Sie einen reinen Suchtext in Schrägstriche notieren, ist das für die meisten Suchsituationen äquivalent mit der Angabe eines normalen Texts in Hochkommata. Notwendig ist dieser Umweg über einen regulären Ausdruck nur dann, wenn explizit ein regulärer Ausdruck in einer Methode gefordert wird. Das Suchen nach /in/, führt zu Treffern in allen Strings, in denen diese Textpassage vorkommt. Beispielsweise in "in", "innen", "alleine", "kein" oder "worin".
\$	/in\$/	Die Notation des Dollarzeichens (ohne vorangestelltes Backslash) am Ende eines Suchstrings führt zu Treffern in allen Strings, wo diese Textpassage am Ende vorkommt. Beispielsweise in "in", "kein" oder "worin". Aber nicht in den Strings "innen" oder "alleine", denn hier steht "in" nicht am Ende.
*	/in*/	Die Notation des Sterns bedeutet, dass das letzte Zeichen vor dem Stern kein Mal oder beliebig oft hintereinander stehen kann.
.	/..in/	Die Notation eines Punktes fordert an der Stelle vor dem Suchbegriff mindestens ein beliebiges Zeichen. Notieren Sie einen Punkt, muss vor dem Suchbegriff mindestens ein anderes Zeichen in dem durchsuchten Ausdruck stehen. Notieren Sie zwei Punkt (beispielsweise /..in/), müssen mindestens zwei beliebige Zeichen dort vorhanden sein und so fort. Die Suche nach /...in/ führt beispielsweise zu Treffern in "alleine" und "worin". Aber nicht in den Strings "innen" oder "in".
\(	/(in)/	Die Notation eines Suchbegriffs in Klammern führt dazu, dass die Treffer intern gespeichert werden und über die internen Variablen \$1 bis \$9 verfügbar gemacht werden. Darauf greifen Sie mit RegExp.\$1 bis RegExp.\$9 zu.
\b	/\bin/	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 Worts stehen (die Suche nach /\bin/ führt beispielsweise zu Treffern in "innen" und "in", jedoch nicht in den Strings "alleine" oder "worin") und nachgestellt am Ende (die Suche nach /in\b/ führt beispielsweise zu Treffern in "worin" und "in", jedoch nicht in den Strings "innen", "alleine" oder "hinein").

Tabelle 9: Reguläre Ausdrücke zum Suchen in Strings

Steuerzeichen	Beispielpattern	Beschreibung
<code>\B</code>	<code>/in\B/</code>	Die Notation von <code>\B</code> (achten Sie auf die Großschreibung) sucht den vorangestellten oder nachfolgenden Suchstring und führt zu einem Treffer, wenn dieser nicht an der entsprechenden Wortgrenze steht. Vorangestellt darf der Suchstring nicht am Anfang eines Worts stehen (die Suche nach <code>/\B/</code> führt beispielsweise zu keinen Treffern in "innen" und "in", jedoch in den Strings "alleine" oder "worin" – an der hinteren Wortgrenze kann der Suchbegriff stehen) und nachgestellt zu keinen Treffern am Ende eines Worts (die Suche nach <code>/in\B/</code> führt beispielsweise zu keinen Treffern in "worin" und "in", jedoch in den Strings "innen", "alleine" oder "hinein" – an der vorderen Wortgrenze darf der Suchbegriff stehen).
<code>\d</code>	<code>/\d/</code>	Die Notation von <code>\d</code> (achten Sie auf die Kleinschreibung) sucht nach einer beliebigen ganzen Zahl.
<code>\D</code>	<code>/\D/</code>	Die Notation von <code>\D</code> (achten Sie auf die Großschreibung) sucht nach einem beliebigen Zeichen, das keine Ziffer ist.
<code>\f</code>	<code>/\f/</code>	Sie können in regulären Ausdrücken nach den auch in HTML oder JavaScript üblichen Maskierungen für Sonderzeichen suchen. Diese Sequenz führt zur Suche nach einem Seitenvorschub.
<code>\n</code>	<code>/\n/</code>	Sie können in regulären Ausdrücken nach den auch in HTML oder JavaScript üblichen Maskierungen für Sonderzeichen suchen. Diese Sequenz führt zur Suche nach einem Zeilenvorschubzeichen.
<code>\r</code>	<code>/\r/</code>	Sie können in regulären Ausdrücken nach den auch in HTML oder JavaScript üblichen Maskierungen für Sonderzeichen suchen. Diese Sequenz führt zur Suche nach einem Wagenrücklaufzeichen.
<code>\s</code>	<code>/\s/</code>	Die Notation von <code>\s</code> (achten Sie auf die Kleinschreibung) führt zu Treffern in allen Strings, wo irgendeine Art von White Space vorkommt. Das sind Leerzeichen und alle maskierten Sequenzen für diese Zeichen (also Seitenvorschub, Zeilenschaltung, Tabulator und vertikaler Tabulator – <code>\f</code> , <code>\n</code> , <code>\t</code> und <code>\v</code>).
<code>\S</code>	<code>/\S/</code>	Die Notation von <code>\S</code> (achten Sie auf die Großschreibung) sucht nach einem beliebigen Zeichen, das kein White Space ist. Als White Space werden Leerzeichen und alle maskierten Sequenzen für diese Zeichen (also Seitenvorschub, Zeilenschaltung, Tabulator und vertikaler Tabulator – <code>\f</code> , <code>\n</code> , <code>\t</code> und <code>\v</code>) bezeichnet.

Tabelle 9: Reguläre Ausdrücke zum Suchen in Strings (Forts.)

Steuerzeichen	Beispielpattern	Beschreibung
\t	/\t/	Sie können in regulären Ausdrücken nach den auch in HTML oder JavaScript üblichen Maskierungen für Sonderzeichen suchen. Diese Sequenz führt zur Suche nach einem Tabulator.
\v	/\v/	Sie können in regulären Ausdrücken nach den auch in HTML oder JavaScript üblichen Maskierungen für Sonderzeichen suchen. 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 nicht ein Unterstrich sind.
^	/^in/	Die Notation des ^-Zeichens am Anfang eines Suchstrings führt zu Treffern in allen Strings, wo diese Textpassage am Anfang vorkommt. Beispielsweise 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 nach dem Suchstring so oft, wie es 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 zur Ignorierung von Klein- und Großschreibung. Die Suche /in/i nach führt beispielsweise zu Treffern in "in" als auch "Innen".

Tabelle 9: 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 soll einfach die `search()`-Methode verwenden und in einem Datenfeld mit verschiedenen Texten als Inhalt die unterschiedlichen Steuerzeichen zum Aufbau von regulären Ausdrücken testen.

Beispiel (*str12.html*):

```

01 <html>
02 <body>
03 <script language="Javascript">
04 function suchereg(t,s) {
05     document.write(
06 "<table border width=500><tr><th>Text</th><th>Suche nach</th><th>Rückgabewert</th><th>Treffer</th></tr>");
07     for(i = 0; i < t.length; i++){
08         document.write(
09             "<tr><td>" + t[i] +
10             "</td><td>" + s + "</td><td>" +
11             t[i].search(s) + "</td>");
12             if(t[i].search(s) > -1) {
13                 document.write("<td><b>BINGO</b></td>");
14             }
15             else {
16                 document.write("<td>&nbsp;</td>");
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\nnirgendwohin";
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)/);

```

Listing 382: Die Anwendung verschiedener einfacher Suchpattern auf mehrere Texte

Grund-
lagen

Core

Formu-
lare

Aus-
nahme

Arrays

Strings

Datum

Platt-
form

DHTML

Events

```

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 382: 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 White Space 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 einen Beginn-Tag der Tabelle und eine erste Zeile mit den Spaltenüberschriften für jede einzelne Suchabfrage ("`<table border 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 hinauszuschießen (`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 machen¹⁰, 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><b>BINGO</b></td>"); } else { document.write("<td> </td>"); }`).

10. Aufgrund des Index ist es zwar klar, falls die Suche zu einem Treffer geführt hat (der Rückgabewert von `search()` ist größer -1), aber um es besser lesbar zu machen.

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 263: Die Treffer der Suchabfrage werden im Webbrowser in Tabellenform ausgegeben.

Schauen wir uns nun die einzelnen Ausgaben der jeweiligen Suchen zur Verdeutlichung an.

```
suchereg(t,/in/);
```

Listing 383: 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

Listing 384: Ergebnis der Suche nach einem reinen Text »in«

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 nirgendwohin	/in/	22	BINGO
09	?	/in/	-1	
10	Innen	/in/	-1	
11	eineindeutig	/in/	1	BINGO

Listing 384: Ergebnis der Suche nach einem reinen Text »in« (Forts.)

Dieses Suchpattern führt einfach zur Suche nach dem enthaltenen Text. Bis auf die Suche in dem Text ohne die Zeichenkette "in" (Ausgabezeile 9) 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 385: Zeile 34 – Suche nach »/.+in«

Die Suche führt zu folgender Ausgabe:

	Text	Suche nach	Rückgabewert	Treffer
01	in der Ruhe	/.+in/	-1	
02	innen drin	/.+in/	0	BINGO
03	alleine	/.+in/	0	BINGO
04	keine Zahl zwischen 5 und 9	/.+in/	0	BINGO
05	0 Bock auf 30 Grad in dem Schatten	/.+in/	0	BINGO
06	worin	/.+in/	0	BINGO
07	wo gehen wir hin?	/.+in/	0	BINGO
08	wir gehen nirgendwohin	/.+in/	10	BINGO
09	?	/.+in/	-1	
10	Innen	/.+in/	-1	
11	eineindeutig	/.+in/	0	BINGO

Listing 386: 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 die Zeichenkette, an 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 387: 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	0 Bock auf 30 Grad in dem Schatten	/in\$/	-1	
06	worin	/in\$/	3	BINGO
07	wo gehen wir hin?	/in\$/	-1	
08	wir gehen nirgendwohin	/in\$/	22	BINGO
09	?	/in\$/	-1	
10	Innen	/in\$/	-1	
11	eineindeutig	/in\$/	-1	

Listing 388: Ergebnis der Suche nach »/in\$/«

Die Notation des Dollarzeichens (ohne vorangestelltes Backslash) am Ende eines Suchstrings führt zu Treffern in allen Strings, wo 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 389: 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	0 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 nirgendwohin	/...in/	19	BINGO
09	?	/...in/	-1	
10	Innen	/...in/	-1	
11	eineindeutig	/...in/	1	BINGO

Listing 390: Ergebnis der Suche nach »/...in/«

Ein Punkt erfordert an der notierten Stelle ein beliebiges Zeichen. Die aktuelle Suche hier wird 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, sowie den Fall, in dem die Zeichenkette "in" in dem Textdatenfeldelement zwar vorkommt, aber mit großem I geschrieben wird¹¹ (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 391: Zeile 37 – Suche nach »/..in/«

Die Suche führt zu folgender Ausgabe:

	Text	Suche nach	Rückgabewert	Treffer
01	in der Ruhe	/..in/	-1	

Listing 392: Ergebnis der Suche nach »/..in/«

11. Zudem gibt es keine drei Zeichen davor.

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 nirgendwohin	/in/	21	BINGO
09	?	/in/	-1	
10	Innen	/in/	-1	
11	eineindeutig	/in/	0	BINGO

Listing 392: Ergebnis der Suche nach »/in/« (Forts.)

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) zu sehen sind, sowie den Fall, in dem die Zeichenkette "in" in dem Textdatenfeldelement zwar vorkommt, aber mit großem I geschrieben wird (Ausgabezeile 10)¹². 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". Ebenso sollten Sie hier den unterschiedlichen Index zu dem Fall der Suche nach "...in" im Fall zuvor beachten. Der Index gibt ja immer die Position des Beginns der Übereinstimmung mit dem gesamten Suchpattern (nicht nur des reinen Texts) an.

```
suchereg(t,/in/);
```

Listing 393: Zeile 38 – Suche nach »/^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/	-1	

Listing 394: Ergebnis der Suche nach »/^in/«

12. Zudem gibt es kein Zeichen davor.

04	keine Zahl zwischen 5 und 9	/^in/	-1	
05	0 Bock auf 30 Grad in dem Schatten	/^in/	-1	
06	worin	/^in/	-1	
07	wo gehen wir hin?	/^in/	-1	
08	wir gehen nirgendwohin	/^in/	-1	
09	?	/^in/	-1	
10	Innen	/^in/	-1	
11	eineindeutig	/^in/	-1	

Listing 394: Ergebnis der Suche nach »/^in/« (Forts.)

Die Notation des ^-Zeichens am Anfang eines Suchstrings führt zu Treffern in allen Strings, wo diese Textpassage am Anfang vorkommt. Die Suche wird folglich nur in denen 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 395: 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	0 Bock auf 30 Grad in dem Schatten	/\t/	-1	
06	worin	/\t/	-1	
07	wo gehen wir hin?	/\t/	8	BINGO
08	wir gehen nirgendwohin	/\t/	-1	
09	?	/\t/	-1	
10	Innen	/\t/	-1	
11	eineindeutig	/\t/	-1	

Listing 396: Ergebnis der Suche nach einem Tabulator

Die Sequenz `\t` führt zur Suche nach einem Tabulator. Obwohl der Browser den Tabulator nicht darstellt, zeigt die Ausgabezeilen 7 deutlich, dass in dem Text ein Tabulator sein muss.

```
suchereg(t, /\n/);
```

Listing 397: 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	0 Bock auf 30 Grad in dem Schatten	/\n/	-1	
06	worin	/\n/	-1	
07	wo gehen wir hin?	/\n/	-1	
08	wir gehen nirgendwohin	/\n/	9	BINGO
09	?	/\n/	-1	
10	Innen	/\n/	-1	
11	eindeutig	/\n/	-1	

Listing 398: Ergebnis der Suche nach einem Zeilenumbruch

Die Sequenz `\n` führt zur Suche nach einem Zeilenvorschubzeichen. Obwohl der Browser den Zeilenumbruch nicht darstellt, zeigt die Ausgabezeilen 8, dass in dem Text in diesem Textdatenfeldelement ein selbiger sein muss.

```
suchereg(t, /\s/);
```

Listing 399: 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

Listing 400: Ergebnis der Suche nach `»\s/«`

03	alleine	/\s/	-1	
04	keine Zahl zwischen 5 und 9	/\s/	5	BINGO
05	0 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 nirgendwohin	/\s/	3	BINGO
09	?	/\s/	0	BINGO
10	Innen	/\s/	-1	
11	eindeutig	/\s/	-1	

Listing 400: Ergebnis der Suche nach »\s/« (Forts.)

Die Notation von `/\s/` führt zu Treffern in allen Strings, wo irgendeine Art von White Space vorkommt. In den Ausgabezeilen 3, 6, 10 und 11 wurde kein White Space gefunden. Beachten Sie Ausgabezeile 9. Der Browser hat die führenden Leerzeichen in der Ausgabe wegoptimiert, aber die Suche sie selbstverständlich dennoch gefunden.

```
suchereg(t,/\S/);
```

Listing 401: 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	0 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 nirgendwohin	/\S/	0	BINGO
09	?	/\S/	4	BINGO

Listing 402: Ergebnis der Suche nach »\S/«

10	Innen	/\S/	0	BINGO
11	eindeutig	/\S/	0	BINGO

Listing 402: Ergebnis der Suche nach »S« (Forts.)

Die Notation von `/\S/` sucht nach dem ersten beliebigen Zeichen, das kein White Space ist. Ein solches Zeichen wird in unserem Beispiel in allen Textdatenfeldelementen gefunden. Das scheint trivial, aber dennoch gibt es eine interessante Stelle. Beachten Sie Ausgabezeile 9. Der Browser hat die führenden Leerzeichen in der Ausgabe wegoptimiert, aber die Suche sie wie im Fall zuvor selbstverständlich dennoch gefunden. Von Interesse ist jetzt aber der Trefferindex 4. Es müssen also drei White Space davor vorhanden sein. Das Fragezeichen ist das erste Zeichen, das kein White Space ist.

```
suchereg(t,/\w/);
```

Listing 403: 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	0 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 nirgendwohin	/\w/	0	BINGO
09	?	/\w/	-1	
10	Innen	/\w/	0	BINGO
11	eindeutig	/\w/	0	BINGO

Listing 404: 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 der Ausgabezeile 9 gefunden. Dort sind in dem Textdatenfeldelement nur drei Leerzeichen und ein Fragezeichen vorhanden.

```
suchereg(t,/\W/);
```

Listing 405: Zeile 44 – Suche nach »\W«

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	0 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 nirgendwohin	/\W/	3	BINGO
09	?	/\W/	0	BINGO
10	Innen	/in/	-1	
11	eineindeutig	/\W/	-1	

Listing 406: 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 White Space 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 407: 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

Listing 408: Ergebnis der Suche beim Ignorieren von Groß- und Kleinschreibung

03	alleine	/\in/i	4	BINGO
04	keine Zahl zwischen 5 und 9	/\in/i	2	BINGO
05	0 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 nirgendwohin	/\in/i	22	BINGO
09	?	/\in/i	-1	
10	Innen	/\in/i	0	BINGO
11	eineindeutig	/\in/i	1	BINGO

Listing 408: Ergebnis der Suche beim Ignorieren von Groß- und Kleinschreibung (Forts.)

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 409: Zeile 46 – Suche nach »/(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 nirgendwohin	/((in))/	22	BINGO
09	?	/((in))/	-1	
10	Innen	/((in))/	-1	
11	eineindeutig	/((in))/	1	BINGO

Listing 410: 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 das Suchpattern hier 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 werden 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 411: 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	0 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 nirgendwohin	/in\b/	22	BINGO
09	?	/in\b/	-1	
10	Innen	/in\b/	-1	
11	eindeutig	/in\b/	-1	

Listing 412: Ergebnis der Suche am Wortanfang

Die Notation von \b sucht den vorangestellten Suchstring am Anfang eines Worts. Das ist in den Ausgabezeilen 3 (die Zeichenkette "in" kommt nicht am Anfang eines Worts vor), 4 (die Zeichenkette "in" kommt nicht am Anfang eines Worts vor), 9 (die Zeichenkette "in" kommt gar nicht vor), 10 (die Zeichenkette "in" kommt zwar am Anfang eines Worts vor, wird aber mit großem I geschrieben) und 11 (die Zeichenkette "in" kommt nicht am Anfang eines Worts vor) nicht der Fall.

```
suchereg(t,/^\bin/);
```

Listing 413: 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 nirgendwohin	/\bin/	-1	
09	?	/\bin/	-1	
10	Innen	/\bin/	-1	
11	eindeutig	/\bin/	-1	

Listing 414: Ergebnis der Suche am Wortende

Die Notation von `\b` sucht den nachgestellten Suchstring am Ende eines Worts. Das ist nur in den Ausgabezeilen 1 (die Zeichenkette "in" steht natürlich in "in" selbst am Ende), 2 (die Zeichenkette "in" steht in "drin" am Ende) und 5 (die Zeichenkette "in" kommt als einzelnes Wort vor) der Fall.

```
suchereg(t,/in\b/);
```

Listing 415: 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	

Listing 416: Ergebnis der Suche, bei der der Suchstring nicht am Wortanfang stehen darf

06	worin	/in\B/	-1	
07	wo gehen wir hin?	/in\B/	-1	
08	wir gehen nirgendwohin	/in\B/	-1	
09	?	/in\B/	-1	
10	Innen	/in\B/	-1	
11	eindeutig	/in\B/	1	BINGO

Listing 416: Ergebnis der Suche, bei der der Suchstring nicht am Wortanfang stehen darf (Forts.)

Die Notation von \B führt zu einem Treffer in dem vorangestellten Suchstring, wenn dieser nicht am Anfang eines Words steht. Das ist nur in den Ausgabzeilen 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 417: 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 nirgendwohin	/\Bin/	22	BINGO
09	?	/\Bin/	-1	
10	Innen	/\Bin/	-1	
11	eindeutig	/\Bin/	1	BINGO

Listing 418: 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 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 419: 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 nirgendwohin	/\d/	-1	
09	?	/\d/	-1	
10	Innen	/\d/	-1	
11	eindeutig	/\d/	-1	

Listing 420: Ergebnis der Suche nach einer beliebigen ganzen Zahl

Die Notation von `\d` sucht den nach einer beliebigen ganzen Zahl. Eine solche ist nur in den Ausgabezeilen 4 und 5 zu finden.

```
suchereg(t,/\D/);
```

Listing 421: 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 nirgendwohin	/\D/	0	BINGO
09	?	/\D/	0	BINGO
10	Innen	/\D/	0	BINGO
11	eineindeutig	/\D/	0	BINGO

Listing 422: 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 White Space ist natürlich keine ganze Zahl.

Beispiel zur Durchsuchung eines Strings mit `exec()`

In dem folgenden Beispiel soll ein String unter Verwendung der `RegExp`-Methode `exec()` nach Übereinstimmung mit einem vorgegebenen Suchpattern getestet werden. 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 gefundene 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;
```

Listing 423: Anwendung von `exec()` und Auswertung der Werte in den resultierenden Eigenschaften

```

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>");
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 423: Anwendung von `exec()` und Auswertung der Werte in den resultierenden Eigenschaften

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 + "<br>");`).

In Zeile 9 und 10 folgt dann die Ausgabe der Trefferanzahl (`document.write("Anzahl der Treffer: " + erg.length + "<br>");`).

In Zeile 11 und 12 geben wir den Index des ersten Treffers aus (`document.write("Index des ersten Treffers: " + erg.index + "<br>");`) und in Zeile 13 und 14 die letzte gefundene Übereinstimmung (`document.write("Letzte gefundene Übereinstimmung: " + erg[0] + "<br>");`).

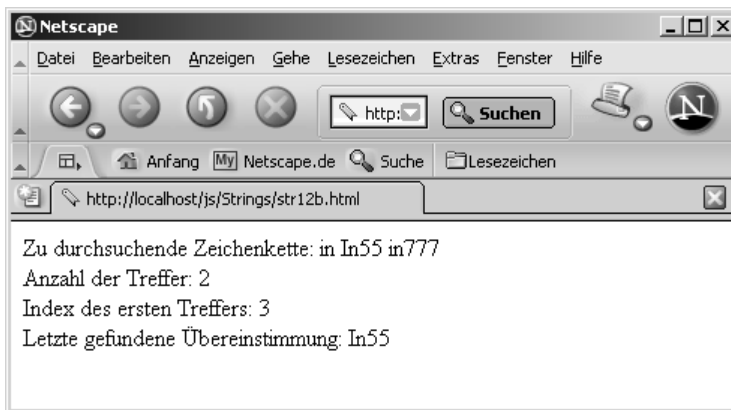


Abbildung 264: Anwendung von `exec()`

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.



Abbildung 265: Wenn `exec()` nicht erfolgreich war, gibt es einen Laufzeitfehler.

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 424: Abgesicherte Auswertung des Rückgabewerts von `exec()`

Beispiel zur Extrahierung der Versionsnummer eines Browsers mit `exec()` und Auswertung von geklammerten Teilen

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. Das wollen wir nutzen, um die Versionsnummer eines Webbrowsers zu ermitteln.

Über das Objekt `navigator` stehen Ihnen zahlreiche Informationen über das Clientprogramm eines Besuchers bereit. 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. Im Kapitel der Grundtechniken von JavaScript haben wir mit den Toplevel-Funktionen `parseFloat([Zeichenkette])` und `parseInt([Zeichenkette])` die Eigenschaft `navigator.appVersion` durchsucht und darauf gebaut, dass die Versionsinformation am Beginn von dem String 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 ebenfalls 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 darstellt¹³.

Beispiel (*str12c.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   browser = /(\d)/;
05   browser.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 425: 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.

13. Das Pattern wird nicht mehr funktionieren, wenn eine Browserversion zweistellig (oder noch größer) wird – das sollte aber die nächsten Jahre nicht der Fall sein.

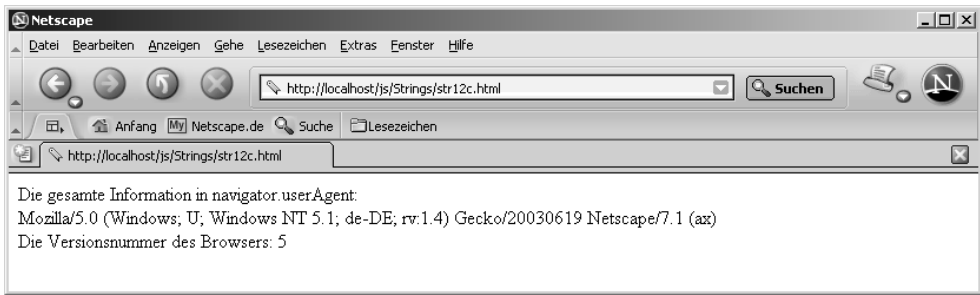


Abbildung 266: Extrahierung der Versionsnummer eines Webbrowsers mit einem regulären Ausdruck

Beispiel zum Test auf eine gültige E-Mail-Adresse mit test()

Wie im Kapitel zur Formularbehandlung ausführlich besprochen wurde, ist ein Hauptzweck 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. Dieses muss in jeder gültigen E-Mail-Adresse zwingend enthalten sein. Zudem muss vor dem Zeichen mindestens ein weiteres beliebiges Zeichen stehen und dahinter mindestens ein weiteres¹⁴. 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('.@.');
```

Listing 426: Eine Überprüfung der Gültigkeit einer E-Mail-Adresse mit test()

14. In der Praxis im Internet müssen sowohl vor als auch hinter dem @-Zeichen natürlich mehr 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.

```

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 426: 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 dreht den Wert in `false` um (`if(!regausdruck.test(feld.value)) {}`). Dann wird eine Fehlermeldung angezeigt (Zeile 8 und 9) und der Wert `false` zurückgegeben (Zeile 10)¹⁵.

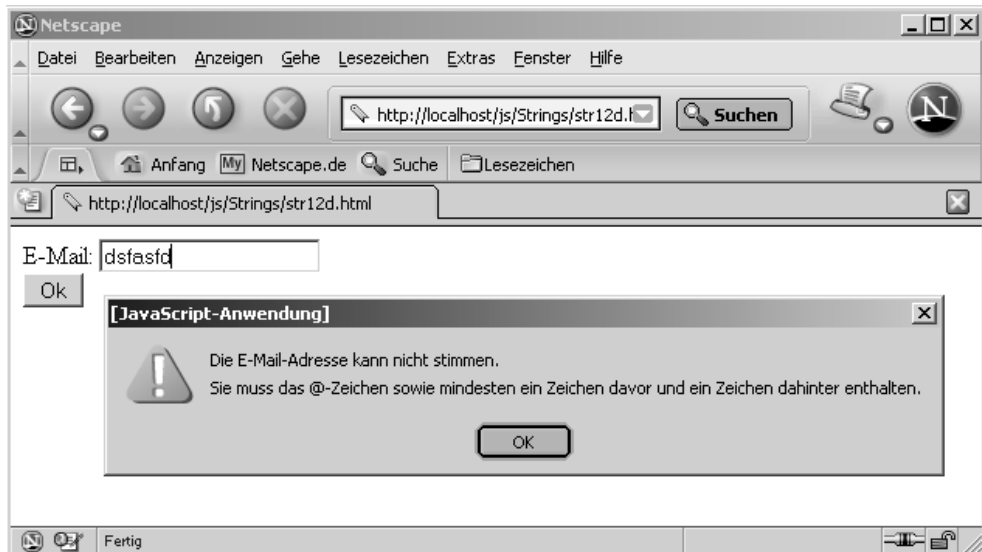


Abbildung 267: Mit `test()` wurde die E-Mail-Adresse überprüft.

15. Der Rückgabewert wird in diesem Beispiel nicht weiter ausgewertet.

Beispiel zum Sperren bestimmter Eingaben in einem Webformularfeld mit regulären Ausdrücken und der match()- beziehungsweise replace()-Methode

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). Sie können nun explizit bestimmte Inhalte verhindern oder sie ersetzen. Im Kapitel zum Umgang mit Webformularen finden Sie ein Rezept, wie bestimmte Inhalte ersetzt werden¹⁶. Das wollen wir hier mit regulären Ausdrücken in leicht veränderter Form realisieren.

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 427: HTML-Code mit »***« maskieren

16. Dabei wird jedes <-Zeichen und jedes >-Zeichen durch seine ungefährliche Kodierung ersetzt.

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.

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 einer 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 der Tag `<script>` ohne Parameter notiert wird¹⁷.

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 428: Eine Applet-Referenz

Mit `<applet\s` ist die Eingabe des Einleitungs-Tag und nachfolgenden Leerzeichens verhindert. Da eine Applet-Referenz immer einen Parameter benötigt, langt diese Angabe.

Im letzten Teil wird der `<object>`-Tag mit nachfolgendem Leerzeichen oder als reiner Tag `<object>` verhindert. Das unterbindet die Referenzierung von ActiveX-Controls.

Mit dem nachgestellten `ig` wird 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 Rückgabewert¹⁸ (`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.

17. Dann verwendet ein Webbrowser seine Defaultskriptsprache, und das ist so gut wie immer JavaScript beziehungsweise JavaScript.

18. Durch Kommata getrennt.

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 Zeile 10 bis 13 zeigen eine Fehlermeldung an (`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 zurück in das fehlerhafte Formularfeld (`feld.focus();`), Zeile 15 selektiert es (`feld.select();`), und Zeile 16 gibt den Wert `false` zurück (`return false;`).

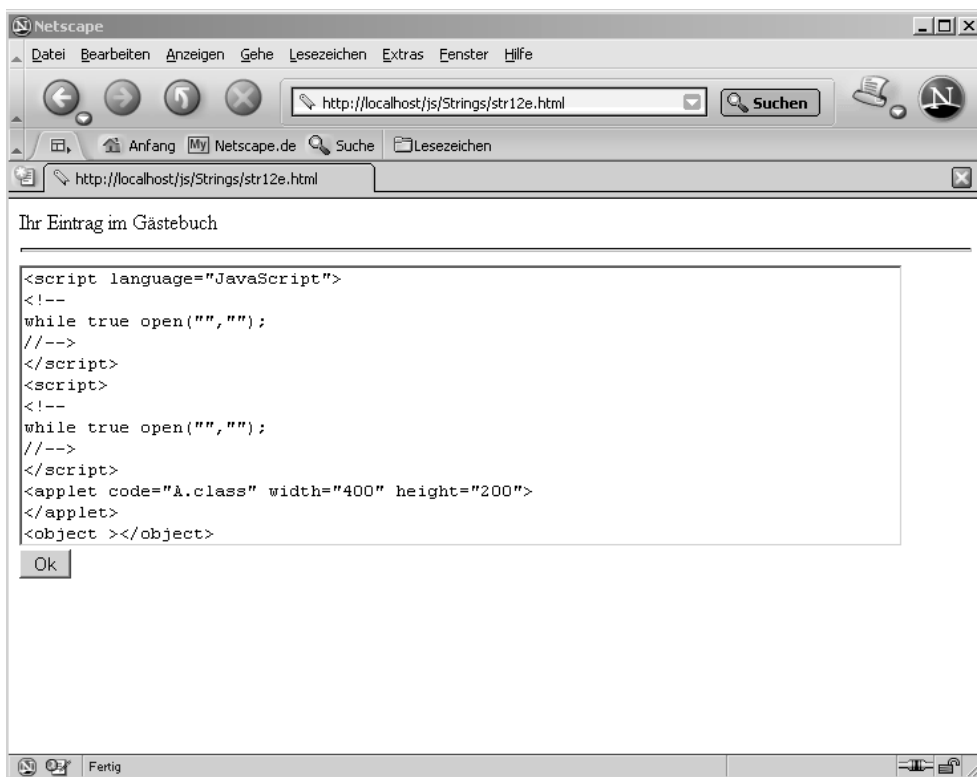


Abbildung 268: Viele kritische Befehle in dem Textarea-Feld

Ihr Eintrag im Gästebuch



Abbildung 269: 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 594.

153 Wie kann ich einen Teil eines Strings extrahieren (slice)?

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 bereits für diese Tätigkeit mehrere Methoden mit, und damit wird die Tätigkeit zum Kinderspiel.

Eine Möglichkeit zur Extrahierung von einem Teilstring 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(
```

Listing 429: Extraktion von Teilstrings mit `slice()`

```
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 429: Extraktion von Teilstrings mit `slice()` (Forts.)

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.

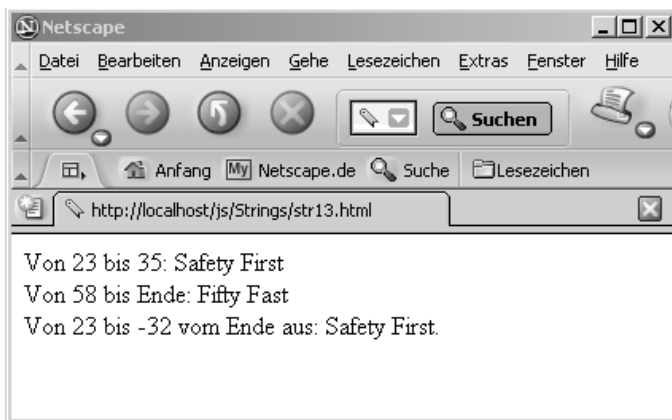


Abbildung 270: Extrahierung mehrerer Teilstrings

154 Wie erfolgt die Extrahierung mit `substr()`?

Eine weitere Möglichkeit zur Extrahierung eines Teilstrings ist die Methode `substr()`. Diese gibt die Zeichen in einem String zurück, wobei bei dem als ersten Parameter angegebenen Index begonnen wird und bis Ende oder der optionalen Anzahl von Zeichen extrahiert wird. Eine negative Angabe ist nicht möglich.

Achtung

Beachten Sie die verschiedenen Funktionen des zweiten Parameters bei `slice()` und `substr()`. Bei `slice()` ist es eine 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(
09   "Von 58 bis Ende: " + text.substr(58) + "<br>");
10 </script>
11 </body>
12 </html>
```

Listing 430: Extraktion von Teilstrings mit `substr()`

In Zeile 5 wird wieder eine Variable mit einem String eingeführt. In Zeile 7 extrahieren wir von einem vorgegebenen Anfangspunkt 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 271: Verschiedene Extrakte mit `substr()`

155 Wie erfolgt die Extrahierung mit `substring()`?

Eine dritte Möglichkeit der `String`-Klasse zur Extrahierung eines Teilstring ist die Methode `substring()`. Diese gibt die Zeichen in einem String zurück, wobei bei dem als ersten Parameter angegebenen Index begonnen wird und bis zu Ende oder optionalen Endindex extrahiert wird. Eine negative Angabe ist nicht möglich.

Achtung

Beachten Sie die verschiedenen 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>");
08 </script>
09 </body>
10 </html>
```

Listing 431: Extraktion von Teilstrings mit substring()

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.

156 Wie kann ich einen String an einem 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 Teilsegmente erfolgen soll, und trennt einen vorangestellten String in ein Datenfeld mit Teilstrings, 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.

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(",");
06   for(i = 0; i < adr.length; i++){
07     document.write(adr[i] + "<br>");
08   }
09 </script>
10 </body>
11 </html>
```

Listing 432: Auftrennen eines Strings mit `split()`

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 von dem Datenfeld in einer `for`-Schleife ausgeben.

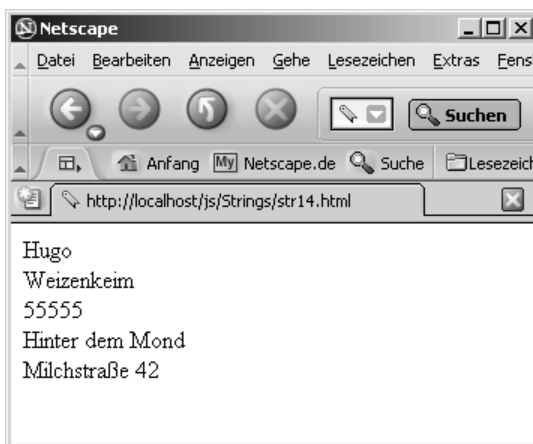


Abbildung 272: Die Ausgabe von dem zerlegten String – das Komma ist in keinem der Datenfeldelemente mehr enthalten.

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 nichtdruckbaren 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.

Beachten Sie dazu das Rezept »Wie kann ich mit regulären Ausdrücken in Strings suchen und sie ersetzen?« auf Seite 560.

Beispiel (*str14a.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   text = "Hugo\tWeizenkeim\t5555\tHinter dem Mond\tMilchstraße 42";
05   adr = text.split(/\t/);
06   for(i = 0; i < adr.length; i++){
07     document.write(adr[i] + "<br>");
08   }
09 </script>
10 </body>
11 </html>
```

Listing 433: Auftrennen eines Strings mit split() unter Verwendung eines regulären Ausdrucks

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

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 das Objekt `Date` für Aktionen rund um Datum und Zeit zur Verfügung. Über `Date` stehen Ihnen dazu über die reine Erzeugung von Datumsobjekten hinaus zahlreiche Methoden bereit. Der 1. Januar 1970, 0.00, Uhr ist dabei der interne Zeitnullpunkt, der als Speicherungs- und Berechnungsbasis für alle Operationen mit der Zeit und dem Datum dient. Die Zeit wird in verstrichenen Millisekunden seit diesem Zeitpunkt verwaltet.

157 Wie kann ich ein Datumsobjekt mit dem aktuellen Systemdatum erzeugen?

Die einfachste Variante zur Erzeugung eines Datumsobjekts verwendet den Konstruktor `Date()` ohne Parameter. Mit diesem wird ein Objekt mit dem aktuellen Systemdatum als enthaltenen Wert erzeugt:

Die Syntax sieht so aus:

```
new Date();
```

Listing 434: 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 435: 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 bitte Achtung. Die Geschichte ist keinesfalls so trivial, wie die primitive Syntax des aktuellen Beispiels vielleicht glauben macht. Mit `document.write()` schreiben wir direkt ein Objekt 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 Semikolon? Oder der Inhalt von nur einem Datenfeldelement? Und wenn ja, welchem denn? Und wenn es sich hier um ein Objekt vom fiktiven Typ *Schwein*¹ handeln würde – was wäre die Ausgabe? Etwa *Grunz*? Es ist keinesfalls trivial, dass die Ausgabe von `new Date()` einen sinnvollen Text in die 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.



Abbildung 273: Die unformatierte Ausgabe des Systemdatums – hier im Netscape Navigator 7.1

1. Sie können ja in JavaScript auch eigene Objekte erzeugen.

Hinweis

Beachten Sie, dass sich die unformatierte Ausgabe von dem `Date`-Objekt je nach Webbrowser und Einstellung des Browsers unterscheiden kann.

Datum: Fri Nov 05 12:45:59 GMT+0100 (Westeuropäische Normalzeit) 2004

Abbildung 274: Die unformatierte Ausgabe des Systemdatums – hier im Netscape Navigator 4.7 mit englisch-deutscher Sprachmischung

Allgemein ist die unformatierte Ausgabe des Werts im `Date`-Objekt wenig für die direkte Ausgabe im Webbrowser geeignet.

158 Wie kann ich ein Datumsobjekt mit einem vorgegebenen Datum erzeugen?

Es gibt verschiedene Varianten des `Date`-Konstruktor zur Erzeugung eines Datumsobjekts mit einem vorgegebenen Datum.

Variante 1

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 436: 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.

Variante 2

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 437: 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 438: Erzeugung eines Datumsobjekts mit dem vorgegebenen Datum in Form von sieben Parametern

```
new Date(Jahr, Monat, Tag, Stunde, Minute, Sekunde);
```

Listing 439: 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(2004, 10, 5, 12, 45, 59, 4);
05   b = new Date(2004, 10, 5, 12, 45, 59);
06   c = new Date(2004, 10, 5, 12, 45, 59);
07   document.write(new Date("october 12, 2004 12:11:03") + "<br>");
08   document.write(new Date(2004, 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 440: Erzeugung eines Datumsobjekts mit einigen vorgegebenen Datumsangaben und Ausgabe der Werte in der Webseite

In Zeile 4 erzeugen wir ein Datumsobjekt *a* mit Hilfe des Konstruktors mit sieben numerischen Parametern und in Zeile 5 ein Datumsobjekt *b* mit dem Konstruktor 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 abziehen¹, zeigt aber, dass sie sich sehr wohl unterscheiden. In der Zeile 7 kommt der Konstruktor mit dem Stringparameter zum Einsatz und in Zeile 8 der Konstruktor mit drei numerischen Parametern.

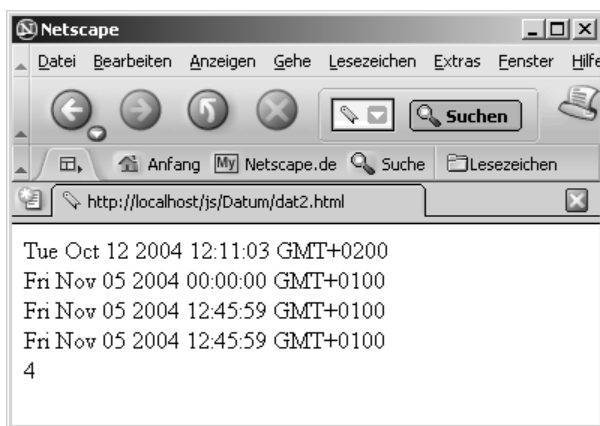


Abbildung 275: Verschiedene Datumsangaben sowie die Differenz von zwei Datumsobjekten

1. 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.

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 nächst größeren Bereich fortgesetzt werden. Geben Sie etwa für die Sekunden den Wert 90 an, wird der Wert der Minute um 1 erhöht, und die Sekunden werden auf den Wert 30 gesetzt.

Beispiel (*dat2a.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   a = new Date(2004, 10, 5, 12, 75, 0);
05   b = new Date(2004, 10, 5, 12, 45, 90);
06   c = new Date(2004, 10, 40);
07   document.write("new Date(2004, 10, 5, 12, 75, 0): " + a + "<br>");
08   document.write("new Date(2004, 10, 5, 12, 45, 90): " + b + "<br>");
09   document.write("new Date(2004, 10, 40): " + c + "<br>");
10 </script>
11 </body>
12 </html>
```

Listing 441: Im ersten Moment unsinnig erscheinende Verwendungen in den Parametern des Date()-Konstruktors

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(2004, 10, 5, 12, 75, 0): Fri Nov 5 13:15:00 UTC+0100 2004

Abbildung 276: 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(2004, 10, 5, 12, 45, 90): Fri Nov 5 12:46:30 UTC+0100 2004

Abbildung 277: 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(2004, 10, 40): Fri Dec 10 00:00:00 UTC+0100 2004
```

Abbildung 278: Die Monatstage sprengen den Bereich

Der Grund für diesen doch sehr sinnvollen Umgang mit der Überschreitung von Wertebereichen 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.

159 Wie kann ich den Tag des Monats aus einem Datumsobjekt extrahieren?

Das `Date`-Objekt 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">
04   a = new Date(2004, 10, 5, 12, 45, 59, 4);
05   document.write(
06     "Datum: " + a + "<br>Tag des Monats: " + a.getDate() + "<br>");
07 </script>
08 </body>
09 </html>
```

Listing 442: Extraktion des Monatstags

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 6 der Tag des Monats unter Verwendung der Methode `getDate()` ausgegeben.

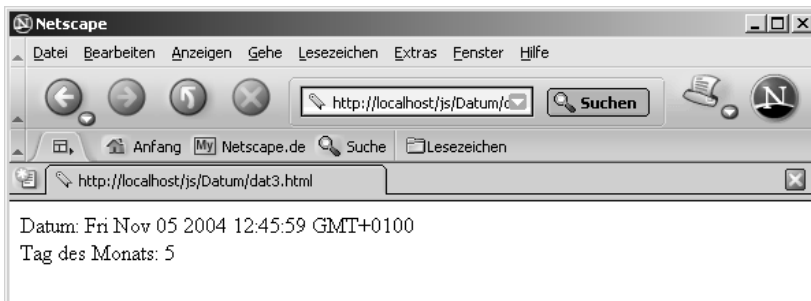


Abbildung 279: Extrahierung des Monatstags

160 Wie kann ich in einem existierenden Datumsobjekt den Monatstag ändern?

Das `Date`-Objekt 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(2004, 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 443: 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.

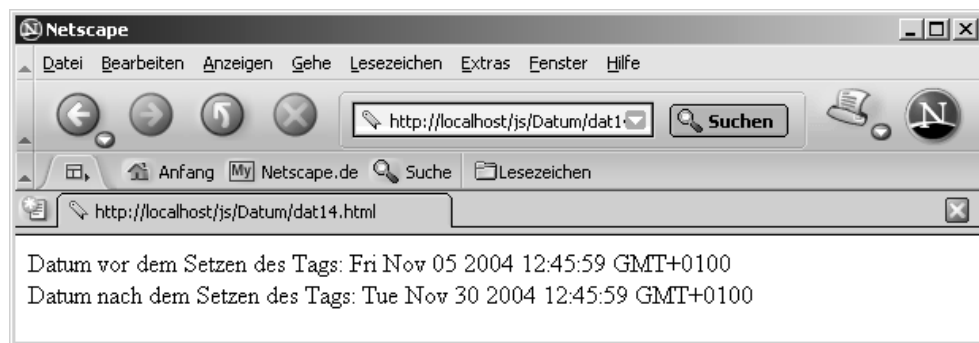


Abbildung 280: Der ursprüngliche Tag im Datumsobjekt, gefolgt von dem geänderten Tag

Achtung

Die erlaubten Monatstage als Parameter 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 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 fortgesetzt 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. 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.

161 Wie kann ich den Wochentag des Objekts aus einem Datumsobjekt extrahieren?

Das `Date`-Objekt 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(2004, 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 444: Extraktion des Monatstags

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 6 der Tag der Woche über die Methode `getDay()` als Zahl ausgegeben.

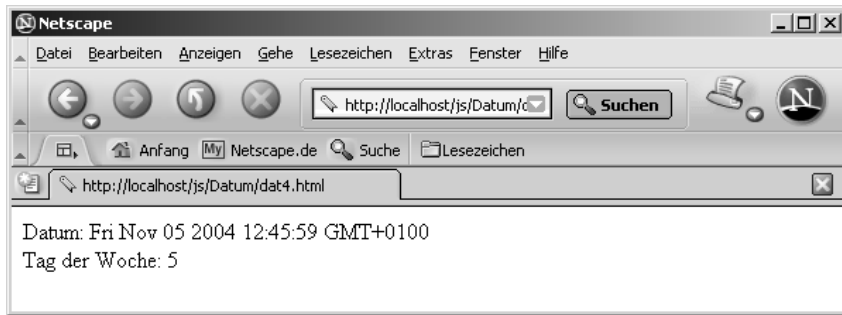


Abbildung 281: Extrahierung des Monatstags

Natürlich ist eine rein numerische Ausgabe eines Wochentags nicht sonderlich komfortabel, und vor allem ist die Zählweise mit dem Wert 0 für Sonntag etwas ungewohnt. JavaScript stellt keine direkte Möglichkeit bereit, die deutschen Wochentagsnamen aus einem Datumsobjekt zu extrahieren². 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 445: 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 keine `break`-Anweisung hier benötigen, denn `return` verlässt bereits das `switch-case`-Konstrukt, und die nachfolgenden Anweisungen werden damit nicht erreicht.

- 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 in eine Zeichenkette mit lokaler Darstellung umwandeln?« auf Seite 628).

Schauen wir uns die Funktion in einem vollständigen Beispiel an.

Beispiel (*dat4a.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
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(2004, 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 446: Die Ausgabe von dem Wochentag in ausgeschriebener Form

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.



Abbildung 282: Der ausgeschriebene Wochentag

162 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 repräsentieren. Welchen Wert für die Millisekunden sollte ein Setzen auf einen Wochentag wie Montag denn ergeben? Ein Wochentag ist im Gegensatz zu den anderen Zeitdaten wie einem Monat, einem Jahr oder einem Tag des Monats keine eindeutig zu bestimmende Zeitspanne ab einem Zeitnullpunkt. 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.

163 Wie kann ich die Stunden aus einem Datumsobjekt extrahieren?

Das `Date`-Objekt 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(2004, 10, 5, 12, 45, 59, 4);
05   document.write(
06     "Datum: " + a + "<br>Aktuelle Stunde: " + a.getHours() + "<br>");
07 </script>
08 </body>
09 </html>
```

Listing 447: Extraktion der Stunde

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 6 die extrahierte Stunde mit der Methode `getHours()` ausgegeben.

Grund-
lagen

Core

Formu-
lare

Aus-
nahme

Arrays

Strings

Datum

Platt-
form

DHTML

Events

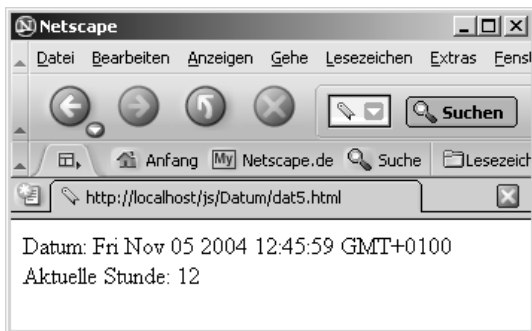


Abbildung 283: Die Extraktion der Stunde aus einem Datumsobjekt

164 Wie kann ich in einem existierenden Datumsobjekt die Stunden ändern?

Das Date-Objekt stellt die Methode `setHours()` bereit, um darüber die in einem vorangestellten Datumsobjekt gespeicherte Stunde auf die als Zahl zu übergebenden 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 in den nächsten Tagen fortgesetzt werden. Diese Anweisung setzt das Datum einfach auf 16 Uhr des Folgetags. Sie können sogar negative Werte als Parameter verwenden. 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.

Beispiel (*dat15.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 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 448: Das Setzen des Montagstags

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.

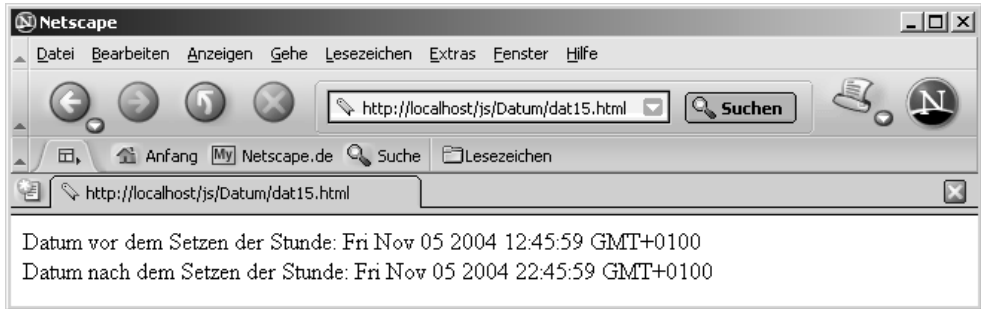


Abbildung 284: Die ursprüngliche Stunde im Datumsobjekt, gefolgt von der geänderten Stunde

165 Wie kann ich die Minuten der Uhrzeit aus einem Datumsobjekt extrahieren?

Das Date-Objekt 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(2004, 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 449: Extraktion der Minuten

In Zeile 4 wird ein Datumsobjekt erzeugt, und in Zeile 6 werden die mit `getMinutes()` extrahierten Minuten ausgegeben.

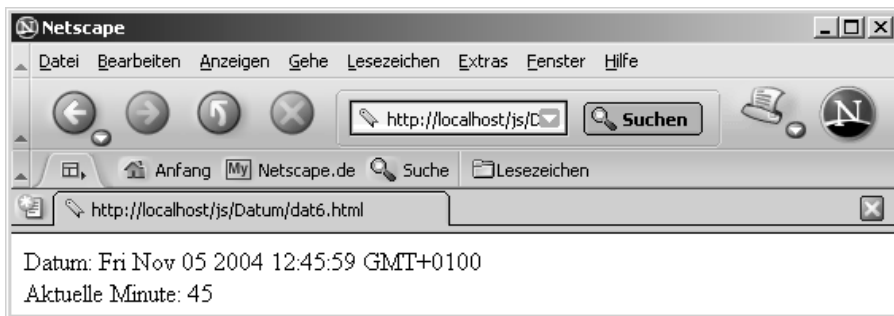


Abbildung 285: Extraktion der Minuten aus dem Datumsobjekt

166 Wie kann ich in einem existierenden Datumsobjekt die Minuten ändern?

Das `Date`-Objekt 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 fortgesetzt werden. Diese Anweisung setzt das Datum einfach auf 5 Minuten in der Folgestunde. 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.

Beispiel (*dat16.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 der Minuten: " + a + "<br>");
06   a.setMinutes(42);
07   document.write("Datum nach dem Setzen der Minuten: " + a + "<br>");
08 </script>
09 </body>
10 </html>
```

Listing 450: 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.

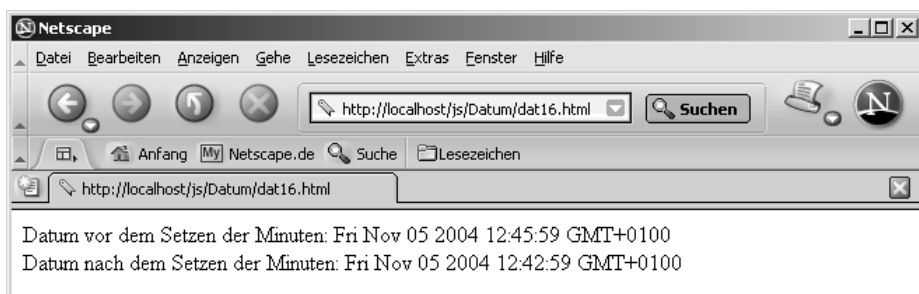


Abbildung 286: Die ursprünglich enthaltenen Minuten im Datumsobjekt, gefolgt von den geänderten Minuten

167 Wie kann ich die Sekunden der Uhrzeit aus einem Datumsobjekt extrahieren?

Das Date-Objekt 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(2004, 10, 5, 12, 45, 59, 4);
05   document.write(
06     "Datum: " + a + "<br>Aktuelle Sekunde: " + a.getSeconds() + "<br>");
07 </script>
08 </body>
09 </html>
```

Listing 451: Extrahierung der Sekunden

In Zeile 4 wird ein Datumsobjekt erzeugt, und in Zeile 6 werden die mit `getSeconds()` extrahierten Sekunden ausgegeben.

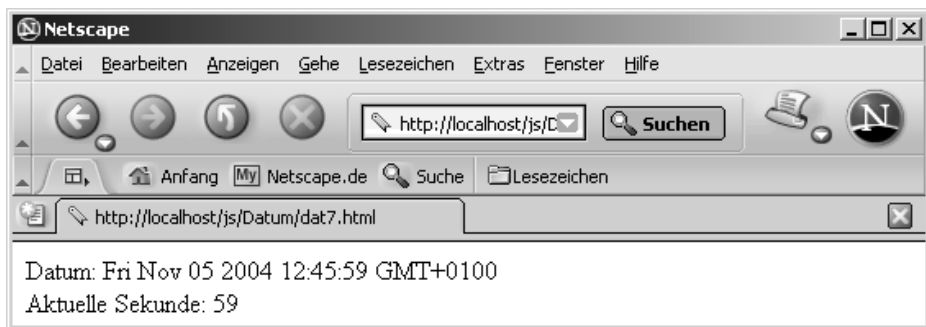


Abbildung 287: Extraktion der Sekunden

168 Wie kann ich in einem existierenden Datumsobjekt die Sekunden ändern?

Das Date-Objekt 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 fortgesetzt werden. Diese Anweisung setzt das Datum einfach auf 5 Sekunden in der Folgeminute. 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.

Beispiel (*dat18.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 der Sekunden: " + a + "<br>");
06   a.setSeconds(1);
07   document.write("Datum nach dem Setzen der Sekunden: " + a + "<br>");
08 </script>
09 </body>
10 </html>
```

Listing 452: 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.

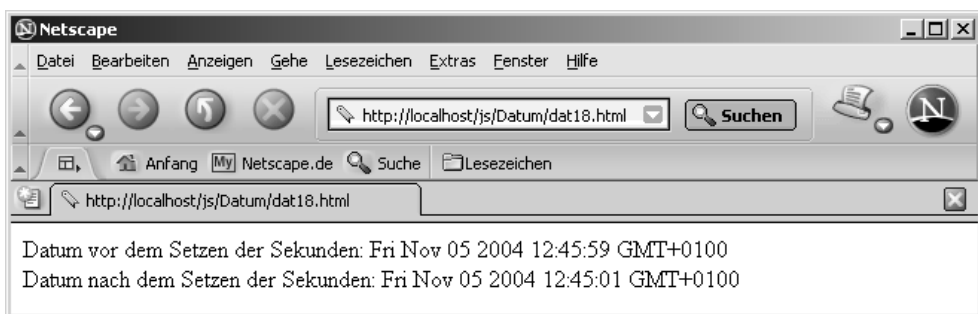


Abbildung 288: Die veränderten Sekunden

169 Wie kann ich die Jahreszahl aus einem Datumsobjekt extrahieren?

Das `Date`-Objekt stellt die Methode `getFullYear()` 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(2004, 10, 5, 12, 45, 59, 4);
05   document.write(
06     "Datum: " + a + "<br>Aktuelles Jahr: " + a.getFullYear() + "<br>");
07 </script>
08 </body>
09 </html>
```

Listing 453: Extraktion des Jahres

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 6 das Jahr ausgegeben, wie es mit `getFullYear()` extrahiert wurde.

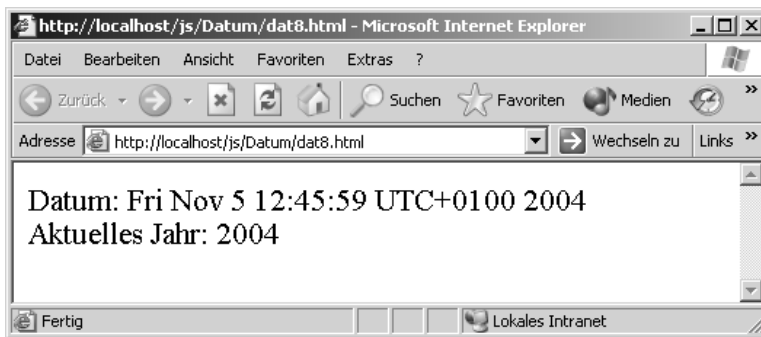


Abbildung 289: Extraktion des Jahres – hier die Anzeige im Internet Explorer

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.

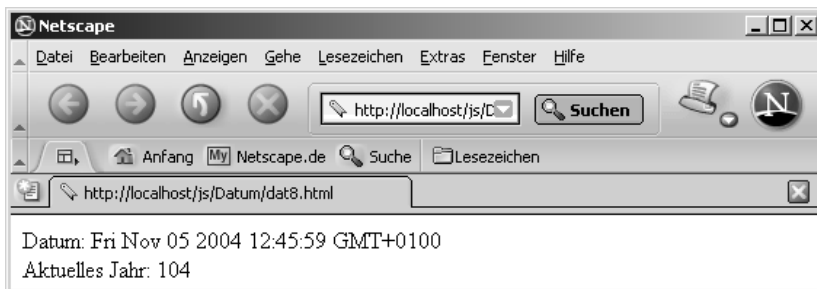


Abbildung 290: Extraktion des Jahres – hier der Netscape Navigator 7.1

Das Jahr wird bei fast allen Browsern ausgehend von dem Jahr 1900 dargestellt. Also für das Jahr 2004 erhalten Sie den Wert 104. Im Internet Explorer hingegen erhalten Sie den Wert 2004.

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 `getFullYear()` konsequenterweise negative Werte für das Jahr. Sie können das Phänomen mit der Datei *dat8b.html* testen.



Abbildung 291: Extrahierung des Jahres bei einem Datum von dem Jahr 1900 – hier der Netscape Navigator 4.7

Hinweis

Auf den ersten Blick scheint das Verhalten im Internet Explorer, bei `getFullYear()` 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 `getFullYear()` einige »interessante« 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.getFullYear() + "<br>");
07   document.write(b.getFullYear() + "<br>");
08 </script>
09 </body>
10 </html>
```

Listing 454: 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?

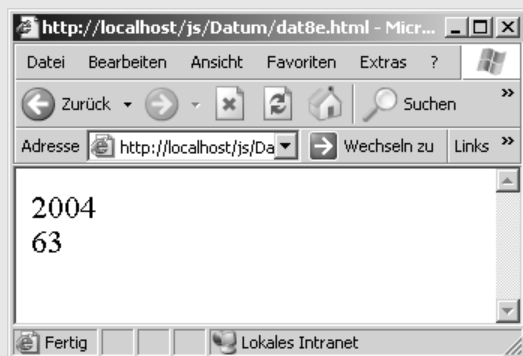


Abbildung 292: Bei Jahren ab 2000 liefert der Internet Explorer die vollständige Jahresangabe, davor den Wert nur zweistellig.

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 den Wert nur zweistellig. Dieses inkonsistente Verhalten erscheint doch mehr als unlogisch.

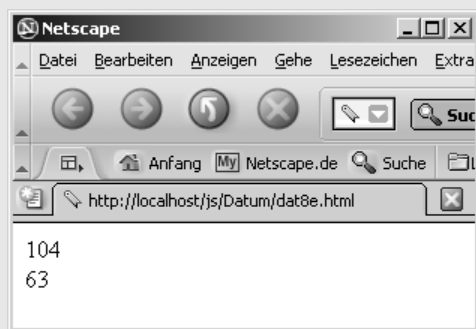


Abbildung 293: Dann schon lieber einheitlich wie hier im Navigator

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 635 und die Warnung dort).

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 455: 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 2004 sowohl im Internet Explorer als auch den anderen Browsern den Wert 4. Dieser Wert wird auf den konstanten Wert 2000 addiert. Damit wird die Darstellung in allen Browsern identisch 2004 sein. Sie können die Funktion mit der Datei `dat8a.html` testen.

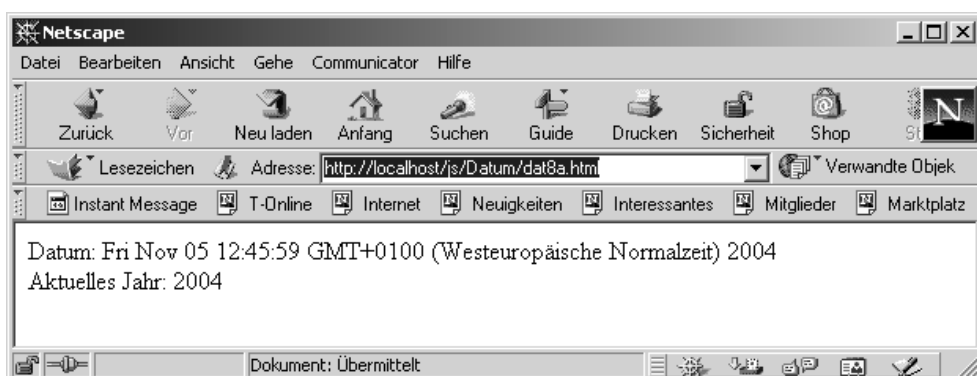


Abbildung 294: Extraktion des Jahres – hier der Netscape Navigator 4.7

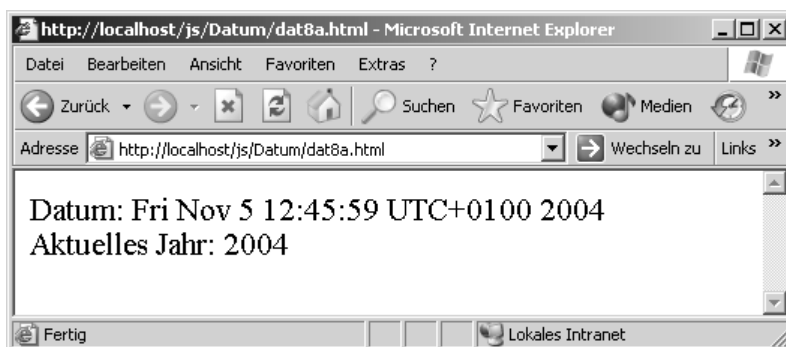


Abbildung 295: Extraktion des Jahres – hier der Internet Explorer

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 beziehungs-

weise 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.getYear()) % 1900);  
03 }
```

Listing 456: Eine Funktion zur gleichen Darstellung des Jahres ab 1900 in allen Browsern

Die Funktion bekommt wieder als Übergabewert ein Datumsobjekt und extrahiert erneut das Jahr mit `getYear()`. Allerdings wird der Wert hier nicht modulo dem Wert 100 genommen. Stattdessen wird auf den Wert von `getYear()` 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 von dem Jahr 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 aufgrund der konstanten Addition des Werts 1900). Falls Sie einen Wert vor diesem Zeitpunkt darstellen wollen (warum auch immer), macht eine Browserweiche viel Sinn und dann 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 457: 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 Browserweiche³, ob der Browser des Anwenders der Internet Explorer ist oder nicht. Dazu durchsuchen wir mit Hilfe

-
3. Natürlich können hier auch komplexere Browserweichen zum Einsatz kommen, wenn das notwendig erscheint.

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.

170 Wie kann ich in einem existierenden Datumsobjekt die Jahreszahl ändern?

Das `Date`-Objekt stellt die Methode `setYear()` bereit, um darüber das in einem vorangestellten Datumsobjekt gespeicherte Jahr auf den als Zahl zu übergebenden Monatstag zu ändern. Das Jahr ist vierstellig⁴ anzugeben. Das gilt auch für die Browser, bei denen die Abfrage des Jahres einen Wert vom Startjahr `1900` aus gerechnet ergibt (siehe das Rezept »Wie kann ich die Jahreszahl aus einem Datumsobjekt extrahieren?« auf Seite 612).

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 Jahrs: " + a + "<br>");
06   a.setYear(2005);
07   document.write("Datum nach dem Setzen des Jahrs: " + a + "<br>");
08 </script>
09 </body>
10 </html>

```

Listing 458: Das Setzen des Jahres

4. Oder genauer – mit dem exakten Wert (falls Sie ein Jahr vor dem Jahr `1000` oder nach `9999` setzen wollen).

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.

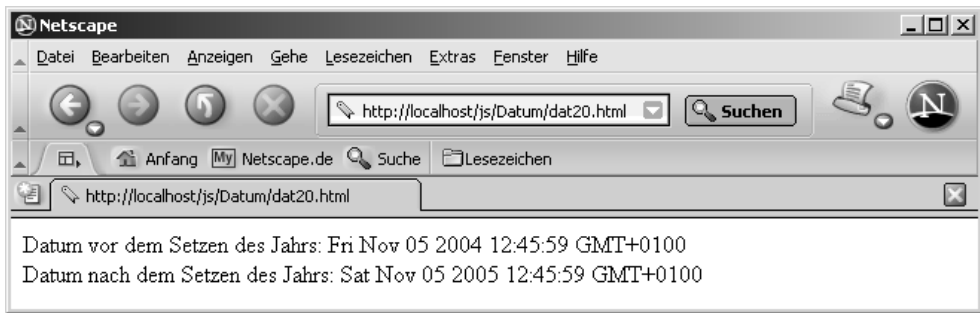


Abbildung 296: Die Veränderung von dem Jahr

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 Browsern wie dem Netscape Navigator oder Opera das Jahr über die tatsächliche Jahreszahl.

171 Wie kann ich den Monat aus einem Datumsobjekt extrahieren?

Das `Date`-Objekt stellt die Methode `getMonth()` bereit, um darüber den Monat als Zahl daraus zu extrahieren.

Beispiel (*dat9.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
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 459: Extrahierung des Monatstags

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 6 der mit `getMonth()` extrahierte Monat ausgegeben.

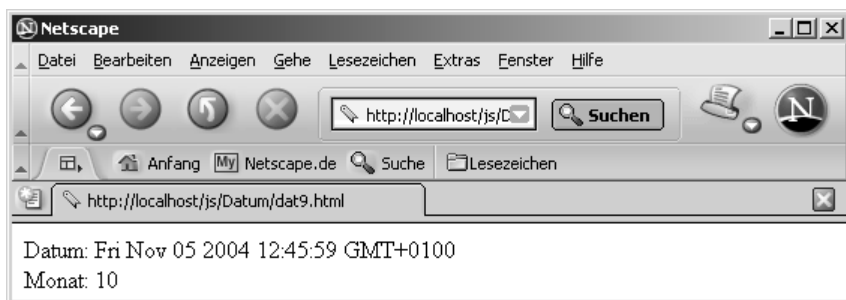


Abbildung 297: Extraktion des Monats

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 über die ungewöhnliche Darstellung der Monatszahl mit Beginn 0 hinaus eine rein numerische Ausgabe eines Monats nicht sonderlich komfortabel. JavaScript stellt keine direkte Möglichkeit bereit, die deutschen Monatsnamen aus einem Datumsobjekt zu extrahieren⁵. 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";
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     default : return "Dezember";
16   }
17 }
```

Listing 460: Rückgabe des Monats in deutscher Schreibweise

5. 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 in eine Zeichenkette mit lokaler Darstellung umwandeln?« auf Seite 628).

Der Funktion wird beim Aufruf ein Datumsobjekt übergeben. In Zeile 2 wird der Monat aus dem übergebenen Datumsobjekt mit `getMonth()` numerisch extrahiert. Das nachfolgende `switch-case`-Konstrukt gibt als String den jeweils passenden Monat in ausgeschriebener Version zurück.

Hinweis

Beachten Sie, dass Sie trotz des Fall-Through-Charakters des `switch-case`-Konstrukts keine `break`-Anweisung hier 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         default : return "Dezember";
19     }
20 }
21 a = new Date(1963, 4, 17);
22 document.write("Datum: " + a + "<br>Monat: " + monat(a) + "<br>");
23 </script>
24 </body>
25 </html>
```

Listing 461: Die Ausgabe von dem Monat in ausgeschriebener Form

In Zeile 21 wird ein Datumsobjekt erzeugt und in Zeile 22 der Monat in ausgeschriebener Form ausgegeben, da der Rückgabewert der Funktion `monat()` verwendet wird.

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

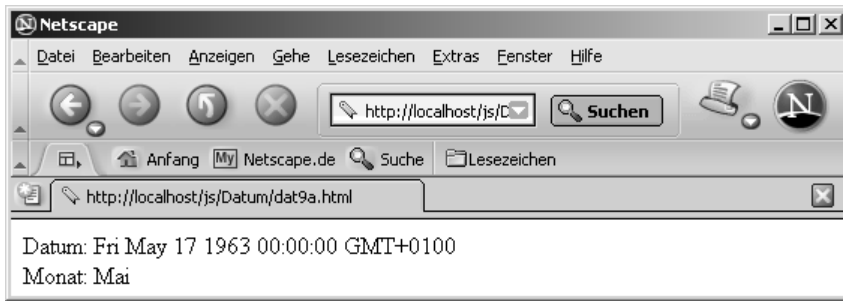


Abbildung 298: Darstellung des extrahierten Monats in ausgeschriebener Form

172 Wie kann ich in einem existierenden Datumsobjekt den Monat ändern?

Das Date-Objekt stellt die Methode `setMonth()` bereit, um darüber den in einem vorangestellten Datumsobjekt gespeicherten Monat auf den als Zahl zu übergebenden Monat zu ändern. Es sind als Werte für den Monat 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 fortgesetzt werden. Diese Anweisung setzt das Datum nur auf den Januar in dem Folgejahr. 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.

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);
07   document.write("Datum nach dem Setzen des Monats: " + a + "<br>");
08 </script>
09 </body>
10 </html>
```

Listing 462: Das Setzen des Monatstags

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.

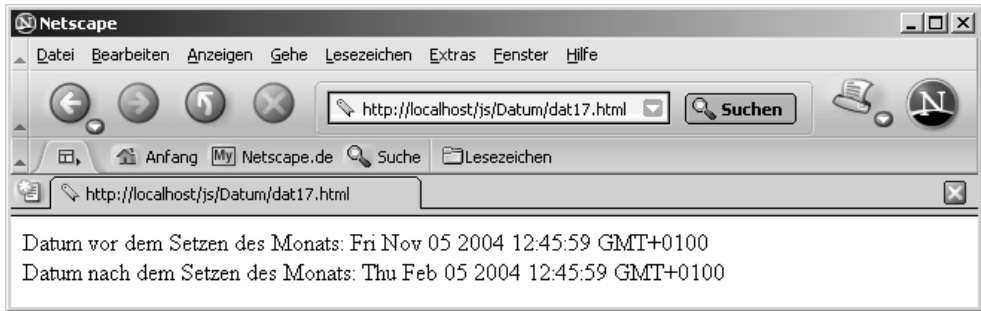


Abbildung 299: Die ursprünglichen Minuten im Datumsobjekt, gefolgt von den geänderten Minuten

173 Wie kann ich die Zeit, die seit dem 1.1.1970, 0:00:00 vergangen ist, extrahieren?

Sämtliche Datums- und Zeitangaben in JavaScript berechnen sich in Millisekunden, die seit dem 1. Januar 1970, 0:00:00 Uhr, vergangen sind. Das `Date`-Objekt 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 463: Extrahierung des Monatstags

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 7 die Anzahl der Millisekunden ausgegeben, wie sie von `getTime()` geliefert werden.

Interessant ist es natürlich, welche Werte die Methode `getTime()` liefert, wenn ein Datumsobjekt mit einem Wort vor dem 1. Januar 1970, 0:00:00, die Basis bildet. Es werden konsequenterweise einfach negative Werte extrahiert.

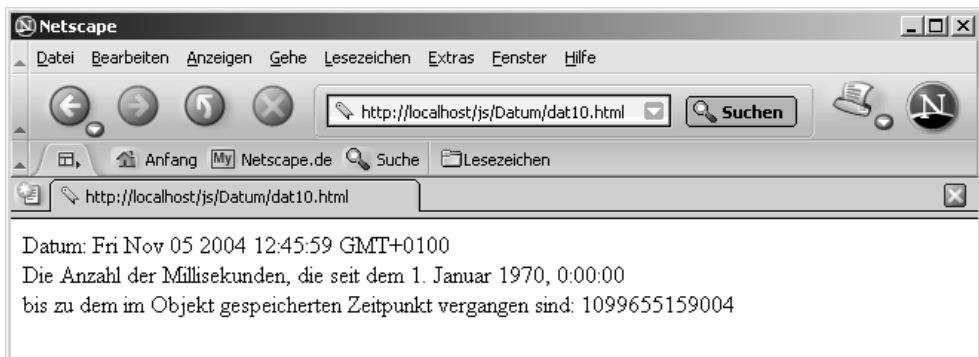


Abbildung 300: Die Anzahl der Millisekunden seit dem 1. Januar 1970, 0:00:00

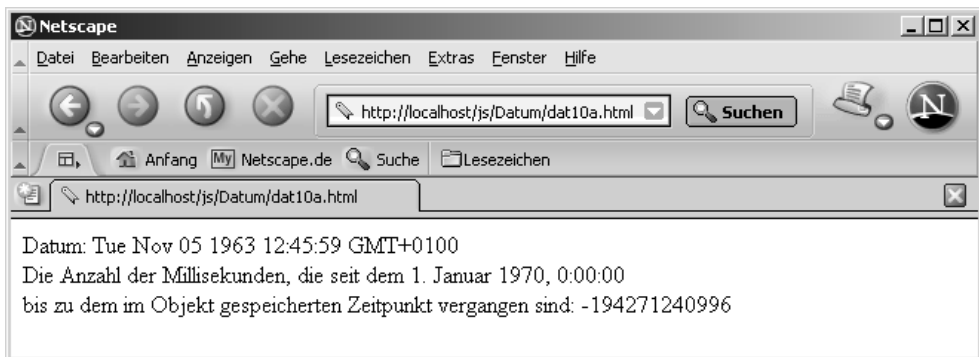


Abbildung 301: Die Anzahl der Millisekunden seit dem 1. Januar 1970, 0:00:00, bei einem Datumsobjekt, das vor diesem Zeitpunkt liegt

Tipp

Es gibt noch zwei Alternativen, um die Anzahl 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 übergebenden Zeit in Form einer Zeichenkette nach dem IETF-Standard (Internet Engineering Task Force) diesen Wert 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 in den IETF-Standard umwandeln?« auf Seite 627.*

Die zweite Alternative ist die Methode `UTC()`. Dabei wird die folgende Syntax verwendet:

```
UTC(Jahr, Monat, Tag [, Stunden] [, Minuten] [, Sekunden])
```

Listing 464: 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.

174 Wie kann ich in einem existierenden Datumsobjekt die Millisekunden ändern?

Das Date-Objekt stellt die Methode `setTime()` bereit, um darüber die in einem vorangestellten Datumsobjekt gespeicherten Millisekunden auf den als Zahl zu übergebenden Monatstag zu ändern. Diese Methode ändert den kompletten Inhalt von dem Objekt 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">
04   a = new Date(2004, 10, 5, 12, 45, 59, 4);
05   document.write("Datum vor dem Setzen der Millisekunden: " + a + "<br>");
06   a.setTime(a.getTime() + 10000);
07   document.write("Datum nach dem Setzen der Millisekunden: " + a + "<br>");
08 </script>
09 </body>
10 </html>
```

Listing 465: Das Setzen der Millisekunden seit dem 01.01.1970 – 0:00:00 Uhr

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 `setMinutes()` geändert. Dabei fragen wir zuerst die Anzahl der Millisekunden in dem bereits erzeugten Datumsobjekt mit der Methode `getTime()` ab⁶ und addieren darauf einfach einen Wert. In Zeile 7 wird das veränderte Datum ausgegeben.

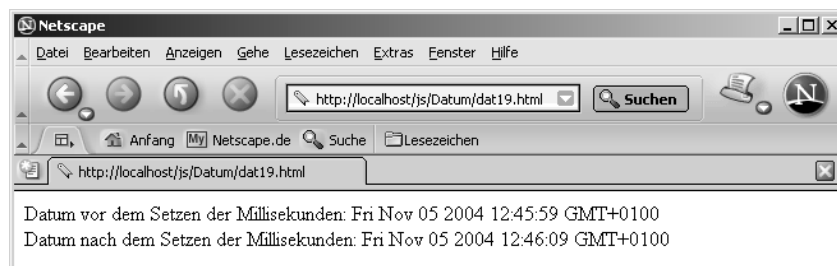


Abbildung 302: Veränderung der Millisekunden

6. Der Grund ist, dass die Zahl der Millisekunden bei einem aktuellen Datum recht groß ist und man ohne Hilfsmittel beim Setzen eines Werts kaum auskommt. Oder wissen Sie aus dem Stehgreif, wie viele Millisekunden seit dem 1. Januar 1970, 0.00 Uhr, bis heute vergangen sind :-).

175 Wie kann ich den Unterschied zwischen der lokalen Zeit und Greenwich Meridian Time ermitteln?

Zeitangaben unterscheiden sich je nach 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 Längengreis. 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. Das `Date`-Objekt stellt die Methode `getTimezoneOffset()` bereit, um darüber den Unterschied zwischen der lokalen Zeit, wie sie in dem Rechner des Anwenders eingestellt ist, und Greenwich Meridian 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 +
06     "<br>Unterschied zwischen der lokalen Zeit und Greenwich Meridian Time: "
07     + a.getTimezoneOffset() + "<br>");
08 </script>
09 </body>
10 </html>
```

Listing 466: Extrahierung des Montagstags

In Zeile 4 wird ein `Datums`objekt erzeugt und in Zeile 7 der Unterschied zwischen der lokalen Zeit und Greenwich Meridian Time in Minuten ausgegeben. Deutschland ist in der Sommerzeit zwei Stunden und in der Winterzeit eine Stunde hinter der GMT.

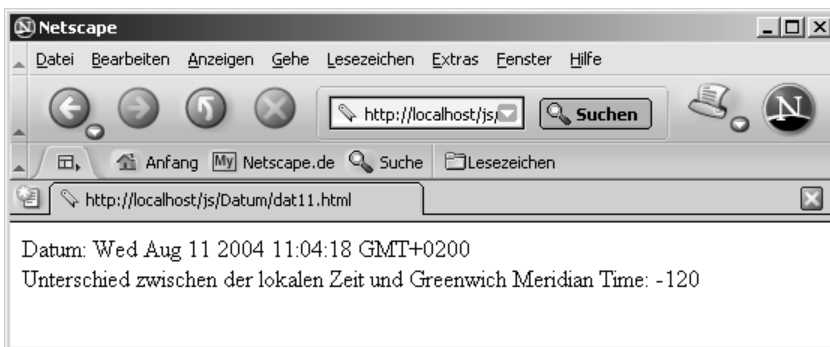


Abbildung 303: Deutschland im Sommer



Abbildung 304: Deutschland – ein Wintermärchen

176 Wie kann ich eine Zeit 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), der im deutschsprachigen Raum kaum üblich und ziemlich unbequem ist. Das `Date`-Objekt 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 IETF-Format: "
06     + a.toGMTString() + "<br>");
07 </script>
08 </body>
09 </html>

```

Listing 467: Umwandlung in das IETF-Format

In Zeile 4 wird ein Datumsobjekt erzeugt und in Zeile 5 der Datumsstring nach dem IETF-Format ausgegeben.

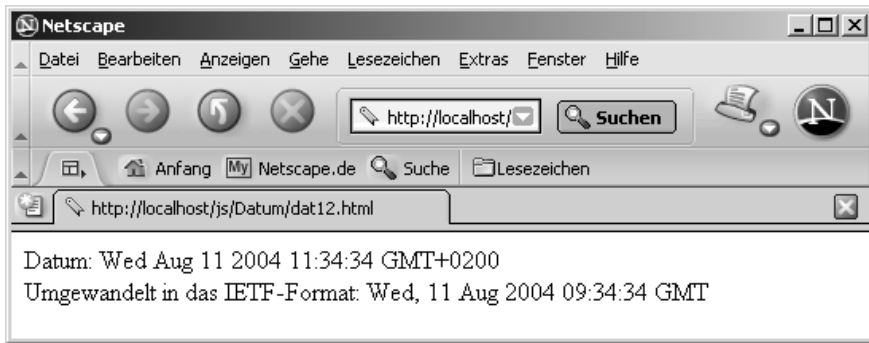


Abbildung 305: Ein Datumsstring im IETF-Format

Beachten Sie auch das folgende, verwandte Rezept »Wie kann ich eine Zeit in eine Zeichenkette mit lokaler Darstellung umwandeln?« auf Seite 628.

177 Wie kann ich eine Zeit in eine Zeichenkette mit lokaler Darstellung umwandeln?

Der Umgang mit Datum und Uhrzeit ist in JavaScript sehr stark am angloamerikanischen Sprachgebrauch als auch der entsprechenden Darstellung von Zahlen und Datums- beziehungsweise Zeitformaten orientiert. Das `Date`-Objekt 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 des Zeit und des Datums (entsprechend der Einstellung auf dem Computer) berücksichtigt.

Hinweis

Beachten Sie auch das verwandte Rezept »Wie kann ich eine Zeit in den IETF-Standard umwandeln?« auf Seite 627.

Beispiel (*dat13.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   a = new Date();
05   document.write("Datum: " + a +
06   "<br>Umgewandelt in einen String mit einem lokalen Format: " +
07   a.toLocaleString() + "<br>");
08 </script>
```

Listing 468: Umwandlung in das IETF-Format


```
09 </body>
10 </html>
```

Listing 468: Umwandlung in das IETF-Format (Forts.)

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 306: Ein Datumsstring mit lokaler Darstellung – hier im Internet Explorer

So weit erscheint – wie so oft – alles in Ordnung und ganz einfach. Und – wie ebenfalls so oft – ist das ein Trugschluss. Laden Sie das Beispiel einfach in verschiedene Browser. Sie werden recht interessante Unterschiede zwischen den Browsern⁷ entdecken, die eine Verwendung der Methode nur eingeschränkt sinnvoll machen.

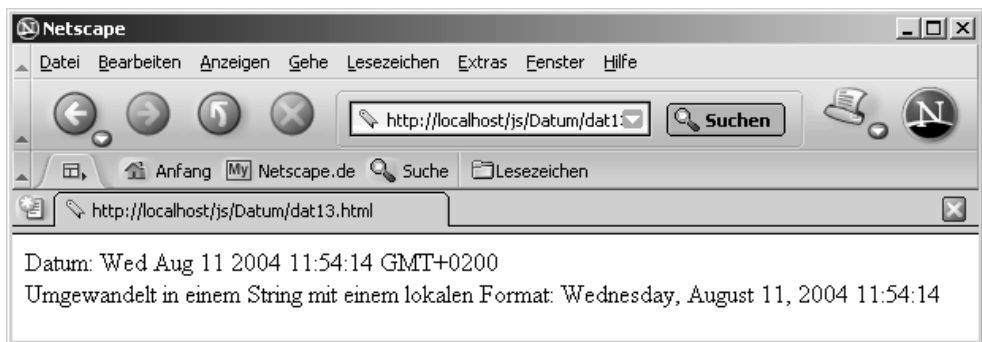


Abbildung 307: Ein lokal angepasster Datumsstring im Netscape Navigator mit deutscher Einstellung – seit wann schreibt man den Mittwoch in Deutsch als Wednesday?

7. Konkret hängt die Darstellung an den Einstellungen des jeweiligen Browsers respektive der gesamten Plattform des Anwenders.

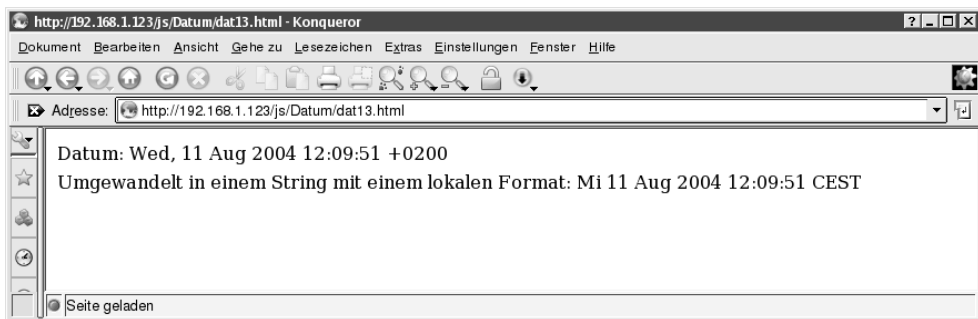


Abbildung 308: Ein lokal angepasster Datumsstring im Konqueror mit deutscher Einstellung – die Darstellung ist zwar in Deutsch, aber abgekürzt.

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 469: 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()` im Internet Explorer geliefert wird. Dabei greifen wir auf die Standardmethoden von dem `Date`-Objekt zur Extrahierung der einzelnen Daten wie Stunden und Minuten zurück, soweit sie unpro-

blematisch 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 des Objekts aus einem Datumsobjekt extrahieren?« auf Seite 603, »Wie kann ich den Monat aus einem Datumsobjekt extrahieren?« auf Seite 618 und »Wie kann ich das Jahr aus einem Datumsobjekt extrahieren?« auf Seite 608 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">
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() + ":" +
```

Listing 470: Die angepasste Darstellung eines Datumsstrings

```

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 43 </script>
46 </body>
47 </html>

```

Listing 470: Die angepasste Darstellung eines Datumsstrings (Forts.)

In Zeile 42 wird ein Datumsobjekt erzeugt und in Zeile 44 der Datumsstring in einer Darstellung ausgegeben, der die lokale Darstellung eines Datums berücksichtigt. Dabei kommt die Funktion `lokaleZeitDarstellung()` wie besprochen zum Einsatz.

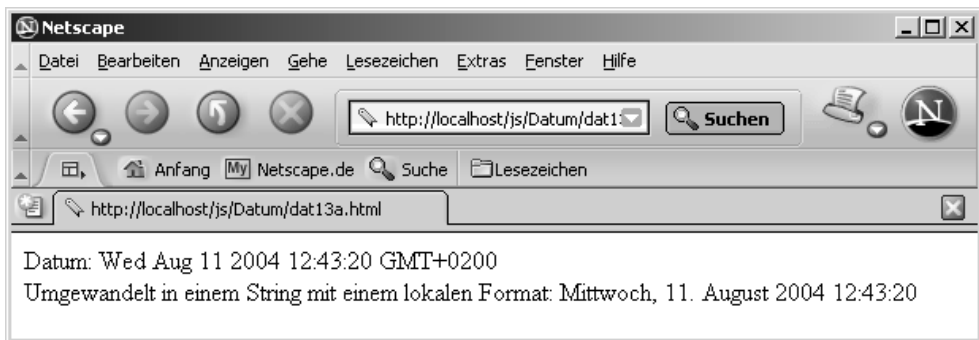


Abbildung 309: Dann klappt es auch mit dem Navigator.

178 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();

```

Listing 471: Bestimmung von einem Schaltjahr

```
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 471: Bestimmung von einem Schaltjahr (Forts.)

Die Funktion bekommt beim Aufruf ein Datumsobjekt als Übergabewert übergeben. Aus diesem wird in Zeile 2 mit `getYear()` das Jahr extrahiert und in der Variablen `jahr` gespeichert. Der folgende Test auf ein Schaltjahr erfolgt unter Verwendung von dem konditionalen Operator, 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 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 die Funktion gibt `true` zurück. Eine weitere Auswertung von dem konditionalen Operator 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")` 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 die Funktion gibt `false` zurück. Eine weitere Auswertung von dem konditionalen Operator 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")` 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 die Funktion gibt `true` zurück. 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 die Funktion gibt `false` zurück. Damit ist die Auswertung von dem konditionalen Operator beendet und zuverlässig identifiziert, ob ein Schaltjahr vorliegt oder nicht.

Testen wir die Funktion in einem vollständigen Beispiel (*dat21.html*):

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

```

01 <html>
02 <body>
03 <script language="JavaScript">
04 function schaltjahrTest(a) {
05     jahr = a.getYear();
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);
21     document.write("Jahr: " + a.getYear() + ", Schaltjahr: "
22 + schaltjahrTest(a) + "<br>");
23 }
24 </script>
25 </body>
26 </html>

```

Listing 472: Der Test, ob verschiedene Jahre Schaltjahre sind

In Zeile 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 310: Der Test, ob ein Jahr ein Schaltjahr ist oder nicht

179 Wie kann ich mit Datumsangaben rechnen?

In JavaScript dient der 1. Januar 1970, 0.00 Uhr, als der interne Zeitnullpunkt, der die Speicherungs- und Berechnungsbasis für alle Operationen rund um Datum und Uhrzeit darstellt. Die Zeit wird in verstrichenen Millisekunden seit diesem Zeitpunkt verwaltet. 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 ist trivial und 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 hier 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 haben⁸. Wir wollen uns darauf beschränken.

180 Wie kann ich die Differenz zwischen zwei Datumsangaben bestimmen?

Die Antwort ist primitiv. Sie ziehen zwei Datumsobjekte einfach mit dem Minusoperator voneinander ab.

Beispiel:

```
new Date(2001, 1, 1) - new Date(2001, 1, 2);
```

Listing 473: Die Differenz zwischen zwei Datumsobjekten

Als Ergebnis erhalten Sie die Anzahl der Millisekunden, die zwischen den Inhalten der beiden Datumsobjekte liegt. Schauen wir uns drei praktische Anwendungen von diesem Verfahren an.

181 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). Das Alter einer Person bestimmen Sie zum Beispiel, indem Sie ein Datumsobjekt mit deren Geburtstag bilden und dieses von dem Datumsobjekt des Zeitpunkts abziehen, an dem Sie das Alter bestimmen wollen. Das ist im Grunde ganz einfach, jedoch es gibt ein paar Feinheiten zu beachten. Schauen wir uns die Prozedur in einem kompletten Beispiel an.

8. Operationen wie die Multiplikation, Division und Modulobildung machen hauptsächlich auf den Teilaspekten eines Datumsobjekts wie den Tagen oder Stunden viel Sinn.

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 stimmen.\n";
15         fehler++;
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 ein.\n";
24             fehler++;
25         }
26         if(
27             (datum[1] < 1) ||
28             (datum[1] > 12)
29         ){
30             meldung = meldung +
31                 "Der Monat ist ungültig.\n";
32             fehler++;
33         }
34         if(
35             (datum[2]%400 == "0") ? (1) : (
36                 (datum[2]%100 == "0") ? (0) : (
37                     (datum[2]%4 == "0") ? (1) : (0)
38                 )
39             )
40         ){
41             tag = 29;
42         }
43         else{
44             tag = 28;
45         }
46         if(
47             (datum[1] == 1) ||
48             (datum[1] == 3) ||
49             (datum[1] == 5) ||
```

Listing 474: Die Altersbestimmung einer Person

```

50     (datum[1] == 7) ||
51     (datum[1] == 8) ||
52     (datum[1] == 10) ||
53     (datum[1] == 12)
54   ){
55     tag = 31;
56   }
57   if(
58     (datum[1] == 4) ||
59     (datum[1] == 6) ||
60     (datum[1] == 9) ||
61     (datum[1] == 11)
62   ){
63     tag = 30;
64   }
65   if(
66     (datum[0] < 1) ||
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>

```

Listing 474: Die Altersbestimmung einer Person (Forts.)

Grund-
lagen

Core

Formu-
lareAus-
nahme

Arrays

Strings

Datum

Platt-
form

DHTML

Events

```
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 474: 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)) {}`). Die Funktion ist von Zeile 3 bis 83 definiert und nicht ganz primitiv. *Sie finden sie aber im Formularkapitel im Rezept »Wie kann ich 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 Zeile 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 (Jahre⁹) geteilt werden.



Abbildung 311: 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 den Wert nur zweistellig (siehe dazu das Rezept »Wie kann ich das Jahr aus einem Datumsobjekt extrahieren?« auf Seite 612). 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())`)¹ dazu führt, dass `a.getYear()` einen vierstelligen Wert liefert (unter der Voraussetzung, dass das aktuelle Tagesdatum in einem Jahr jenseits von 2000 liegt²), wohingegen `gebdat.getYear()` für alle Geburtstage von dem Jahr 2000 einen zweistelligen Wert liefert. Das Ergebnis der Altersbestimmung wird dementsprechend »ungenau«³.

9. Ohne Berücksichtigung der Schaltjahre.



Abbildung 312: Methusalem weilt unter uns.

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.

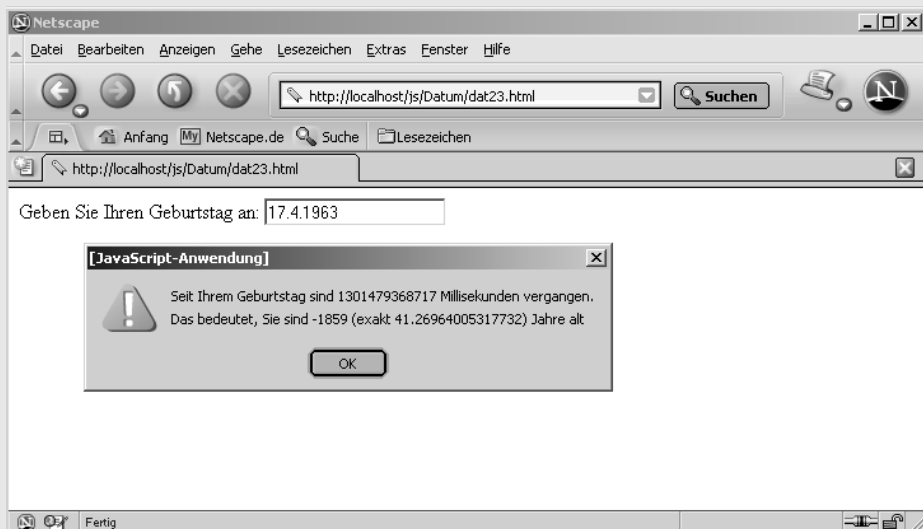


Abbildung 313: Die Familienplanung liegt noch etwas in der Zukunft.

Dafür werden diese Browser mit der zweiten Variante `dat23b.html` zurechtkommen.

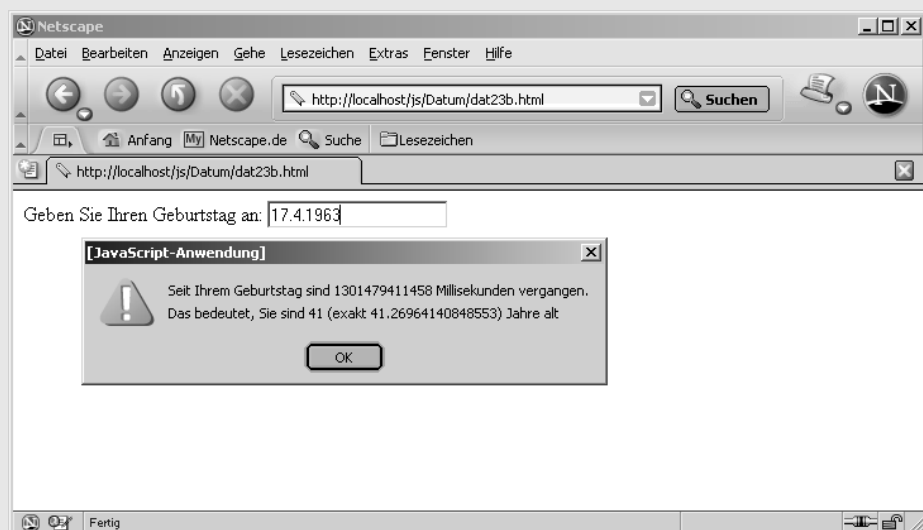


Abbildung 314: In der Variante 2 stimmt es wieder.

Um dem gesamten Problem zu begegnen, können Sie das Alter wie beschrieben aus den Millisekunden berechnen⁴, oder Sie sollten eine der Funktionen zur einheitlichen Extrahierung der Jahreszahl verwenden, wie sie im Rezept »Wie kann ich das Jahr aus einem Datumsobjekt extrahieren?« auf Seite 612 beschrieben wird:

```
01 function extrahiereJahr(a) {
02   return 1900 + ((1900 + a.getYear()) % 1900);
03 }
```

Listing 475: Eine Funktion zur identischen Extrahierung des Jahres ab 1900 in allen Browsern

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 476: Die geänderte Zeile 93 mit dem zuverlässigen Extrahieren des Jahres

1. Also die Extrahierung des Geburtsjahres aus dem Datumsobjekt mit `getYear()`.
2. Wovon auszugehen ist, solange es keine Zeitmaschinen gibt :-).
3. ;-)
4. Dann sollten Sie zur Sicherheit aber die Schaltjahre einkalkulieren und die Funktion zur Berechnung eines Schaltjahres verwenden, wie sie im Rezept »Wie kann ich ein Schaltjahr bestimmen?« auf Seite 632 beschrieben ist.

182 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 Datumsobjekt mit dem Termin des Ablaufdatums zu erzeugen und beim Laden der Webseite das Systemdatum des Besuchers abzufragen. Daraus erzeugen Sie ein zweites Datumsobjekt und bilden die Differenz. Je nachdem, ob die Differenz kleiner oder größer dem Wert 0 ist, zeigen Sie unterschiedliche Inhalte an.

Tip

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, dass 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. 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 runter gefasst!</
h2>" +
10       "<h3>Wer sich meldet, ist selbst schuld</h3>"
11     );
12   }
13   else {
14     document.write(
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 477: Dynamisches Schreiben einer Webseite aufgrund eines Ablaufdatums

In Zeile 5 erzeugen wir ein Datumsobjekt `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 Datumsobjekt `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.

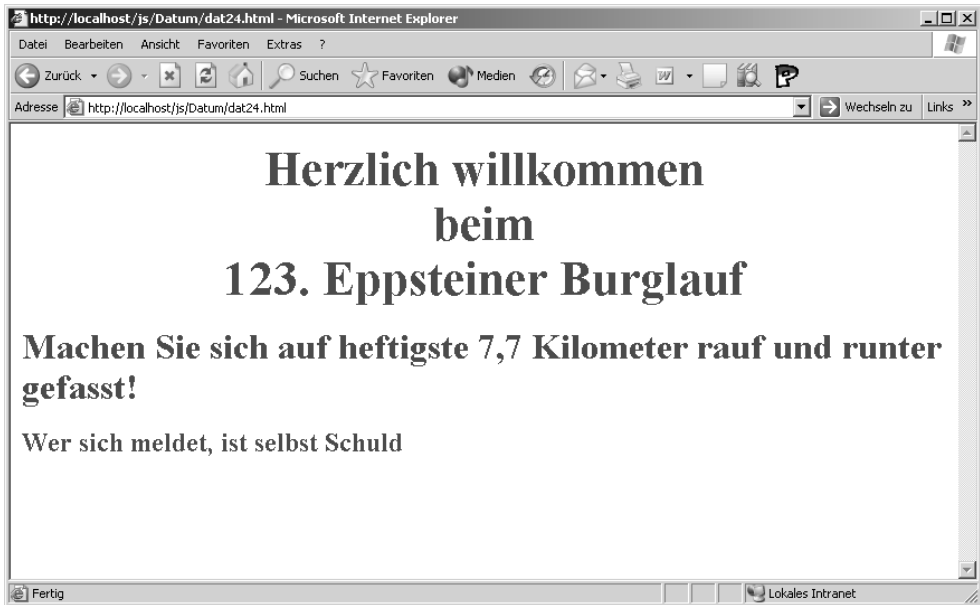


Abbildung 315: Vor dem Ablauftermin

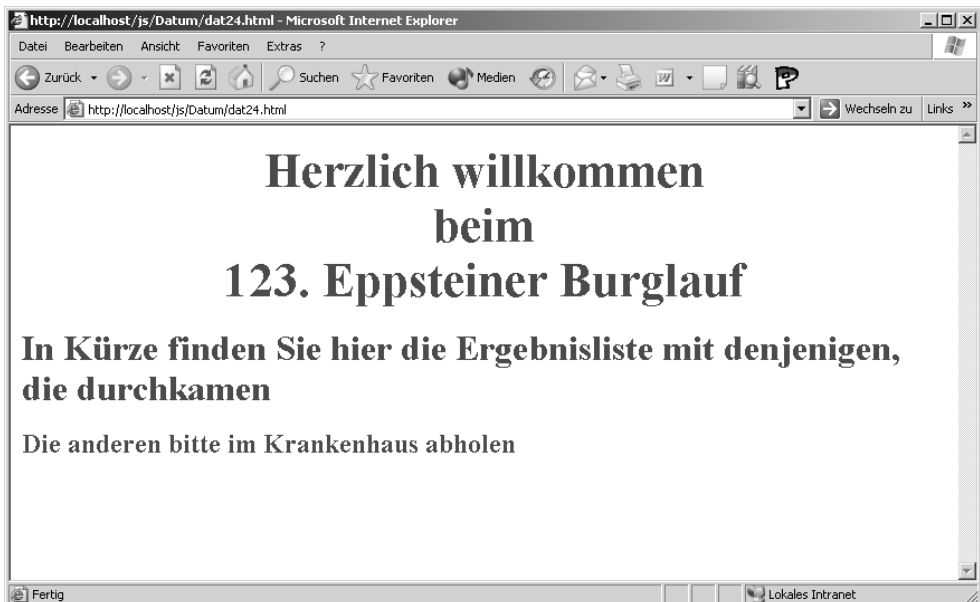


Abbildung 316: Nach dem Ablauftermin

183 Wie kann ich die Zeitspanne bis zu einem bestimmten Termin berechnen?

Eine ebenfalls interessante Anwendung der Differenzbildung zweier Datumsobjekte 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 ist. 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(2004,7,30);
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 478: Anzeige der Restzeit

In Zeile 5 erzeugen wir auch hier wieder ein Datumsobjekt `ablauf`, das den Ablauftermin repräsentieren soll (`ablauf = new Date(2004, 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 in Zeile 9 dann ausgegeben wird.

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.

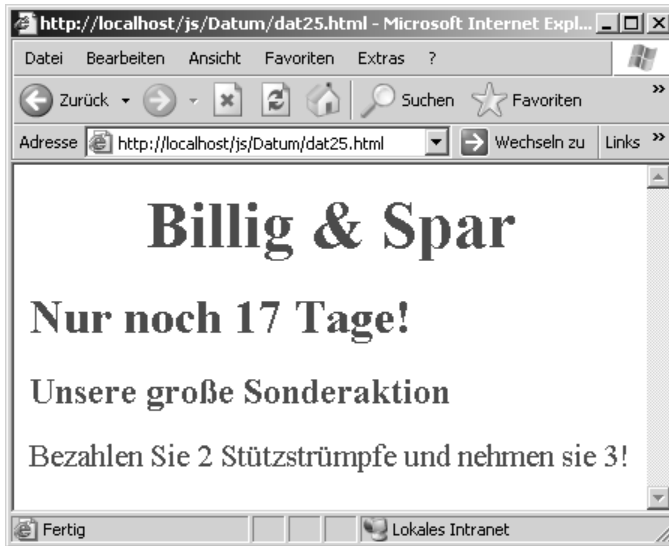


Abbildung 317: Kein Hektik – es ist noch viel Zeit für das Schnäppchen.

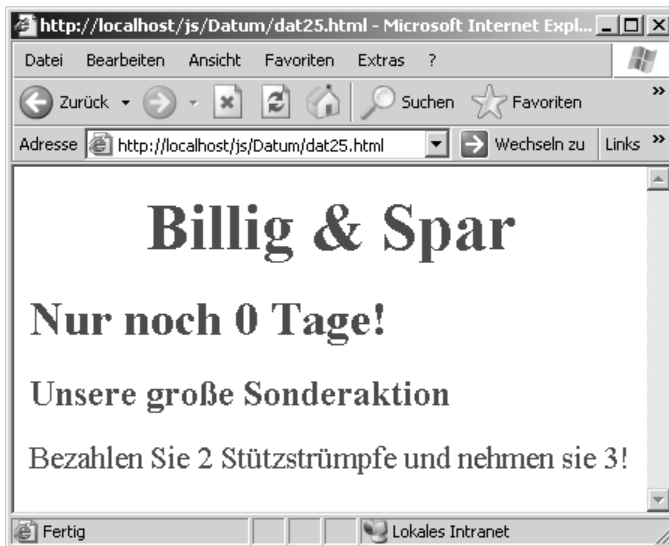


Abbildung 318: Nun aber hurtig! Aber wozu brauche ich eigentlich den dritten Strumpf?

184 Wie kann ich 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 Datumsobjekts einfach mit dem Plusoperator einen konstanten Wert. Schauen wir uns einige Beispiele mit praktischen Anwendungen an.

Wie erfolgte die Berechnung eines Termins in der Zukunft?

Angenommen, Sie wollen 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.getYear(), a.getMonth(), a.getDate() + 30);
10   document.write(
11     "Wir erwarten die Bezahlung bis zum" + b + ".");
12 </script>
13 </body>
14 </html>
```

Listing 479: 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 weiter verwendet. 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 Monats¹⁰ sprengt, führt beispielsweise einfach dazu, dass die Tage in den nächsten Monat fortgesetzt werden. Handelt es sich beim aktuellen Monat um einen Monat mit 30 Tagen, setzt die Verwendung des Werts 40 für den Monatstag

10. Von 1 bis 28, 29, 30 oder 31 – je nach Monat.

das Datum einfach auf den 10. Monatstag des Folgemonats. 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.

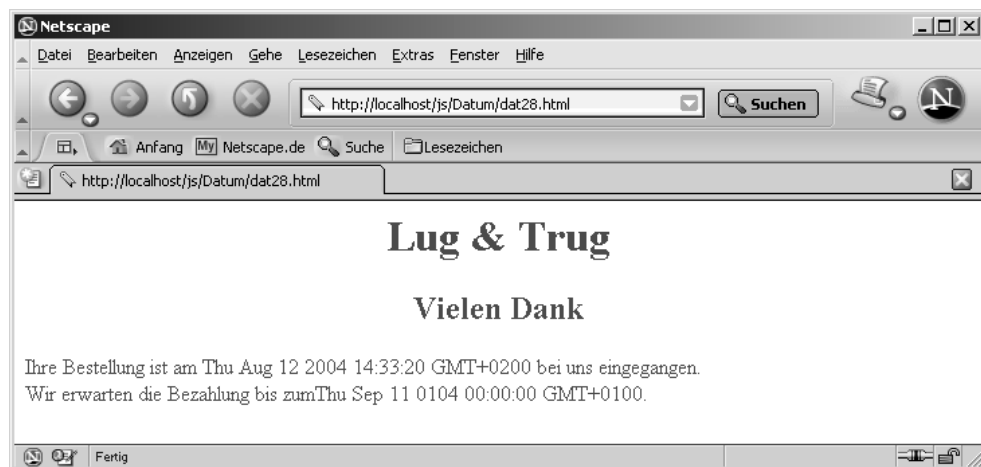


Abbildung 319: Am 12. August gekauft, und am 11. September muss die Kohle da sein.

Wie erfolgte die Berechnung regelmäßiger Termine?

Eine weitere sinnvolle Anwendung für die Addition eines konstanten Werts zu einem Teil eines Datumsobjekts wie dem Monat ist, für eine regelmäßige Veranstaltung ein regelmäßiges Intervall mehrfach aufzuaddieren. Also eine Terminserie.

Beispiel (*dat30.html*):

```
01 <html>
02 <body text="red">
03 <h1 align="center">Fifty Fast</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(2004, 0, 3);
10   for(i = 0; i < 12; i++) {
11     document.write(
12       "<tr><td>" + (1 + i) + ". Termin:</td><td> " + a + "</td></tr>");
13     a.setMonth(a.getMonth() + 1);
14   }
15 </script>
16 </table>
17 </body>
18 </html>
```

Listing 480: Die regelmäßigen Termine in einem Jahr

In Zeile 9 wird ein Datumsobjekt `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 Datumsobjekt und addieren dazu den Wert 1. Dann wird der neue Wert dazu verwendet, den Monat im Datumsobjekt mit `setMonth()` zu erhöhen. Die Darstellung der Ausgabe ist der Übersicht halber als Tabelle aufgebaut. Sonst hat die Tabelle keine Relevanz. Natürlich kann man statt der nativen Darstellung des Werts in dem Datumsobjekt eine der besser aufbereiteten Varianten wählen, wie sie in den Rezepten in dem Kapitel beschrieben wird.



Fifty Fast

Termine für die Probe

Wie proben an jedem 3. Tag eines Monats
Das sind in diesem Jahr folgende Tage:

1. Termin:	Sat Jan 3 00:00:00 UTC+0100 2004
2. Termin:	Tue Feb 3 00:00:00 UTC+0100 2004
3. Termin:	Wed Mar 3 00:00:00 UTC+0100 2004
4. Termin:	Sat Apr 3 00:00:00 UTC+0200 2004
5. Termin:	Mon May 3 00:00:00 UTC+0200 2004
6. Termin:	Thu Jun 3 00:00:00 UTC+0200 2004
7. Termin:	Sat Jul 3 00:00:00 UTC+0200 2004
8. Termin:	Tue Aug 3 00:00:00 UTC+0200 2004
9. Termin:	Fri Sep 3 00:00:00 UTC+0200 2004
10. Termin:	Sun Oct 3 00:00:00 UTC+0200 2004
11. Termin:	Wed Nov 3 00:00:00 UTC+0100 2004
12. Termin:	Fri Dec 3 00:00:00 UTC+0100 2004

Fertig Lokales Intranet

Abbildung 320: Regelmäßige Termine

185 Wie kann ich einem Anwender suggerieren, dass eine Webseite topaktuell ist?

Die meisten Besucher einer Webseite verlassen selbige wieder fluchtartig, wenn sie darauf einen Hinweis entdecken, dass die Seite bereits längere Zeit nicht aktualisiert wurde¹¹. 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 JavaScripts helfen, dem Anwender zumindest zu suggerieren, dass eine Seite auf dem neuesten Stand ist. Dabei helfen schon ganz wenige Tricks, 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.

Trick 1: Wie erfolgt das dynamische Schreiben des Aktualisierungsdatums?

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 hartkodierte Hinweis, wann die Seite zuletzt aktualisiert wurde. Die Information an sich wird durch den Hinweis selbst nicht schlechter, aber der Besucher denkt es vielleicht¹². Deshalb ist es sicher nicht schlecht, wenn man dem Besucher das gibt, was er sehen will. Schreiben wir das Aktualisierungsdatum einer Webseite einfach dynamisch mit JavaScript, statt es hartkodiert zu dort 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 `getYear()` beachten (siehe dazu das Rezept »Wie kann ich das Jahr aus einem Datumsobjekt extrahieren?« auf Seite 612). Dazu sollten die Tages- und Monatsangaben aus optischen Gründen auch dann zweistellig

11. 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 sollten nicht den Eindruck machen, dass da nur einmal in der Woche (oder gar seltener) etwas passiert.
12. Nach dem Motto – eine Webseite ist wie eine Wurst. Wenn die ein Jahr alt ist, esse ich sie ja auch nicht mehr :-).

angezeigt werden, wenn die Werte einstellig sind. Es gibt also einiges zu tun, und die Formatierung des Tagesdatums ist eine klassische Anwendung, wo 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.getYear()) % 1900);
15 }
16 var datum = new Date();
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 481: Anpassen des Aktualisierungsdatums

Beim Laden der Webseite wird in Zeile 16 ein Datumsobjekt `datum` mit dem aktuellen Systemdatum erzeugt. In Zeile 17 werden 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 `aktuelleDatum` übergeben. Dieses Objekt ist von Zeile 5 bis 15 zu finden. In Zeile 6 bis 9 kümmern wir uns darum, dass der Tag immer zweistellig angezeigt wird. In Zeile 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 das Jahr aus einem Datumsobjekt extrahieren?« auf Seite 612)).



Abbildung 321: Geschummelt – von wegen topaktuell

Tipp

Da Sie das Systemdatum des Besuchers abfragen, kann es bei dem Generieren von dem Aktualisierungsdatum 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, das Aktualisierungsdatum mit einem oder zwei Tagen Luft zum Besuchstermin zu generieren. Dann wird die Manipulation weniger offensichtlich. Vor allem, wenn ein Gast mehrfach Ihre Seite besucht.

Trick 2: Wie erfolgt das dynamische Schreiben von Inhalten aufgrund 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 Aktualisierungsdatum 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 642 und 644 haben wir gesehen, dass Sie von einem Datum abhängig 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
08 anregende Wirkung des Kaffees steigert sich gewaltig.";
09 rezepte[1] = "Als kleiner Snack für die Frühstückspause bietet sich ein
10 Wischsandwich mit Gänsewein an.";
11 rezepte[2] = "Für ein leichtes Mittagessen nehme man ein Kamel und fülle es mit
12 drei Schweinen. Dazu reiche man eine Zitrone und etwas Reis.";
13 rezepte[3] = "Den Sand für unseren Sandkuchen finden Sie auf dem Spielplatz um die
14 Ecke. Allerdings bevorzugen Liebhaber echten Strandsand.";
15 rezepte[4] = "Die Suppe für das bekömmliche Nachmahl würze man mit Zyankali und
16 einer Prise Arsen.";
17 if(new Date().getHours() < 9){
18   document.write(rezepte[0]);
19 }
20 else if(new Date().getHours() < 11){
21   document.write(rezepte[1]);
22 }
23 else if(new Date().getHours() < 13){
24   document.write(rezepte[2]);
25 }
26 else if(new Date().getHours() < 19){
27   document.write(rezepte[3]);
28 }
29 else {
30   document.write(rezepte[4]);
31 }
32 </script>
33 </body>
34 </html>
```

Listing 482: Generieren von Seitencontent, der von der Besuchsstunde abhängt

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.

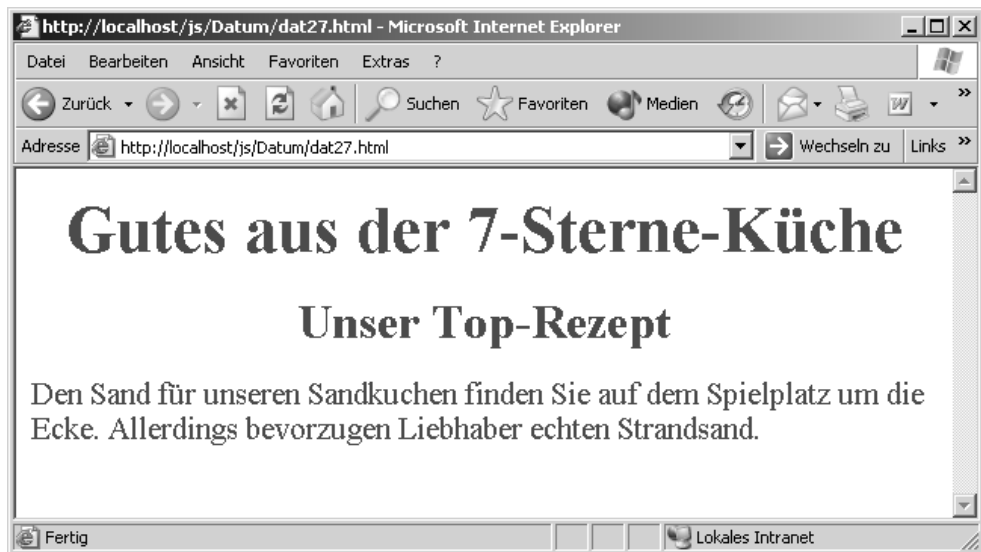


Abbildung 322: Das Nachmittagsrezept



Abbildung 323: Das gibt es zum Mittagessen

Tipp

Natürlich können Sie die Inhalte auch mit einem Zufallsprozess auswählen. Dazu gibt es mit `Math.random()` eine geeignete Technik.

Trick 3: Wie erfolgt das dynamische Verändern des Layouts mit Style Sheets aufgrund des Datums?

Eine weitere automatisierte Aktion, um einen wiederkehrenden Besucher bei Laune zu halten, ist das dynamische Verändern des Layouts einer Webseite aufgrund eines zufälligen oder – in diesem Fall 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 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-Dateien¹³), 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-Datei (*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 483: 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()` der entsprechende HTML-Tag (`<link 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 484: 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`:

```
body {
  color: #00593D;
  background: #9F00AF;
  font-family: Roman;
}
h1, h2, h3 {
  background: #A200FB;
}
a {
  color: #11FF80;
  text-decoration: none;
}
```

Listing 485: Die zweite Style-Sheet-Datei

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

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.

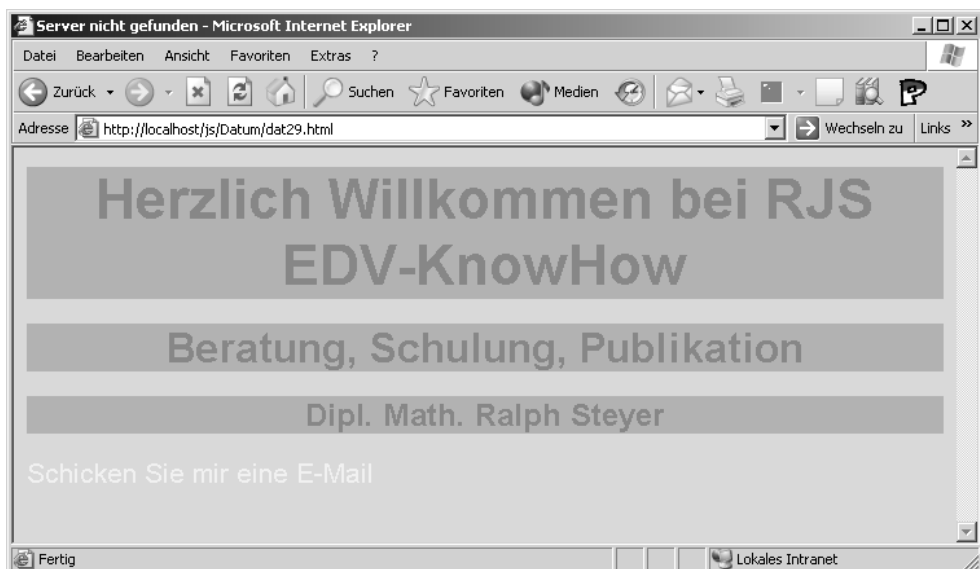


Abbildung 324: Das Layout am Vormittag



Abbildung 325: Das Layout am Nachmittag und Abend mit anderen Farben und anderer Schriftart

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 Offscreen-Bereich positionieren, befinden diese sich 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 Kriegsflagge¹ 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. Im Kapitel zu DHTML finden Sie dazu zahlreiche Rezepte.

1. Weißer Adler auf weißem Grund :-).

Trick 4: Wie kann ich aktuelle Bilder automatisch einbinden?

Ein wirkungsvoller Effekt zur Erzeugung eines aktuellen Eindrucks einer Webseite ist die Einbindung von aktuellen Bildern oder Bildersequenzen mit sich ständig ändernden Bildern. Im einfachsten Fall können animierte GIFs zum Einsatz kommen, aber besser sind Bilder, die wirklich ausgetauscht werden. 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 von dem Bild dabei immer gleich bleibt, braucht in Ihrer Webseite selbst überhaupt keine Veränderung stattzufinden. Die Webseite kann rein statisch bleiben. Nur muss 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 `<img>`-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">
```

Listing 486: Das zeitlich gesteuerte Anzeigen von Bildern

```

06  a = new Date();
07  if(a.getHours()<10) {
08      document.write(
09      '<tr><td></td>'
+
10      '<td></td>' +
11      '<td></td></tr>');
12  }
13  else if(a.getHours()<18) {
14      document.write(
15      '<tr><td></td>'
+
16      '<td></td>' +
17      '<td></td></tr>');
18  }
19  else {
20      document.write(
21      '<tr><td></td>'
+
22      '<td></td>' +
23      '<td></td></tr>');
24  }
25 </script>
26 </table>
27 </body>
28 </html>

```

Listing 486: Das zeitlich gesteuerte Anzeigen von Bildern (Forts.)

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()` der 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.

Aber Sie können ebenso mit JavaScript direkt Bilder in einer Webseite 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.



Abbildung 326: Mal das

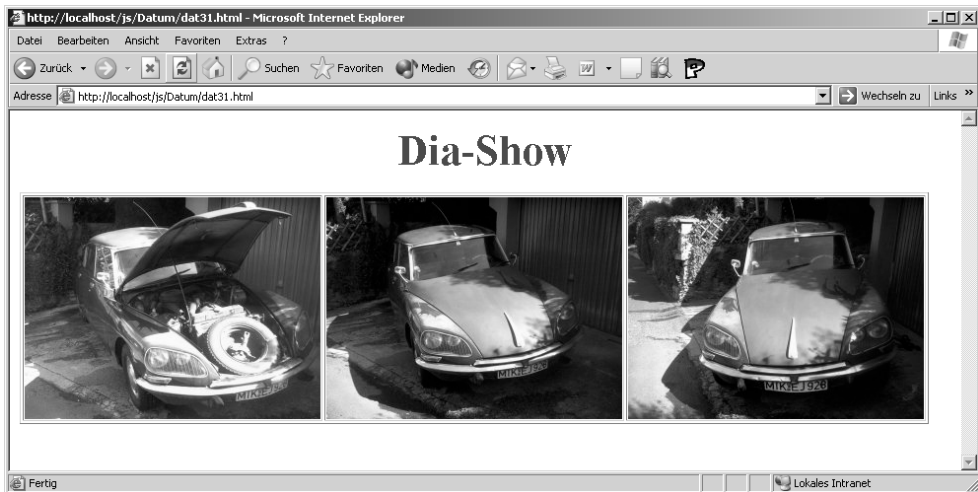


Abbildung 327: Mal dies

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";

```

Listing 487: Dynamischer Austausch von Bildern über die src-Eigenschaft

```

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></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>
35 <td>
36     
37 </td>
38 </tr>
39 </table>
40 </body>
41 </html>

```

Listing 487: Dynamischer Austausch von Bildern über die src-Eigenschaft (Forts.)

Das Beispiel enthält in den Zeile 27 bis 39 eine Tabelle mit einer Zeile und drei Spalten. In diesen werden drei Bilder über die üblichen `<img>`-Tag 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.

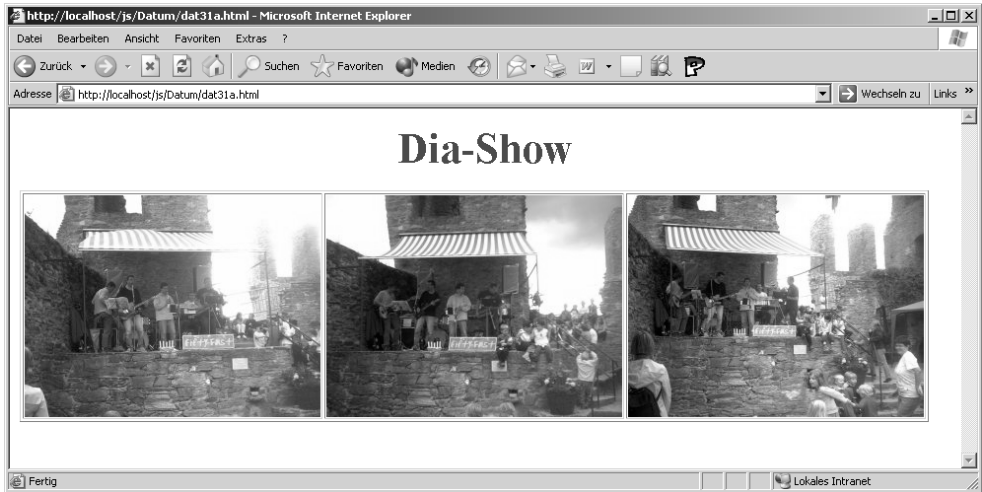


Abbildung 328: Die Anzeige von Bildern mit dem Setzen der Eigenschaft `src` eines `Image`-Objekts

Tipp

Sie benötigen zwar beim Laden der Webseite unbedingt einen Platzhalter für die Bilder, damit der Webbrowser die Objekte in dem `images`-Objektfeld anlegen kann. Allerdings ist es in einigen Browsern möglich, beim `<img>` 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 488: Austausch der Zeilen 27 bis 39

Die `<img>`-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 allgemein gültig, denn einige (meist ältere) Browser wie der Netscape Navigator 4.7 erzeugen aus `<img>`-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 329: Der Netscape Navigator 4.7 kommt nicht zurecht, wenn Sie in den `<img>`-Tags nicht per HTML bereits ein anfängliches Bild referenzieren.

Achtung

Beachten Sie, dass Sie vor dem Laden der HTML-Bildreferenzen nicht versuchen dürfen, mit JavaScript auf das `images`-Objektfeld zuzugreifen. Im Beispiel zuvor verwenden wir den Eventhandler `onLoad`, um die Funktion `bilder()` aufzurufen. Das stellt sicher, dass der Zugriff auf die Bildobjekte in dem JavaScript nicht vor dem Aufbau des `images`-Objektfeld erfolgt. Wenn Sie die JavaScript-Anweisungen direkt beim Laden der Webseite ausführen, bevor die HTML-Anweisungen mit den `<img>`-Tags interpretiert wurden, erhalten Sie einen Laufzeitfehler. Das können Sie mit dem Beispiel `dat31c.html` testen.



Abbildung 330: Zugriff auf ein Bildobjekt, das der Browser noch nicht angelegt hat

Wenn Sie dagegen den `<script>`-Container hinter die HTML-`<img>`-Tags notieren, wird das Beispiel funktionieren. Allerdings gibt es für so eine Vorgehensweise wenig einleuchtende Gründe. Der Einsatz von `onLoad` ist die vernünftigste Vorgehensweise.

Tipp

Wenn ein Besucher einer Webseite längere Zeit auf der Seite verweilt, bleiben die dort angezeigten Informationen statisch erhalten (mit Ausnahme von Animationen jeglicher Form, die bereits vom Client geladen wurden). Unter Umständen wollen Sie aber nach einem gewissen Zeitintervall vom Server Informationen automatisch nachladen. Die HTML-Datei muss sich dazu im Browser des Clients automatisch in gewissen Zeitintervallen aktualisieren. Grundsätzlich besteht der Trick darin, dass sich eine Webseite nach einem definierten Zeitintervall wieder automatisch selbst aufruft, entweder durch eine Anfrage des Clients (wozu unsere beiden nachfolgenden Varianten zählen) oder eine Aktion durch serverseitige Befehle (was wir nicht weiter verfolgen). Damit können Sie beispielsweise dann alternative Bilder laden oder auch andere Informationen anzeigen.

Hinweis

Besonders in Verbindung mit Framesets macht ein solcher zeitgesteuerter, automatischer Selbstaufwurf Sinn.

Ein Rezept für einen solchen Selbstaufwurf basiert ausschließlich auf HTML. Der `<meta>`-Tag verfügt über das `refresh`-Attribut, um damit 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 489: Der 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 490: 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   }
```

Listing 491: Die sich selbst mit dem <meta>-Tag aufrufende Webseite

```

18 else {
19     document.write(
20         '');
21     }
22     //-->
23 </script>
24 <noscript></noscript>
25 </div>
26 </body>
27 </html>

```

Listing 491: Die sich selbst mit dem <meta>-Tag aufrufende Webseite (Forts.)

Wichtig in dieser Webseite, die im unteren Frame des Framesets geladen wird, ist die Angabe `<meta http-equiv="refresh" content="20; URL=datunten.html">` in Zeile 3, mit der die Datei sich selbst alle 20 Sekunden aufruft und damit immer ein anderes Bild anzeigt, 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.



Abbildung 331: Das laufend aktualisierte Bild in der Webseite beim Client

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></noscript>
26 </div>
27 </body>
28 </html>
```

Listing 492: Die JavaScript-Variante mit der Datei, die sich immer wieder selbst aufruft

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 zu rekursivem Aufruf in bestimmten Zeitintervallen (Wert 1000 = 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.

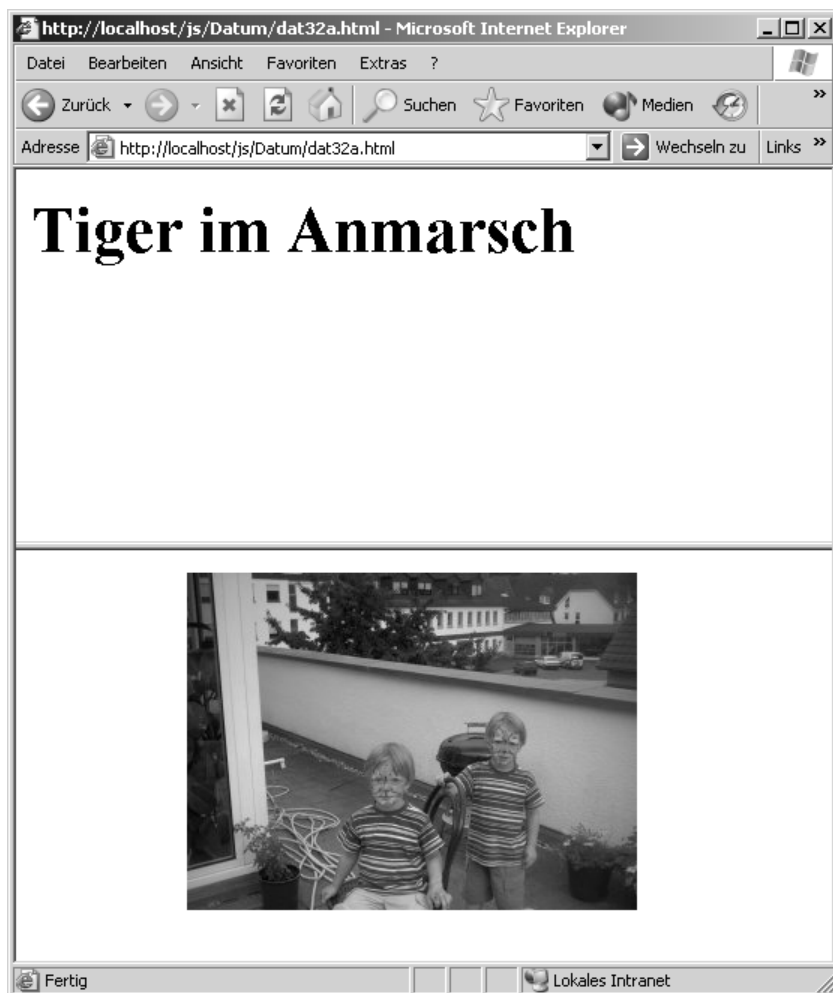


Abbildung 332: Auch hier aktualisiert sich der untere Frameset in regelmäßigen Abständen.

Trick 5: Wie kann ich Fremdinhalte einbinden?

Kein JavaScript-Trick, aber ebenso nützlich zum Anbieten von aktuellen Informationen ist die Einbindung fremder Inhalte, etwa Wetterdaten aus Ihrer Region, aktuelle Schlagzeilen etc. Diese Informationen sind zum Teil kostenpflichtig¹⁴, 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 Aktualisierungsdatums per JavaScript dann sogar in jedem Fall gerechtfertigt.

14. Sie dürfen sowieso in der Regel fremde Inhalte nicht ohne Genehmigung des Anbieters in Ihr Webprojekt einbinden.

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 Aktualisierungsdatums). Das nimmt er Ihnen vielleicht übel. Sie sollten die Routinen zum dynamischen Schreiben der Inhalte vielleicht ein bisschen verstecken, zum Beispiel in eine externe JavaScript-Datei auslagern.

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 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.

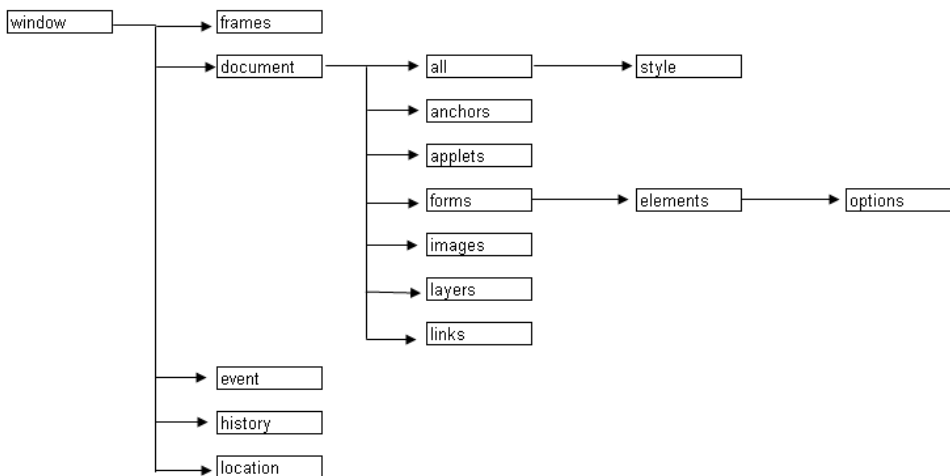


Abbildung 333: Das Objekt `window` mit Unterobjekten

In diesem Kapitel finden Sie Rezepte zum Zugriff auf das Browserfenster selbst, auf Instanzen eines Browserfensters sowie Frames, auf die Historie des Browser, 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.

186 Wie kann ich den Inhalt einer Webseite dynamisch schreiben?

Die Webseite steht Ihnen in JavaScript als Objekt zur Manipulation über das Objekt `document` bereit. Dieses ist ein direktes Unterobjekt von `window` und stellt zahlreiche Eigenschaften (von denen viele selbst wiederum Objekte sind) und Methoden bereit. Wesentliche Grundlagen für die Behandlung einer Webseite als Objekt und Zugriff darauf sowie die 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 die Behandlung von Bestandteilen der 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 Konzept beinhaltet verschiedene Teilaspekte. 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 einer Webseite bezüglich ihres Typs, ihrer relevanten Eigenschaften und ihrer Position innerhalb der Webseite indizieren. Dies ist eine Art dynamische Datenbank im Hauptspeicher des Rechners, die 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, d.h., in der Denkweise einer relationalen Datenbank in den gleichen Datenbanktabellen eingetragen. Auf diese Weise hat der Browser nach dem Laden der Webseite genaue Kenntnis über alle relevanten Daten sämtlicher eigenständig für ihn 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 bestimmter HTML-Tag) wird beim Laden einer Webseite in einem eigenen Datensatz abgespeichert, der bei Bedarf auch während der Lebenszeit der Webseite upgedatet 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.

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 liegende Idee ab dem Internet Explorer 3.0 in Verbindung mit VBScript (wenngleich vollkommen unterschiedlich implementiert). Die Auswirkungen der 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 jedoch nicht von Belang.

187 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([anzuweisender Text]);
```

Listing 493: 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 Schreibaufruf 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 geschlossen².

Hinweis

Beachten Sie dazu auch die Ausführungen beim Rezept »Wie kann ich ein Dokument explizit öffnen oder schließen?« auf Seite 675.

Beim Schreiben einer Webseite als 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

-
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 in die aufrufende eine bereits vollkommen aufgebaute 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.

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 741). Zuerst brauchen wir einen Frameset.

Beispiel (*f16.html*)

```
01 <html>
02 <frameset cols="30%,70%">
03   <frame src="fl.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 494: Ein einfacher Frameset

In dem Frameset werden zwei Spalten definiert. In der linken Spalte wird die Datei *fl.html* geladen und in der rechten Spalte die Datei *fr.html*. Die beiden Frames erhalten über das `name`-Attribut in der HTML-Notation Namen in dem Frameset zugeordnet. Der linke Frame ist damit aus JavaScript heraus über den Namen `links` anzusprechen und der rechte nahe liegender Weise über `rechts`. Die Datei *fr.html* ist leer, und wir brauchen sie nur als Platzhalter. Allerdings ist bewusst die Hintergrundfarbe gesetzt, um zu erkennen, wie sich folgende Schreiboperationen auswirken.

Beispiel (*fr.html*):

```
01 <html>
02 <body bgcolor="red">
03 </body>
04 </html>
```

Listing 495: Eine reine Platzhalterdatei



Abbildung 334: Der 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></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 496: 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 Frame³. 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 ändert⁴.

-
3. Man kann hier auch mit einem Element des Objektfelds `frames` arbeiten.
 4. Sie können natürlich auch in einem Browser wie dem Netscape Navigator den Quelltext ansehen, den der Browser tatsächlich interpretiert.

Die Zeile 6 sollte Ihnen auffallen, denn obwohl die 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 dem 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.



Abbildung 335: Die erste Schreiboperation

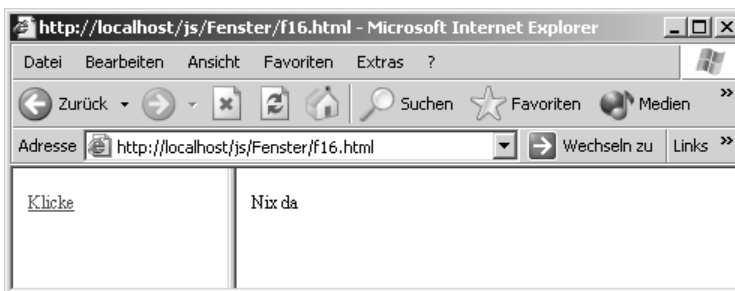


Abbildung 336: Eine Sekunde danach

Testen Sie das Beispiel auch, indem Sie mehrfach mit dem Klick auf den Hyperlink die Schreiboperation auslösen. Der Inhalt wird immer wieder an den vorherigen angehängt, wobei das Wort `da` immer zeitverzögert geschrieben wird.

5. Dieser hat ja den bestehenden Inhalt überschrieben.



Abbildung 337: Es wird immer weiter in das Dokument geschrieben.

188 Wie kann ich ein Dokument explizit öffnen oder schließen?

Das gerade behandelte Beispiel sollte noch einige Fragen offen lassen. Fassen wir das Verhalten zusammen: 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 wo 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. Das Verhalten erscheint möglicherweise nicht ganz schlüssig, aber es hängt daran, 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 wenige 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. Es gibt allerdings Situationen, da wollen Sie selbst bestimmen, ob eine Webseite geöffnet wird oder auch geschlossen. Dazu gibt es auch beim `document`-Objekt die Methoden `open()` und `close()`.

Die 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 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 von `document.write()` oder explizit mit `document.open()`). Beachten Sie folgende Überarbeitung der Datei `fr.html`, die in `f17.html` geladen wird.

Beispiel (*fr1.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);
08   top.rechts.document.close();
09 }
10 //-->
11 </script>
12 <noscript></noscript>
13 <body text="#000000" bgcolor="#FFFFFF" link="#FF0000" alink="#FF0000"
14   vlink="#FF0000">
15   <a href="javascript:schreiben()">Klicke</a>
16 </body>
17 </html>
```

Listing 497: Das Schreiben in einem anderen Frame

In Zeile 5 öffnen wir explizit das Dokument im rechten Frame. Dann wird die erste Ausgabe in dem 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 dem 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 jedoch, wenn Sie auf den expliziten Öffnungsvorgang verzichten und nur die Datei am Ende schließen. Wenn Sie den zweiten Schreibvorgang nicht zeitverzögert ausführen, ist das Resultat 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 aber die Datei 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, 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. Mit der ersten Schreiboperation durch `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 an den bestehenden Inhalt den Text `da` an. Das bleibt so bei jedem Aufruf.

189 Wie habe ich Zugriff auf Layoutinformationen in einer Webseite über document?

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.

Tabelle 10: Zugriff auf die Parameter des `<body>`-Tags

Eigenschaft	Beschreibung
vlinkColor	Farbe für Verweise zu bereits besuchten Verweiszeilen, wie sie bei der Zusatzangabe vlink im <body>-Tag oder vom Anwender in seinen Browser-Einstellungen festgelegt wurde.

Tabelle 10: Zugriff auf die Parameter des <body>-Tags (Forts.)

Schauen wir uns ein Beispiel an, das die Eigenschaften abfragt und in der Webseite die Werte dann 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("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 + "<hr>");
11 document.write("Angabe der Hintergrundfarbe der HTML-Datei, wie sie bei der
Zusatzangabe bgcolor im Body oder vom Anwender in seinen Browser-Einstellungen
festgelegt wurde: " + document.bgColor + "<hr>");
12 document.write("Die Textvordergrundfarbe der HTML-Datei, wie sie bei der
Zusatzangabe text im Body oder vom Anwender in seinen Browser-Einstellungen
festgelegt wurde: " + document.fgColor + "<hr>");
13 document.write("Angabe der Farbe für Verweise zu noch nicht besuchten
Verweiszeilen, wie sie bei der Zusatzangabe link im Body oder vom Anwender in seinen
Browser-Einstellungen festgelegt wurde: " + document.linkColor + "<hr>");
14 document.write("Farbe für Verweise zu bereits besuchten Verweiszeilen, wie sie bei
der Zusatzangabe vlink im Body oder vom Anwender in seinen Browser-Einstellungen
festgelegt wurde: " + document.vlinkColor);
15 //-->
16 </script>
17 </body>
18 </html>
```

Listing 498: Die Ausgabe von Layoutinformationen

In der Webseite werden mit `document.write()` die verschiedenen globalen Layoutinformationen des <body>-Tags ausgegeben.



Abbildung 338: 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 der `<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
05 Verweiszielen, bevor der &lt;body&gt;-Tag geladen wurde: " + document.alinkColor +
06 "<hr>");
07 </script>
08 <noscript></noscript>
09 <body text="#000000" bgcolor="#FFFFFF" link="#FF0000" alink="#FF0000"
10 vlink="#FF0000">
11 <div align="center">
12 <h1>Willkommen auf der Webseite</h1>
13 </div>
14 <a href="http://rjs.de">Klick</a>
15 <br>
16 <script language="JavaScript">
17 <!--
18 document.write("Die Farbe für Verweise zu bereits besuchten Verweiszielen, wie
19 sie bei der Zusatzangabe alink im Body oder vom Anwender in seinen Browser-
20 Einstellungen festgelegt wurde: " + document.alinkColor + "<hr>");
21 </script>
```

```

19 </body>
20 </html>

```

Listing 499: 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 339: Ausgabe der Farbeinstellungen

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.

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";

```

Listing 500: Das Setzen der globalen Farbwerte der Webseite

```

16 document.alinkColor = "#000055";
17 }
18 function m3() {
19 document.bgColor = "#000000";
20 document.vlinkColor = "#007f00";
21 document.linkColor = "#77ddd0";
22 document.fgColor = "#0fffff";
23 document.alinkColor = "#4fff55";
24 }
25 //-->
26 </script>
27 <noscript></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 500: 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 Layouteigenschaften gesetzt.



Abbildung 340: Das Originalaussehen



Abbildung 341: Das geänderte Layout

190 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, 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 Anker 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("anchors[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);
```

Listing 501: Zugriff auf `anchors`-Eigenschaften

```

13 }
14 </script>
15 <body text="#001100" bgcolor="#EEEEFF" 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></noscript>
30 </body>
31 </html>

```

Listing 501: 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.

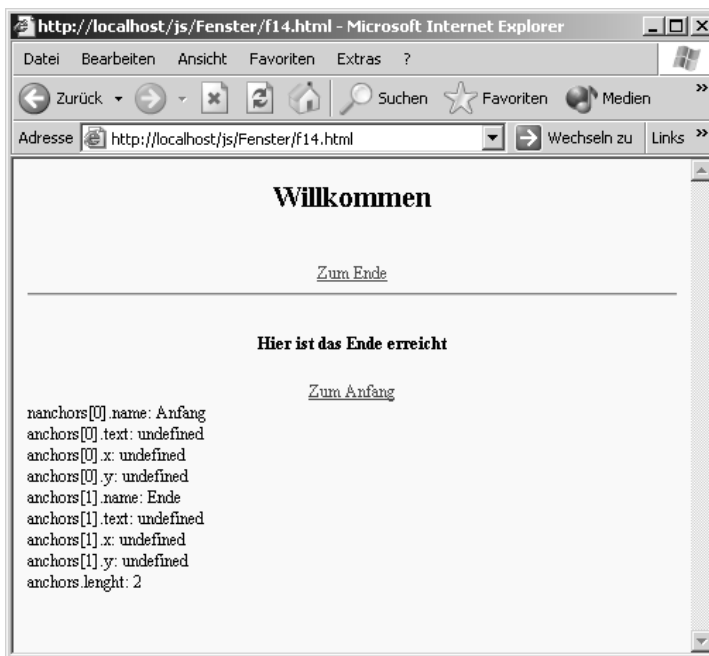


Abbildung 342: Eigenschaften der Anker, soweit ein Browser diese Eigenschaften unterstützt

191 Wie kann ich auf Java-Applets in einer Webseite zugreifen?

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 nahe liegend, dass die beiden Techniken miteinander kommunizieren und sich ergänzen können. Insbesondere können Sie aus JavaScript heraus Java als Erweiterung der eingeschränkten JavaScript-Möglichkeiten nutzen.

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, mit `document.applets[1]` auf das zweite zu usw. Das Objektfeld stellt die gleichen Eigenschaften aller Objektfelder bereit, die sich aufgrund der Datenfeldstruktur ergeben. Über `applets` kann deshalb unter anderem auch die Anzahl der Java-Applets in einer Webseite bestimmt werden (mit der Eigenschaft `length`).

Aber die Möglichkeiten 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` deklarierten Eigenschaften und Methoden zur Verfügung. JavaScript und Java lassen sich im Rahmen einer Webseite also gut miteinander verbinden. In beide Richtungen. Also um von Java auf JavaScript zuzugreifen und umgekehrt, wobei wir hier natürlich den Weg von JavaScript nach Java verfolgen. 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 502: 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 von einem Applet (`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 Passwörter 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 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() {
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", 5, 25);
16     }
17 }
```

Listing 503: Passwortüberprüfung in einem Java-Applet

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 Variablendeklarationen 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`. D.h., 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 äußerst zuverlässige Java-Sicherheitskonzept vor Zugriffen aus anderen Java-Klassen heraus versteckt. Diese Variable ist damit sehr effektiv versteckt. Sie enthält den Kontrollzugangswert, der natürlich nicht nach außen gegeben werden darf (bei der Wertzuweisung setzt auch die individuelle Verwaltung von Passwörtern 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 Java-

Script wieder zur Verfügung steht (weil die gesamte Methode öffentlich ist). Dort kann dann aufgrund des Rückgabewertes entschieden werden, wie weiter zu verfahren ist⁶.

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].password.value;
06     alert(document.forms[0].password.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="blue">
18 <applet code="Passw.class" width="400" height="80">
19 </applet>
20 <form name="form1">
21     Geben Sie Ihr Passwort ein:
22     <input name="password" type=password>
23     <br>
24     <input type=button value="Ueberprüfe Passwort" onClick="pwtest()">
25 </form>
26 </body>
27 </html>
```

Listing 504: Die Verlagerung der Überprüfung von einem Passwort in ein Java-Applet

Innerhalb der HTML-Datei wird ein Applet referenziert (Zeile 18 und 19). Das nachfolgende Formular enthält ein Eingabetextfeld, das als Passwortfeld formatiert ist (Zeile 22 – `<input name="password" type=password>`), und eine Schaltfläche, die beim Klick darauf mit dem Eventhandler `onClick` eine JavaScript-Funktion mit Namen `pwtest()` aufruft (Zeile 24), die in Zeile 4 bis 14 definiert ist. Die wichtigste Funktionalität des Beispiels steckt in der Zeile 5 (`document.applets[0].pw = document.forms[0].password.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.

6. Selbstverständlich könnte diese Entscheidung auch innerhalb von dem Java-Applet erfolgen.

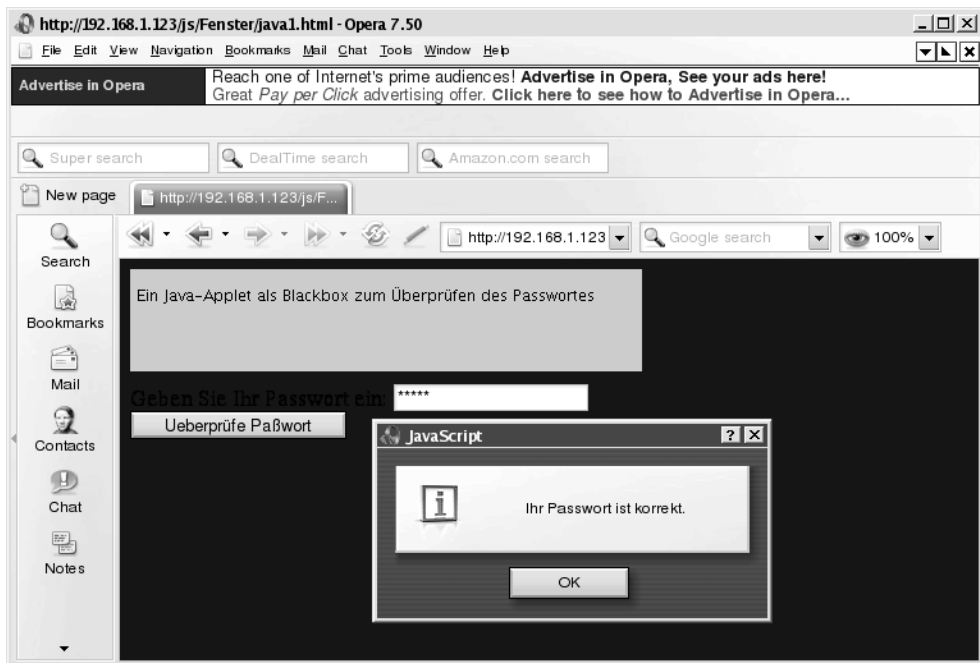


Abbildung 343: Das Passwort ist korrekt – hier im Opera

192 Was ist LiveConnect, und wie kann ich es nutzen? Der direkte Java-Zugriff

In den gleichen Zusammenhang wie der Java-Zugriff über `document.applets` auf Applets fällt dieses Rezept. Bereits sehr frühzeitig hat Netscape 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 direkt Java-Funktionalität in JavaScript bereitstellen. Dieses Konzept wird *LiveConnect* genannt. Über ein spezielles JavaScript-Objekt mit Namen `Packages` hat man in diesem Konzept Zugriff auf alle unter Java als `public` deklarierten Methoden und Felder 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 `JSONArray` (eine Instanz eines Java-Datenfelds, die an JavaScript weitergereicht werden kann), `JavaScriptObject` (eine Instanz eines beliebigen Java-Objekts, welche an JavaScript weitergereicht werden kann) und `JavaScriptPackage` (eine JavaScript-Referenz auf ein Java-Package).

Das Beispiel kollidiert bei einigen Browsern wie Mozilla und neueren Versionen des Navigators mit dem Sicherheitsmodell für Java. Das bedeutet aber nicht, dass grundsätzlich der Zugriff auf Java aus JavaScript heraus dort nicht möglich ist.

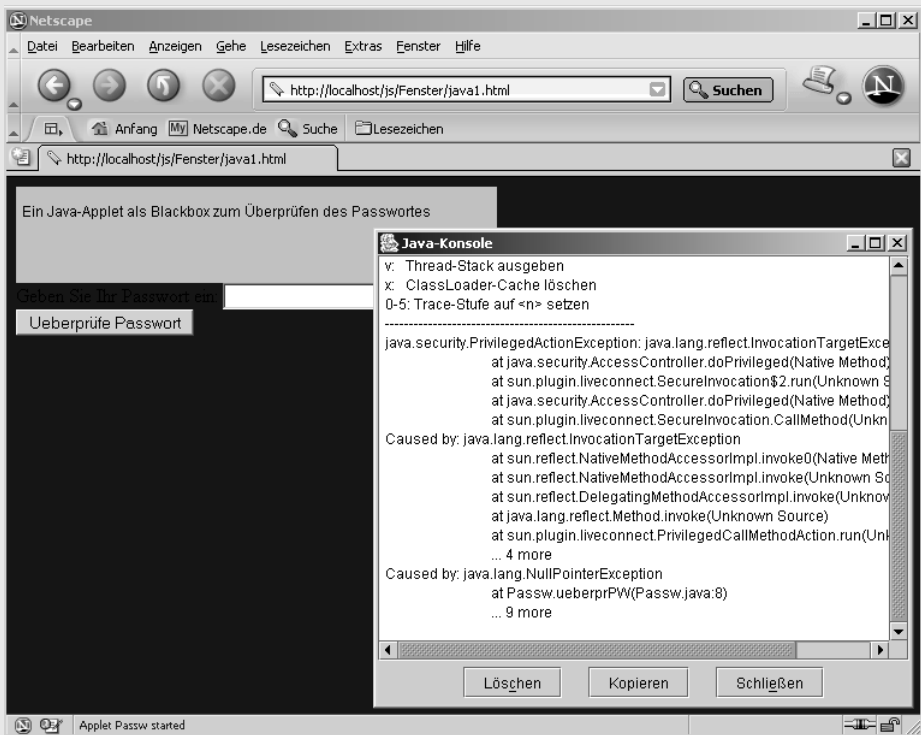


Abbildung 344: In der Java-Konsole sehen Sie die Sicherheitsverletzung.

Der Zugriff aus JavaScript heraus auf den Konstruktor einer beliebigen Java-Klasse erfolgt mit folgender Syntax:

```
[JavaScriptObjektvariable] = new Packages.[Klasse];
```

Listing 505: 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 506: 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 507: Erzeugen eines Java-Fensters mit JavaScript ohne Packages

Das `Packages`-Objekt stellt überdies mit der Eigenschaft `className` den Pfadnamen zu jeder aus JavaScript erreichbaren Java-Klasse zur Verfügung. Damit und mit dem `Packages`-Objekt kann dann sogar auf Java-Klassen außerhalb der direkt auf Toplevel-Ebene zur Verfügung stehenden Pakete zugegriffen werden.

Hinweis

Der explizite Datenaustausch zwischen JavaScript nach Java ist nicht ganz trivial. 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` gewandelt, welches über das Paket `netscape.JavaScript` unter Java bereitgestellt wird¹. Die allgemeinen Regeln für die Umwandlung liegen 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.

1. Auch der umgekehrte Weg der Übergabe von Daten aus Java nach JavaScript liegt fest, etwa wird ein `boolean` Wert unter Java zu `Boolean` in JavaScript oder die Java-Datentypen `byte`, `char`, `short`, `int`, `long`, `float` und `double` allesamt zu `Number`.

Das erste einfache Beispiel demonstriert das Öffnen von einem Java-Fenster 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 eingeschränkt unterstützt. Interessanterweise unterstützt der Opera-Browser in allen neueren Versionen LiveConnect problemlos. Außerdem muss in einem Browser natürlich sowohl die JavaScript-Unterstützung als auch die Java-Unterstützung aktiviert sein. Und da Java-Unterstützung im Internet Explorer sowieso sehr schwach ist, ist die fehlende Unterstützung für LiveConnect in diesem Browser keine wirkliche Einschränkung.

Beispiel (*java2.html*):

```
01 <html>
02 <script language="JavaScript">
```

Listing 508: Zugriff auf java.awt.Frame aus 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 508: Zugriff auf `java.awt.Frame` aus JavaScript (Forts.)

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 von dem Fensterobjekt, Zeile 7 die Titelzeile, und Zeile 8 zeigt das Java-Fenster an. Beachten Sie, dass der Java-Code keinen Schließbefehl für das Fenster enthält. Das Schließen des Browserfensters beendet aber auch das Java-Fenster, oder Sie schließen es mit dem Betriebssystem.



Abbildung 345: Ein per JavaScript-Funktion aufgerufenen Java-Fenster im Navigator 4.7

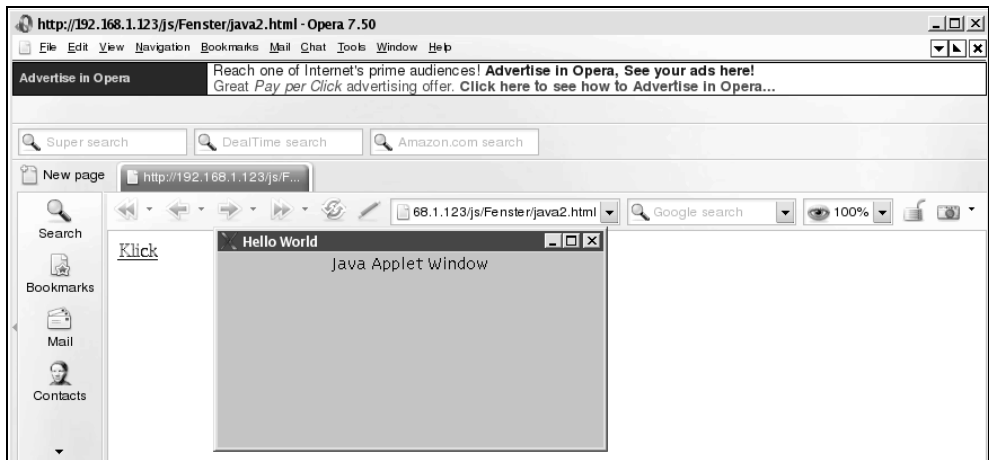


Abbildung 346: Der Opera-Browser hält die LiveConnect-Fahne hoch – hier die Version 7.50 unter Linux.

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 spionage() {
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 + "<br>" +
10     "Die Version ist: " + dieOSversion + "<br>" +
11     "Ihre Java-Version ist: " + dieJavaVersion);
12 }
13 //-->
14 </script>
15 <body onLoad="spionage()">
16 </body>
17 </html>

```

Listing 509: Zugriff auf Systeminformationen aus JavaScript heraus via Java

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 Webbrowser verwendet wird. In Zeile 9 bis 11 wird mit diesen Informationen dynamisch die Webseite geschrieben.



Abbildung 347: Der Navigator 7.x gibt auch bei deaktiviertem Java diese Informationen aus.

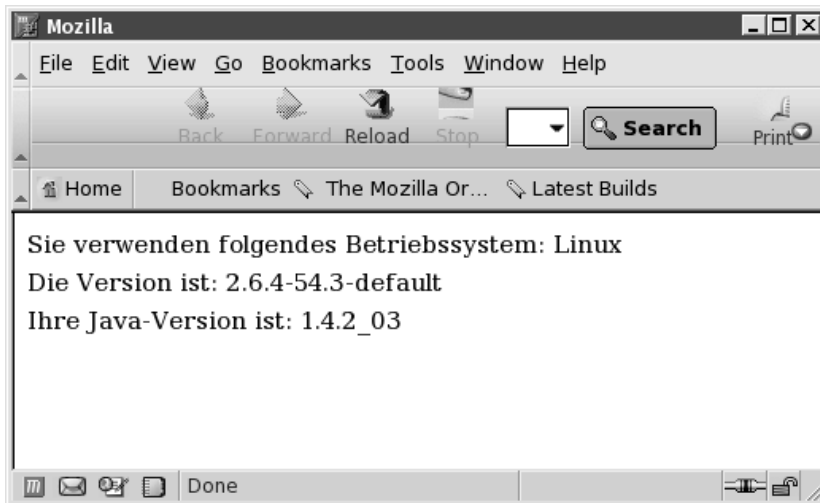


Abbildung 348: Auch Mozilla ist dabei.

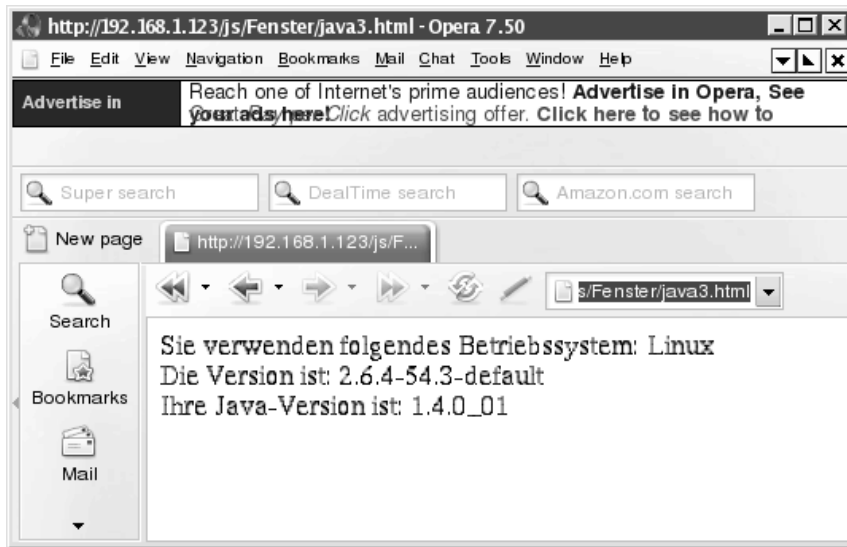


Abbildung 349: Opera connected problemlos live :-)

193 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 aufgrund der Größe von dem Browserfenster. Die Höhe ergibt sich jedoch nicht aufgrund der Größe von dem Browserfenster, sondern aufgrund der tatsächlich benötigten Pixel bis zum Ende der Webseite. Aufgefüllter Leerraum in der Webseite wird nicht berücksichtigt.

Hinweis

Das Setzen der Werte für die Eigenschaften `document.height` bzw. `document.width` mit JavaScript macht 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 +
```

Listing 510: Die Abfrage von der Breite und Höhe eines Dokuments

```

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 510: Die Abfrage von der Breite und Höhe eines Dokuments (Forts.)

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 350: 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 351: Der Internet Explorer bekommt es nicht hin.

Gut kommen Mozilla, Navigator und Konqueror damit zurecht.

194 Wie kann ich den Termin 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 (*f23.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04 function t() {
05   alert("Letzte Änderung: " + document.lastModified);
06 }
07 //-->
08 </script>
09 <body>
10 <a href="javascript:t()">Klick</a>
11 </body>
12 </html>
```

Listing 511: Abfrage des Änderungsdatums

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 den meisten Browsern im internationalen Format nach GMT (Greenwich-Zeit) dargestellt.

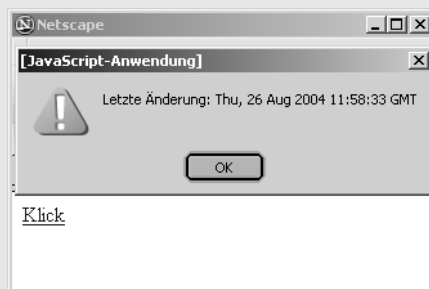


Abbildung 352: Die Ausgabe des Datums im GMT-Format

Allerdings geht der Internet Explorer seinen üblichen Sonderweg.



Abbildung 353: Die Darstellung vom Datum der letzten Änderung im Internet Explorer

195 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 hat selbst 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 716), sowie die Eigenschaft `target` für die gleichnamige HTML-Angabe.

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 (*f14.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 +
```

Listing 512: Zugriff auf die Eigenschaften von Hyperlinks

```

12     "\nhref: " + document.links[i].href +
13     "\ntarget: " + document.links[i].target +
14     "\nhostname: " + document.links[i].hostname);
15 }
16 //-->
17 </script>
18 <body>
19 <a href="javascript:linki(0)">Klicke</a>
20 <br>
21 <a href="javascript:linki(2)">Klocke</a>
22 <br>
23 <a href="http://rjs.de" target="neu">Klacke</a>
24 </body>
25 </html>

```

Listing 512: Zugriff auf die Eigenschaften von Hyperlinks (Forts.)

In der Webseite gibt es drei Hyperlinks. In Zeile 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 einen 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 354: Die Informationen über einen Hyperlink mit einem Inline-Aufruf eines JavaScripts

196 Wie kann ich auf den vollständigen URL einer HTML-Datei zugreifen?

Über die `location`-Eigenschaft von `document` (nicht von `window`!) haben Sie Zugang zum 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.

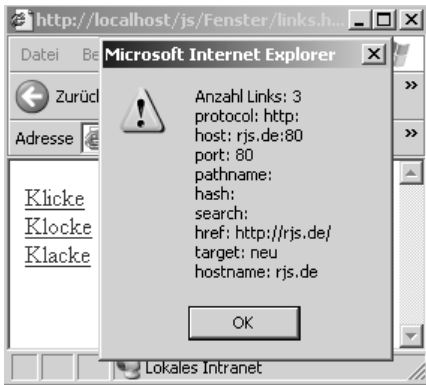


Abbildung 355: Die Informationen zu einem externen Link

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>
16 <br>
17 <a href="javascript:aenderURL2()">Klick</a>
18 </body>
19 </html>

```

Listing 513: Zugriff auf den URL der Webseite

In Zeile 15 bzw. 17 sind zwei Inline-Referenzen notiert, die die Funktion `aenderURL1()` bzw. `aenderURL2()` aufrufen. Die erste Funktion verwendet `document.location`. In Zeile 5 wird der 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.

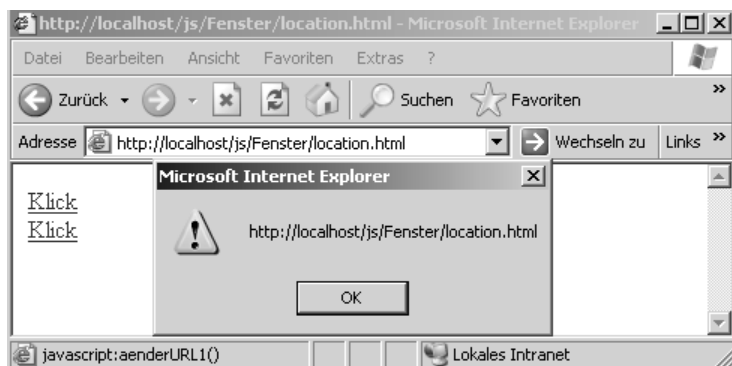


Abbildung 356: Abfrage des URLs

197 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 jedes Plug-In 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 Beispiel für die Anwendung soll auf die verwandten Rezepte wie den Zugriff auf alle Hyperlinks verwiesen werden.

198 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 festlegen.

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 514: 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.

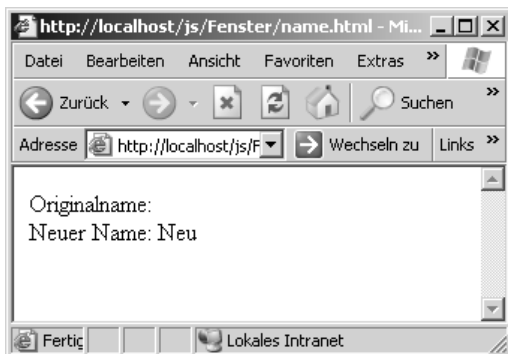


Abbildung 357: Zugriff auf den Namen eines Fensters

199 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 jawaschkrikt";
10 }
11 //-->
12 </script>
13 <body>
14 <a href="javascript:aenderTitel()">Klick</a>
15 </body>
16 </html>

```

Listing 515: Auslesen und Setzen von 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.



Abbildung 358: Auslesen des Werts von `title`



Abbildung 359: Die mit JavaScript geänderte Eigenschaft von `title` sieht man in der Titelzeile des Browsers.

200 Wie kann ich ermitteln, welchen Text ein Anwender im Dokument selektiert hat?

Das `document`-Objekt besitzt die Methode() `getSelection()`, 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="#EEEEFF" link="#FF0000" alink="#FFF000"
  vlink="#FFAA00" >
```

Listing 516: Zugriff auf selektierten Text

```

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 516: Zugriff auf selektierten Text (Forts.)

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.

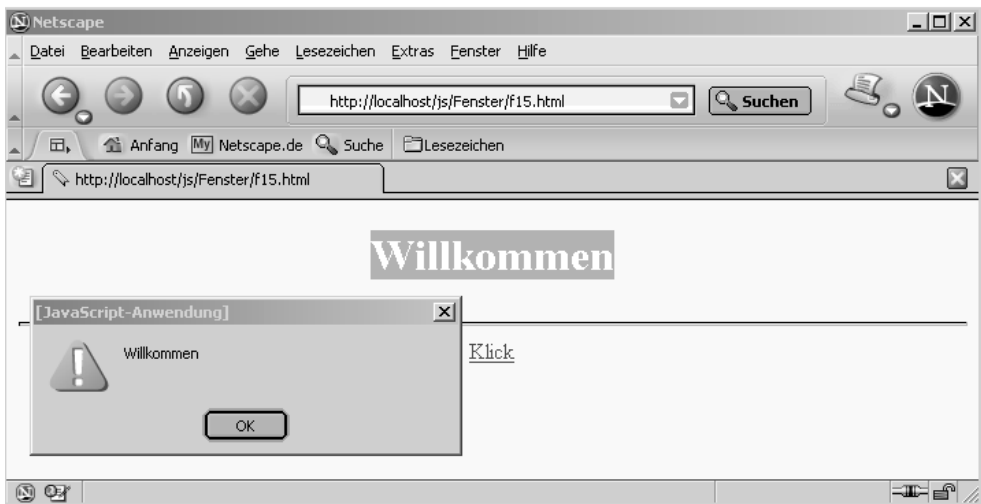


Abbildung 360: Auf die Selektion in der Webseite wird mit JavaScript zugegriffen.

Achtung

Die Methode wird nur sehr eingeschränkt von Browsern unterstützt. Der Internet Explorer unterstützt die Methode beispielsweise nicht!

201 Wie kann ich ein einfaches Mitteilungsfenster anzeigen?

Das `window`-Objekt besitzt eine einfache Methode mit Namen `alert()`, mit der Sie ein Anzeigenfenster mit der in den Klammern als Parameter spezifizierten Nachricht auflenden.

Syntax:

```
alert([Anzeigetext])
```

Listing 517: Syntax eines alert()-Fensters

Die auszugebende Meldung muss als Parameter in Hochkomma gesetzt werden (also ein String). 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 518: Ein einfaches Mitteilungsfenster

In Zeile 4 wird das einfache Mitteilungsfenster angezeigt.



Abbildung 361: Die Darstellung des alert()-Fensters im Internet Explorer

Hinweis

Die Darstellung eines `alert()`-Fensters wird sich je nach verwendetem Browser unterscheiden. Auf jeden Fall 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 potenziell 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 719) bietet Ihnen dazu alle Möglichkeiten, die Sie benötigen.

202 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 519: Syntax eines `confirm()`-Fensters

Es gibt zwei Schaltflächen (`Ok` und `Abbrechen`), 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 (also ein String). Rückgabewerte von eventuell dort notierten Funktionen oder Methoden werden als String angezeigt.

Beispiel (*f2.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   confirm("Alles klar?");
05 </script>
06 </body>
07 </html>
```

Listing 520: Ein einfaches Dialogfenster mit zwei Schaltflächen

In Zeile 4 wird das einfache Dialogfenster mit zwei Schaltflächen angezeigt.



Abbildung 362: Das `confirm()`-Fenster

Hinweis

Es ist in JavaScript nicht möglich, dem `confirm()`-Fenster Parameter mitzugeben, um das Aussehen oder Verhalten zu manipulieren. Das ist keine Schwäche von JavaScript, denn wenn Sie so etwas benötigen, verwenden Sie einfach ein eigenes Browserfenster, in dem Sie das entsprechende Layout und ein Formular anzeigen. Die `open()`-Methode (siehe das Rezept » auf Seite) bietet alle Möglichkeiten, die Sie benötigen.

203 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 521: Syntax einer `prompt()`-Eingabeaufforderung

Als ersten Parameter geben Sie das Label des Eingabefeldes an. Der zweite Parameter gibt optional einen Vorbelegungswert an.

Tip

Obwohl der zweite Parameter optional ist, ist es nicht sinnvoll, ihn wegzulassen. In dem Fall wird im Eingabefeld für den Anwender der – zwar wohl definierte, aber dennoch sehr störende – Wert `undefined` zu sehen sein.



Abbildung 363: Diese Vorbelegung ist sehr störend für einen Anwender.

Besser wählen Sie eine Leerstring `""` als zweiten Parameter, wenn Sie keine Vorbelegung des Eingabefeldes wünschen.

Die Methode fordert den Anwender in einem Dialogfenster mit einer `OK`- und einer `Abbrechen`-Schaltfläche zu einer Eingabe in einem Feld auf. 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 wohl definierten Wert `null`.

Beispiel (*f3.html*):

```
01 <html>
02 <body>
03 <script language="JavaScript">
04   prompt("Ihr Name", "");
05 </script>
```

Listing 522: Ein einfaches Fenster zur Entgegennahme von Benutzereingaben

```
06 </body>
07 </html>
```

Listing 522: Ein einfaches Fenster zur Entgegennahme von Benutzereingaben (Forts.)

In Zeile 4 wird das einfache Fenster zur Entgegennahme von Benutzereingaben angezeigt.



Abbildung 364: Die Entgegennahme der Benutzereingabe

Hinweis

Die Verwendung der `prompt()`-Methode zur Entgegennahme der Benutzereingabe ist heutzutage nicht mehr zeitgemäß. Grundsätzlich verwendet man dazu ein Webformular.

204 Wie kann ich die Startseite des Browsers aufrufen?

In jedem Webbrowser können Sie eine Startseite einstellen. Das kann eine leere Seite oder ein beliebiger 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 523: Ein direktes Laden der voreingestellten Standardseite

In Zeile 4 wird beim Laden der Webseite automatisch die voreingestellte Standardseite des Webbrowsers geladen.

Beim Internet Explorer funktioniert die Methode nicht. Er meldet beim Aufruf einen Fehler.



Abbildung 365: Der Internet Explorer meldet bei der Verwendung von `home()` einen Fehler.

Allerdings ist die Bedeutung der Methode für die Praxis auch nahezu nicht vorhanden.

205 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 524: Zugriff auf die Statuszeile des Webrowsers

In Zeile 7 wird der Wert der Eigenschaft `status` gesetzt.



Abbildung 366: Anzeige von Informationen in der Statuszeile

Hinweis

Die Statuszeile ist die 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 den URL des Verweises dort anzeigen. Wenn Sie gleichzeitig eine Meldung über die `status`-Eigenschaft dort 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 Standardanzeige in der Statuszeile des Browsers ermitteln oder festlegen?« Sie sollten auch beachten, dass der Netscape Navigator 7.x beim Schreiben in die Statuszeile einige ungewöhnliche Probleme hat. Insbesondere, wenn Sie zeitverzögert mit der Methode `setTimeout()` dort schreiben wollen.

206 Wie kann ich Standardanzeigen 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 den URL des Verweises dort 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 (*f5a.html*):

```
01 <html>
02 <script language="JavaScript">
```

Listing 525: Setzen der Defaulteigenschaft der Statuszeile


```

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 525: Setzen der Defaulteigenschaft der Statuszeile (Forts.)

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.



Abbildung 367: Die Statuszeile zeigt den URL des Hyperlinks an.



Abbildung 368: Der Wert in der Statuszeile wird auf den Defaultwert zurückgesetzt, sobald der Mauszeiger den Bereich des Hyperlinks verlässt.

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.

207 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 Beispiel. Die Statuszeile wird für eine Laufschrift genutzt, die von links nach rechts beliebigen Text Zeichen für Zeichen auffrollt. Wird der Text vollständig angezeigt, wird die Statuszeile gelöscht und es geht wieder von vorne los.

Beispiel (*f5b.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 526: Die Statuszeile wird dynamisch geschrieben.

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 gewartet 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 Stringvariablen (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 369: 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.

208 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>
```

Listing 527: Das Ausdrucken der Webseite aus der Seite heraus mit JavaScript

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

```

05 </div>
06 <a href="javascript:print()">Drucke die Webseite</a>
07 </body>
08 </html>

```

Listing 527: Das Ausdrucken der Webseite aus der Seite heraus mit JavaScript (Forts.)

In Zeile 6 wird mit einer Inline-Referenz bei einem Hyperlink die `print()`-Methode aufgerufen.

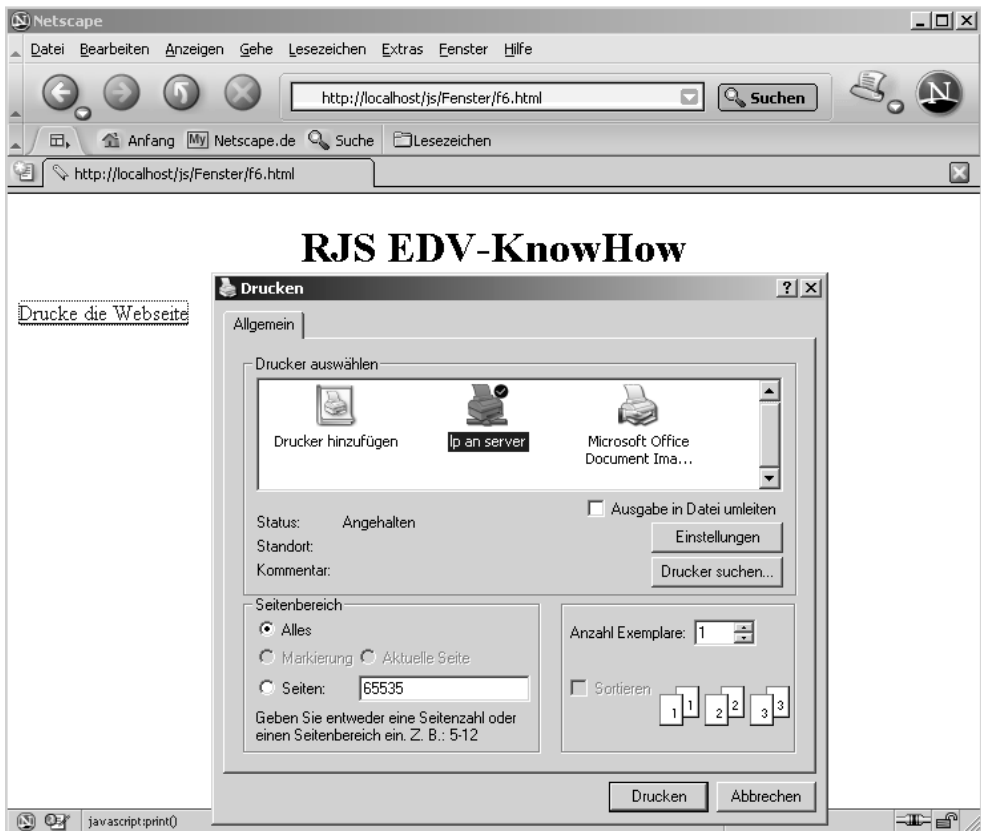


Abbildung 370: Aufruf des Druckdialogs des Webbrowsers

209 Wie kann ich den Ladevorgang einer Webseite abbrechen?

Das `window`-Objekt besitzt die einfache, parameterlose Methode `stop()`, um darüber den Ladevorgang einer Seite abubrechen. 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></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 528: 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 zwei Sekunden die `stop()`-Methode aufgerufen. Anschließend wird über das Setzen der Eigenschaft `location.href` eine neue Webseite aufgerufen. Sollte die Seite nach zwei Sekunden noch nicht geladen worden sein, bricht die `stop()`-Methode den Ladevorgang ab.

Achtung

Beim Internet Explorer funktioniert die Methode nicht. Er meldet beim Aufruf einen Fehler.



Abbildung 371: Der Internet Explorer meldet bei der Verwendung von `stop()` einen Fehler.

Allerdings ist die Bedeutung der Methode für die Praxis auch gering.

210 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 529: 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 530: 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.

Tip

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 715).

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 gleichen¹ (!) 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 aber 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 zeitverzögert in die Seite mehrfach neu schreiben wollen. In dem 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.

1. Zumindest unter der Voraussetzung, dass die entsprechenden Skriptanweisungen zeitgleich dem Interpreter bereitstehen.

211 Wie kann ich einen Aufruf von `setTimeout()` anhalten?

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 (siehe das Rezept »Wie kann ich eine Aktion zeitverzögert aufrufen?« auf Seite 714). Sofern der Rückgabewert von `setTimeout()` einer Variablen 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 531: 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 }
```

Listing 532: Abfangen der Aktion, die mit `setTimeout()` aufgerufen wurde

```

08 function stop() {
09   clearTimeout(aufruf);
10 }
11 //-->
12 </script>
13 <noscript></noscript>
14 <body>
15 <div align="center">
16 <h1>Willkommen auf der Webseite</h1>
17 Sie werden gleich weitergeleitet.
18 <br>
19 <a href='javascript:weiter()'>Weiter</a>
20 <br>
21 <a href='javascript:stop()'>Abbruch</a>
22 </div>
23 </body>
24 </html>

```

Listing 532: Abfangen der Aktion, die mit `setTimeout()` aufgerufen wurde (Forts.)

Mit dem Hyperlink in Zeile 19 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. Die Rückgabe 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 21 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.

212 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 verfügbar⁷. Es repräsentiert den kompletten URL der gerade in einem Browser angezeigten Webseite. Jede der Eigenschaften von dem `location`-Objekt steht für einen anderen Bestandteil des URL. Wenn man die allgemeine Struktur eines URL zugrunde legt, sieht er so aus:

```
protocol//host:port/pathname#hash?search
```

Listing 533: Die allgemeine Struktur eines URLs

7. Sie brauchen es also nicht extra zu erzeugen.

Etwa derart:

```
http://www.webscripting.de:123/abc.html#Anfang?x=7&y=2
```

Listing 534: Ein typischer URL in vollständiger Ausprägung

Jeder der einzelnen Bestandteile des URL steht als Bezeichner einer Eigenschaft des `location`-Objekts zur Verfügung und kann direkt verwendet werden.

Eigenschaft	Beschreibung
<code>protocol</code>	Das den 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 <code>#</code> (nur bei HTTP-URLs).
<code>search</code>	Suchabfragen inklusive dem Trennzeichen <code>?</code> (nur bei HTTP-URLs). Der Suchstring besteht aus Variablen und Wertepaaren, jedes Paar durch <code>&</code> getrennt.
<code>href</code>	Die vollständige Repräsentation eines URL.
<code>hostname</code>	Die Repräsentation des kontaktierten Host samt Port.

Tabelle 11: 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 +
10         "\nsearch: " + location.search +
11         "\nhref: " + location.href +
12         "\nhostname: " + location.hostname);
13 }
14 //-->
15 </script>
16 <noscript></noscript>
17 <body>
18 <a href="javascript:wasIstInLocation()">Klicke</a>
19 </body>
20 </html>
```

Listing 535: Zugriff auf `window.location`

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.

213 Wie kann ich mit JavaScript auf eine Seite weiterleiten?

Die Eigenschaft `href` von `location` kann hervorragend verwendet werden, um mit JavaScript eine neue Seite im aktuellen Browserfenster aufzurufen. Sie müssen bloß einen neuen 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 536: Weiterleitung mit `location.href`

Beim Laden der Webseite sorgt Zeile 4 dafür, dass nach zwei Sekunden die neue Datei geladen wird.

214 Wie kann ich eine aktuelle Webseite neu laden?

Das `location`-Objekt verfügt über eine Methode mit Namen `reload()`. Mit deren Aufruf erzwingen Sie ein Reload 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">
```

Listing 537: Das Neuladen der aktuellen Webseite

```

11 <!--
12   document.write(new Date() + "<br>");
13 //-->
14 </script>
15 <a href="javascript:neuLaden()">Neu</a>
16 </body>
17 </html>

```

Listing 537: Das Neuladen der aktuellen Webseite (Forts.)

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 hervorragend dazu, in Verbindung mit `setTimeout()` 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.

215 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 538: 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.

Der 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, welche 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 wird⁸.

Der Fenstername ist ein weitgehend frei wählbarer Variablenbezeichner, der den üblichen Regeln für Variablennamen genügen muss und nicht mit einigen reservierten Token (`window`,

8. Was aber nicht bei jedem Browser funktioniert.

self, parent, top) übereinstimmen darf, die eine feste Bedeutung haben (siehe dazu das Rezept »Wie kann ich in ein Dokument schreiben?« auf Seite 671). 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 Fenster Namens das Rezept »Wie kann ich den Namen eines Fensters bestimmen?« auf Seite 699).

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.

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 Browsern 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 539: 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 nach einer

Tipp

Wenn Sie die `open()`-Methode gänzlich ohne Parameter angeben, wird von dem meisten Browsern ein leeres Browserfenster geöffnet.

Achtung

Ein Einzeiler wie folgender kann einen Computer vollständig blockieren:

```
while(1) open();
```

Listing 540: Eine hochgefährliche Endlosschleife

Mit diesem Einzeiler öffnen Sie in einer Endlosschleife immer neue Browserfenster. Der Anwender hat keine Chance, die einzelnen Browserfenster auch nur annähernd so schnell zu schließen, wie neue Browserinstanzen erstellt 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 541: 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.

216 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 719). Sofern Sie dabei keine Optionen angeben, wird die Größe des Folgefensters als auch die Position weitgehend zufällig (genau genommen nach den Vorgaben der aufrufenden Konstellation) gewählt. Dieses Verhalten kann natürlich individuell angepasst werden, wozu im Wesentlichen die Optionen der `open()`-Methode genutzt werden können.

Man kann sowohl Parameter beim Aufruf einer Fensterinstanz verwenden als auch bei bereits erzeugtem Fenster neue Werte den 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 <code>yes</code>).
<code>scrollbars=yes/no</code>	Die jeweiligen Optionen legen jeweils fest, ob das Fenster eine fenster-eigene 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 <code>no</code> .
<code>toolbar=yes/no</code>	
<code>status=yes/no</code>	
<code>menubar=yes/no</code>	
<code>location=yes/no</code>	

Tabelle 12: Die möglichen Optionen von `open()`

Option	Beschreibung
directories=yes/no	Die Option legt fest, ob das Fenster über fenstereigene Directory-Buttons (Netscape) verfügt.

Tabelle 12: Die möglichen Optionen von `open()` (Forts.)

Schauen wir uns ein Beispiel für den Einsatz der Optionen von `open()` mit Optionen 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, directories=yes,
menubar=yes, location=yes");
06 }
07 function f2() {
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 542: Einsatz von `open()` in Verbindung mit Optionen

Das Beispiel ruft in den Zeile 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");`).

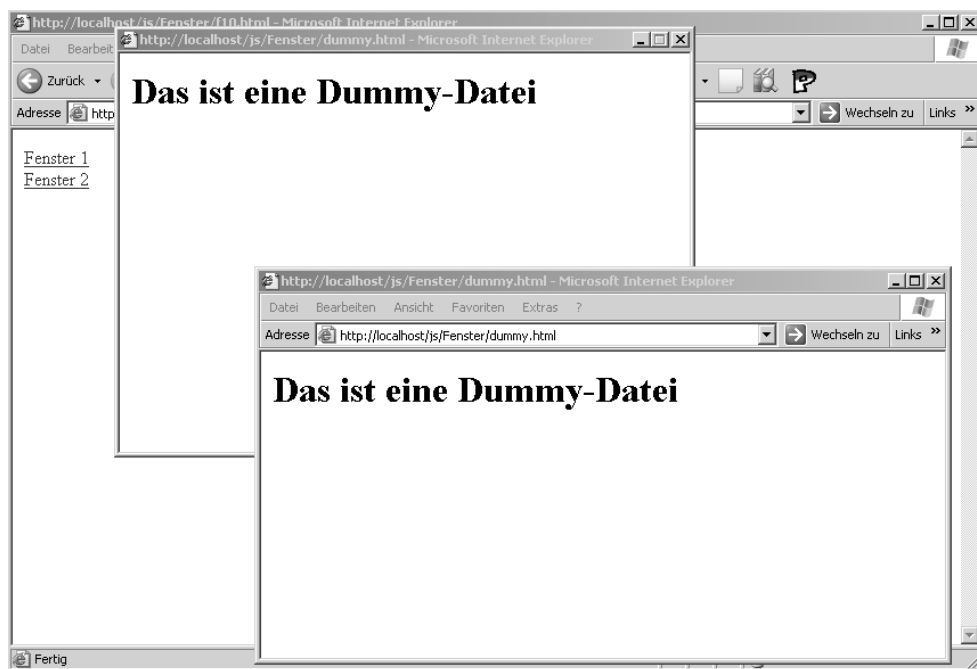


Abbildung 372: Die beiden neuen Fenster unterscheiden sich offensichtlich – dem hinteren Fenster fehlen die Adresszeile, das Menü, Symbolleiste, und der Button für das Vollbild ist deaktiviert (es lässt sich überhaupt nicht in der Größe verändern).

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 noch viel inkonsistenter von den Browsern 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, welche den inneren Anzeigebereich aufgrund optionaler Statuszeile und Menüleiste nicht genau festlegt). Besonders wichtig ist jedoch, dass über `screenX=(Pixel)` und `screenY=(Pixel)` die linke obere Ecke des Browserfensters festgelegt werden kann. Der Navigator kann 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, dependent=yes,
hotkeys=yes, screenX=0, screenY=200");
06 }
07 function f2() {
08   open("dummy.html", "f2", "innerWidth=200, innerHeight=300, dependent=no,
hotkeys=no, screenX=100, screenY=0");
09 }
10 //-->
11 </script>
12 <body>
13 <a href="JavaScript:f1()">Fenster 1</a><br>
14 <a href="JavaScript:f2()">Fenster 2</a>
15 </body>
16 </html>
```

Listing 543: Die erweiterten Optionen von open()

Testen Sie das Beispiel in verschiedenen Browsern. Sie werden schnell erkennen, dass es fast einem Glücksspiel gleicht, wie die verschiedenen Browser reagieren (vor allem ältere).

217 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 vom `window`-Objekt zu versuchen. Dabei wendet man auch sinnvollerweise das Schlüsselwort `self` an, um auch die Position des Hauptfensters vom Browser 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 können Sie `moveBy([horizontal], [vertikal])` verwenden, um damit ein Fenster um die angegebenen Pixel in horizontale oder vertikale Richtung zu verschieben.

Beispiel (*verschieb1.html*):

```
01 <html>
02 <script language="JavaScript">
03 <!--
04   function f1() {
```

Listing 544: Positionierung von Browserfenstern


```

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 
```

```
09  //-->
10  </script>
11  <body>
12  <a href="JavaScript:f1()">Fenster 1</a><br>
13  </body>
14  </html>
```

Listing 545: Öffnen einer leeren Seite, verschieben und dann erst die richtige Datei laden

218 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 vom `window`-Objekt 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 von einem Fenster um die angegebenen Pixel in horizontale oder vertikale Richtung, und `resizeTo([Breite], [Höhe])` setzt die Größe von einem 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 546: Die Anwendung von Größenveränderung von Browserfenstern

Das Beispiel ruft in den Zeilen 18 bis 20 mit Inline-Referenzen bei drei Hyperlinks die Funktionen `f1()`, `f2()` und `f2()` 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 von einem Browserfenster macht in einigen Browsern Schwierigkeiten, wenn Sie während des Ladens einer Webseite versuchen, das Fenster anzupassen. Wenn also das Laden der Webseite noch nicht abgeschlossen ist, dann 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. Dann erst 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 547: Öffnen einer leeren Seite, Anpassen der Größe und dann erst die richtige Datei laden

219 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 von einem Fenster um die angegebenen Pixel in horizontale oder vertikale Richtung, und mit `scrollTo([x-Koordinate], [y-Koordinate])` verschieben Sie den Inhalt von einem Fenster auf die angegebene x-/y-Position. Nur leider unterstützt so gut wie kein Browser die Methoden.

220 Wie kann ich ein Browserfenster schließen?

Allgemein erfolgt das Schließen von Fensterinstanzen aus der Webseite heraus über die `window`-Methode `close()`, welche 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 ist dies für das Hauptfenster des Browsers erst einmal ein Problem. Es gibt aber die reservierten Fensternamen `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", "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 548: 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 Einstellungen von dem Browser nicht zu lasch sind (was aber bei vielen Browsern der Fall ist).

221 Wie kann ich überprüfen, ob ein zuvor geöffnetes Browserfenster noch offen ist?

Zum Öffnen eines Browserfensters mit JavaScript kommt in der Regel die `open()`-Methode des `window`-Objekts zum Einsatz. 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 549: 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 nicht⁹, 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.

9. Achten Sie auf das Ausrufezeichen.

222 Wie kann ich 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 des URL. Über das `history`-Objekt – ein direktes Unterobjekt von `window` – hat man auch aus JavaScript heraus Zugriff auf die zuvor geladenen Seiten im Browser des Anwenders. Maßgeblich ist dabei die Liste, wie sie in der History-Liste des Browsers gespeichert ist. Eine Instanz des Objektes `history` steht automatisch bereit, und sämtliche Eigenschaften und Methoden lassen sich für nachfolgende Syntax direkt verwenden:

```
history.[Eigenschaft/Methode]
```

Listing 550: Zugriff auf eine Eigenschaft oder Methode des history-Objekts

223 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 History-Liste 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 551: 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 373: Sieben Einträge sind in der History.

Hinweis

Rein von der Theorie her stellt das `history`-Objekt noch die Eigenschaften `current` (aktuelle Seite in der Historie), `next` (nächste) und `previous` (vorherige) bereit, aber diese werden von nahezu allen Browsern nicht unterstützt.

224 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 552: Return to sender – via der 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 gigantisch – Sie müssen nicht wissen, von wo der Besucher kam. Die Syntax ist ohne Änderungen (höchstens vielleicht der Text) universell für alle Situationen einsetzbar, wo 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()` bereit. 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 History-Liste. Sie übergeben der Methode als Parameter die Seitenzahl. Bezugspunkt ist dabei die aktuelle Seite. Sie können die Anzahl mit Vorzeichen verwenden und damit also sowohl zurück als auch vorwärts gehen, wenn Sie vorher zurückgegangen sind.

225 Wie kann ich mit Cookies arbeiten?

Bei einem Cookie handelt es sich immer um eine kleine Textdatei, welche von einem Browser in ein, im Browser vorgegebenes, Verzeichnis geschrieben und von dort wieder ausgelesen werden kann. Dieser Vorgang kann sowohl vom Client als auch dem Server initiiert werden. Auf der einen Seite muss man beachten, dass die verschiedenen Browser zwar einen etwas unterschiedlichen Weg bzgl. der konkret angelegten Dateien gehen. Navigator und Mozilla speichern etwa 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. Dabei können in der Regel maximal 20 Cookies gleichzeitig verwaltet werden. Die gesamte Verwaltung übernimmt die Datei *index.dat*.

Wie auch immer 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 ergänzend 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 beim Auslesen ein Cookie wieder identifiziert. Der zugewiesene Wert ist die Information, welche man eigentlich speichern möchte. Eine wichtige optionale Eigenschaft ist die `expires`-Eigenschaft, mit der ein Gültigkeitsdatum eines Cookies gesetzt werden kann. Das Format sieht beispielsweise so aus:

Wdy, DD-Mon-YY HH:MM:SS GMT

Listing 553: 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 554: Ein Beispiel für das Setzen des Ablaufdatums

Weitere optionale Eigenschaften sind `domain`, worüber der URL der Domain als String spezifiziert werden kann, für welche ein Cookie gültig ist. Wenn die Angabe gesetzt ist, kann ein gesetztes Cookie nur von einem Server aus der Domain wieder ausgelesen werden. Es gibt die Eigenschaft `path` zur Angabe von Verzeichnissen innerhalb dieser Domain (fehlt die Angabe, wird der Pfad des aktuellen Dokumentes 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. So weit ist es also ganz einfach, ein Cookie zu setzen, etwa so:

```
document.cookie = "meinCookie=42";
```

Listing 555: Setzen des Werts eines Cookies

Oder wenn ein Haltbarkeitsdatum gesetzt werden soll, geht das so:

```
document.cookie = "meinCookie=42; expires=Wednesday, 09-Nov-04 20:15:40 GMT";
```

Listing 556: Setzen des Werts 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 erst einmal keine große Unterstützung. Cookie-Werte dürfen etwa keine Leerzeichen und keine 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 ein Programmierer namens Bill Dortch entwickelt hat und unter <http://www.hidaho.com/cookies> kostenlos bereitgestellt werden. 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 unverändert im nachfolgenden Beispiel einfach nutzen). Die zusätzliche Funktion `DeleteCookie(name,path,domain)` erlaubt das Löschen eines Cookies, indem das Verfallsdatum auf den 1. Januar 1970 gesetzt wird (es gibt keine Möglichkeit zum direkten Löschen).

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

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 von dem Formular 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;
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(2004,11,31,1,0,0);
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" onSubmit="schreibeCookie()">
44       Geben Sie Ihren Namen ein: <input name="id"><br>

```

Listing 557: Das Setzen und Auslesen von Cookies mit JavaScript

```

45     Geben Sie Ihren Kommentar ein: <input name="komment"><br>
46     <input type="Submit" value="Ok">
47     <input type="reset" value="Abbruch">
48 </form>
49 </body>
50 </html>

```

Listing 557: Das Setzen und Auslesen von Cookies mit JavaScript (Forts.)



Abbildung 374: Das Formular

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 von dem ersten Eingabefeld 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 von dem Formular 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 wird ins Leere laufen. Die Wertzuweisung wäre sinnlos und würde den Anwender nur verwirren.



Abbildung 375: Das Cookies-Verzeichnis beim Internet Explorer

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%FCffe\
localhost/js/Fenster/
1088
2881011712
29683403
4152842240
29658185
*
```

Listing 558: Der Inhalt einer Cookie-Datei beim Internet Explorer

226 Wie kann ich überprüfen, ob ein Anwender die Verwendung von Cookies aktiviert hat?

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 am besten ein Dummy-Cookie und lesen den geschriebenen Wert unmittelbar wieder ein. Klappt dies, können Sie Cookies verwenden.

227 Wie kann ich den Wert einer Variablen zu einer anderen Webseite weitergeben?

Sobald Sie in einem JavaScript einen Variablenwert dynamisch entgegennehmen (etwa eine Benutzereingabe), ist dieser verloren, wenn 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.
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 732*).
3. Sie weisen den Wert einer Variablen 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 den URL der neuen Seite.

Wir demonstrieren hier die vierte Variante. Sie können aus JavaScript heraus über das `location`-Objekt auf die Adresszeile des Webbrowsers zugreifen. Jede der Eigenschaften von dem `location`-Objekt steht für einen anderen Bestandteil des URL (*siehe dazu das Rezept »Wie*

kann ich auf die Adresszeile des Webbrowsers zugreifen?» auf Seite 716). Wenn man die allgemeine Struktur eines URL zugrunde legt, sieht er so aus:

```
protocol//host:port/pathname#hash?search
```

Listing 559: Ein typischer 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 Wertepaaren, jedes Paar durch `&` getrennt.

Angenommen, Sie wollen nun die Variable `a` mit dem Wert `42` an die Datei `zwei.html` übergeben, müssen Sie nur einen Link (hier relativ) so zusammensetzen:

```
zwei.html?a=42
```

Listing 560: Das Anhängen von einer Variablen an den URL

Dieses Zusammensetzen des 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 einem 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 des Pseudo-URL gar nicht zu machen. Verwenden Sie einfach versteckte Formularfelder, die mit den Variablenwerten gefüllt werden, und rufen Sie eine Folgeseite mit der `submit()`-Methode und der Versandmethode `get` sowie der Angabe der Zieldatei 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.fl.a.value = a;
08     window. document.fl.b.value = b;
09     window. document.fl.submit();
10   }
11 //-->
12 </script>
13 <body>
14 <form name="fl" action="vari2.html">
15 <input type="hidden" name="a">
16 <input type="hidden" name="b">
17 </form>
```

Listing 561: Übergabe von zwei Variablenwerten an eine neue Seite mit versteckten Feldern

```

18 <a href="javascript:neueSeite()">Neue Seite</a>
19 </body>
20 </html>

```

Listing 561: Übergabe von zwei Variablenwerten an eine neue Seite mit versteckten Feldern (Forts.)

In dem Skript werden zwei Variablen mit den Namen `a` und `b` mit Werten belegt (Zeile 4 und 5)¹⁰. 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 die Namen der Formularfelder wie 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 von dem Formular – 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 die Daten des Formulars dann abgeschickt (`window.document.f1.submit()`;). Damit wird die über `action` angegebene Seite in den Webbrowser geladen.

Schauen wir uns diese Seite an, in der die Daten wieder ausgelesen werden.

Beispiel (*vari2.html*):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04   function werte(){
05     retur = new Array();
06     document.write("Das steht in der search-Eigenschaft: " + location.search +
07     "<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 + "]: " + zerlegt2[i] + "<br>");
13       zerlegt3 = zerlegt2[i].split("=");
14       document.write("Das steht in zerlegt3[0]: " + zerlegt3[0] + "<br>");
15       document.write("Das steht in zerlegt3[1]: " + zerlegt3[1] + "<br>");
16       document.write("Das ergibt unescape(zerlegt3[1]): " + unescape(zerlegt3[1]) +
17       "<br>");
18       retur[2*i] = zerlegt3[0];
19       retur[2*i + 1] = unescape(zerlegt3[1]);
20     }

```

Listing 562: Die Entgegennahme von Variablenwerten

10. Aus Gründen der Einfachheit hartkodiert.

```

19  document.write("<hr>");
20  return retur;
21  }
22  //-->
23 </script>
24 <body>
25 <script language="JavaScript">
26 <!--
27  wertepaar = werte();
28  for(i = 0; i < wertepaar.length; i++){
29      document.write(wertepaar[i] + "<br>");
30  }
31  //-->
32 </script>
33 <noscript></noscript>
34 </body>
35 </html>

```

Listing 562: Die Entgegennahme von Variablenwerten (Forts.)

Die zentrale Funktionalität, um die Werte der Variablen wieder aus dem 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 sein¹¹. 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 + "<br>");`). 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 (`document.write("Beseitigen des ?: " + zerlegt1 + "<br>");`). 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 Variablennamen 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]);`), womit 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 werden zwar nur der jeweilige Variablenname und der zugehörige Wert ausgegeben (Zeile 29 – `docu-`

11. Bei einer praktischen Anwendung kommentieren Sie diese aus oder löschen sie.

ment.write(wertepaar[i] + "
");), aber Sie erkennen, wie Sie an die Variablenbezeichner und die jeweiligen Werte durch Aufruf der Funktion `werte()` kommen.

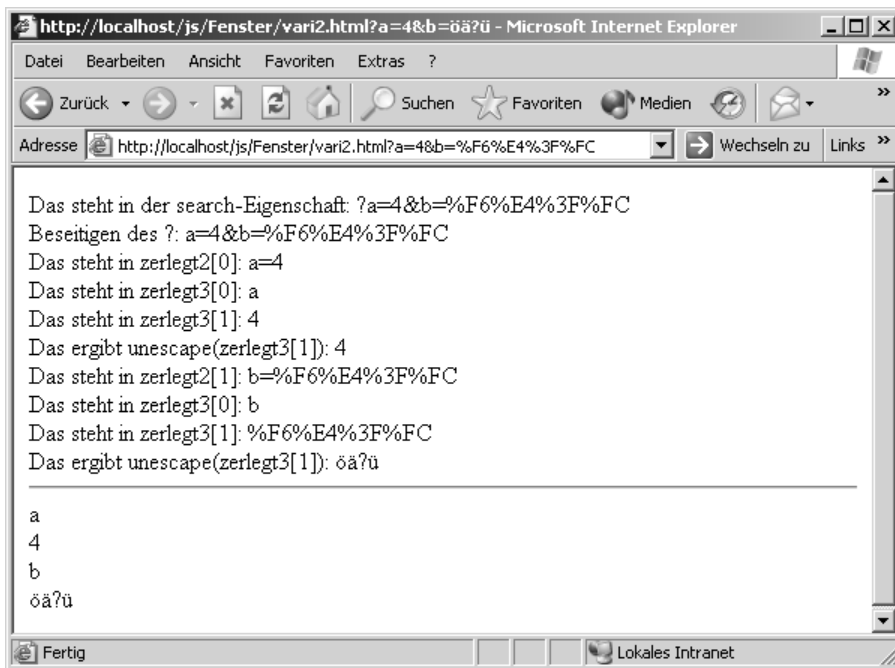


Abbildung 376: Die übergebenen Variablen sind in der Adresszeile und in den einzelnen Ausgaben in der Webseite zu sehen.

228 Wie kann ich sicherstellen, dass eine Webseite nur mit Zugangsbeschränkung aufgerufen werden kann?

Im Kapitel zu den Formularen finden Sie ein Rezept »Wie kann ich sicherstellen, dass ein Formular nur plausibilisiert abgeschickt wird?«. Darin finden Sie bereits diverse Ausführungen zum Thema. Eine sehr zuverlässige 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 unbrauchbar, wenn der Name der Datei einmal bekannt ist.

Beispiel (pw1.html):

```

01 <html>
02 <body>
03 <script Language="JavaScript">
04 <!--

```

Listing 563: Eingabe des Dateinamens als Passwort


```
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">
13     <br>
14     <input type="button" value="OK" onClick="ladeSeite()">
15   </form>
16 </body>
17 </html>
```

Listing 563: Eingabe des Dateinamens als Passwort (Forts.)

In Zeile 12 geben Sie ein Passwort ein. In Zeile 14 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.

229 Wie kann ich mit Frames umgehen?

Frames bedeuten die Aufteilung des Anzeigebereichs eines Browsers in einzelne Segmente, die unabhängig mit Inhalt gefüllt werden können. Die Frametechnologie beinhaltet zwei Facetten. Es gibt einmal die Frames selbst, welche 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, welche in den Browser geladen wird. Sie enthält nur Informationen über Name, Größe und Position der einzelnen im Anzeigenfenster enthaltenen Frames. Insbesondere ist der eigentliche Inhalt der Frames nicht in der Frameset-Datei notiert.

Eine HTML-Datei, welche 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 er nicht verboten ist)¹². Dessen Funktion wird von dem `<frameset>`-Container übernommen. Bei der Definition der Frameset-Datei müssen genaue Angaben darüber gemacht werden, wie das Anzeigenfenster aufgeteilt werden soll und welche Dateien konkret in ein einzelnes Frame zu laden sind.

Es müssen bei der Angabe 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 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 der 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]"`.

12. Das betrifft HTML, nicht XHTML.

Die Prozentangaben hinter dem Gleichheitszeichen sollten in Anführungszeichen stehen und definieren die Verhältnisse der einzelnen Frames zueinander – nicht einen Faktor von der Größe des Gesamtanzeigefenster vom Browser! 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%"`.

Sie können natürlich mehr wie zwei Spalten oder Zeilen angeben. Dabei können Sie anstelle von Prozentangaben für die Größe von Frames Zahlenwerte benutzen, mit denen Sie tatsächliche Größen der Frames in Pixel definieren. Dies erfolgt über die folgende Angabe:

```
rows="[Angabe in Pixel]"
```

Listing 564: Die Größe der Zeile in Pixel

bzw.

```
cols= "[Angabe in Pixel]")
```

Listing 565: Die Größe der Spalte in Pixel

Diese Größenangaben werden allerdings durch die Größe des Browser-Anzeigefensters 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, variablen Angaben, verwendet werden. In Verbindung mit Jokerzeichen (der Stern, wie er auch sonst als Jokerzeichen üblich ist) bewirkt zum Beispiel mit `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 von Prozentangaben durch das Jokerzeichen verändern. Es ist eigentlich unsinnig, hier mit Joker zu arbeiten.

Wenn also über den `<frameset>`-Tag und seine Parameter ein äußeres Gerüst für eine Framestruktur aufgebaut ist, wird im Inneren für jede dort definierte Zeile oder jede Spalte ein einzelnes Frame angegeben. Der Inhalt der Frames wird mit der Syntax `<frame src= "[URL]" name="[Name]">` spezifiziert. Über die `src`-Angabe wird der URL des Inhalts von dem jeweiligen Frames 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 der 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 für Frames sind 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
_self	Referenz auf das aktuelle Frame, wo eine Seite gerade dargestellt wird (Standardsituation).
_parent	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).
_blank	Referenz auf ein neues Browserfenster ohne Framestruktur.
_top	Referenz auf das volle Browserfenster. Die Framestruktur verschwindet.

Tabelle 13: Reservierte Frame-Namen

Ein typisches Grundgerüst einer Framestruktur sieht von der Theorie her so aus:

```
<html>
<frameset>
    ...Frames
</frameset>
<noframes>
    <body>
        ... Alternativinformationen
    </body>
</noframes>
</html>
```

Listing 566: 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 aufgrund des Prinzips der Fehlertoleranz, sondern weil er programmiert ist, diesen Container nicht anzuzeigen. Ein nicht-frame-fähiger Browser fängt hinter der `<noframes>` 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>`-Tag, es kann aber auch nur Klartext sein.

Ein Frame muss nicht von vornherein etwas anzeigen. Es kann leer bleiben, bis dort Inhalt benötigt wird. Verzichtet man auf die Angabe eines Inhaltes über den `<frame>`-Tag (also `src=""` oder das `src`-Attribut ganz weglassen), wird der Raum vom Browser neutral ausgefüllt. Sofern dies für alle Frames geschieht, erkennen Sie unter Umständen überhaupt keine Framestruktur im Browserfenster (bei entsprechender Konfiguration des Browsers – meist sind aber die Trennlinien zu sehen). Allerdings sollten Sie beachten, dass einige Browser Schwierigkeiten machen, wenn Sie das `src`-Attribut ganz weglassen.

Wenn Sie Frames mit Inhalt 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 Frameset-Containers ein darin verschachtelter Frameset-Container definiert wird. Die Verschachtelung erfolgt hierarchisch. Dabei ist offensichtlich, dass nur eine wechselseitige Verschachtelung von Reihen und Spalten Sinn macht.

230 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, wo 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 567: Ein Link in ein anderes Frame

Es können sowohl im Frameset vergebene Namen als auch die reservierten Fensteramen adressiert werden. Sofern der angegebene Zielframe nicht vorhanden ist, wird ein neues Browserfenster (außerhalb der Framestruktur) geöffnet.

Schauen wir uns ein vollständiges Beispiel an. Wir benötigen zuerst die Frameset-Datei.

Beispiel (*fs1.html*):

```
01 <html>
02   <frameset cols="30%,70%">
03     <frame src="verweis.html">
04     <frame src="dummy.html" name="rechts">
05   </frameset>
06 </html>
```

Listing 568: Die Frameset-Datei

In Zeile 3 wird die Datei *verweis.html* referenziert, und die enthält das, worauf es hier ankommt. Hier hinein werden die Hyperlinks notiert, die den in der Frameset-Datei angegebenen Namen verwenden. Die Datei wird im Frameset in der linken Spalte geladen.

Beispiel (*verweis.html*):

```
01 <html>
02 <body>
03   <a href="b1.jpg" target="rechts">Bild 1</a><br>
04   <a href="b2.jpg" target="rechts">Bild 2</a><br>
05   <a href="b3.jpg" target="rechts">Bild 3</a><br>
06 </body>
07 </html>
```

Listing 569: Eine Navigationsdatei mit Referenzen auf das Frame rechts

Die Datei mit den Hyperlinks erlaubt das Laden von jeder der drei Bilddateien in dem rechten Frame, da in der *target*-Angabe dessen Name angegeben wird (Zeilen 3 bis 5).



Abbildung 377: Der Klick auf den ersten Link lädt im rechten Frame eine Bilddatei.



Abbildung 378: Beim Klick auf einen anderen Hyperlink wird im rechten Frame eine andere Datei geladen.

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.

231 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.

Eine Instanz eines `frame`-Objekts können Sie nicht nur über den Index des zugeordneten Objektfeldes ansprechen, sondern natürlich ebenso über den Namen des Frame-Fensters, wie er in der HTML-Framesetseite angegeben wurde (der Parameter `name` im `<frame>`-Tag).

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 570: Zugriff auf eine Eigenschaft oder Methode über das `frames`-Objektfeld

oder

```
[Framename].[Eigenschaft/Methode()]
```

Listing 571: 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 im Falle von Frames in den meisten Fällen dasselbe Frame wie `parent` (außer bei tieferen Verschachtelungen). Bei Verschachtelungseffekten 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.

232 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 das Frameset.

Beispiel (*fs2.html*):

```
01 <html>
02 <frameset cols="30%,70%">
```

Listing 572: Das Frameset

```
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 572: Das Frameset (Forts.)

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() {
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 573: Zugriff auf Frames mit `open()`

In den `open()`-Methoden in den Zeilen 5, 6, 9 und 10 werden die Namen als Ziel angegeben, die im Frameset für das rechte oberere 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) aufgerufen JavaScript-Funktionen immer zwei `open()`-Methoden abgearbeitet werden, werden mit einem Klick durch den Anwender das untere und obere rechte Frame gleichzeitig aktualisiert.

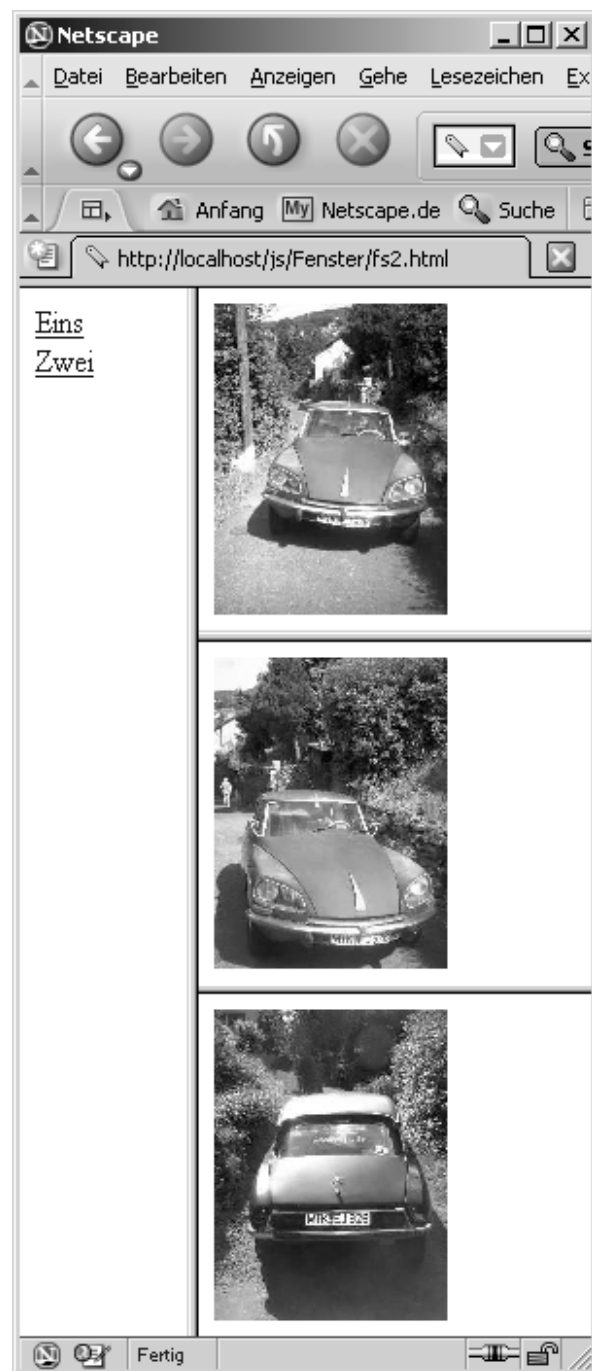


Abbildung 379: Die Anordnung beim Laden des Framesets

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events



Abbildung 380: Umsortierung nach einem Klick

Wie kann ich den Namen eines Frames per JavaScript abfragen?

Natürlich können Sie mit JavaScript die Eigenschaft `name` abfragen. Die Abfrage erfolgt explizit über das Objektfeld `frames`. Hier zuerst die Framesetdatei.

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">
07     <frame src="b2.jpg" name="drei">
08   </frameset>
09 </frameset>
10 </html>
```

Listing 574: Das Frameset

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 Framennamen</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 575: Ausgabe der Framennamen

In den Zeile 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]`).

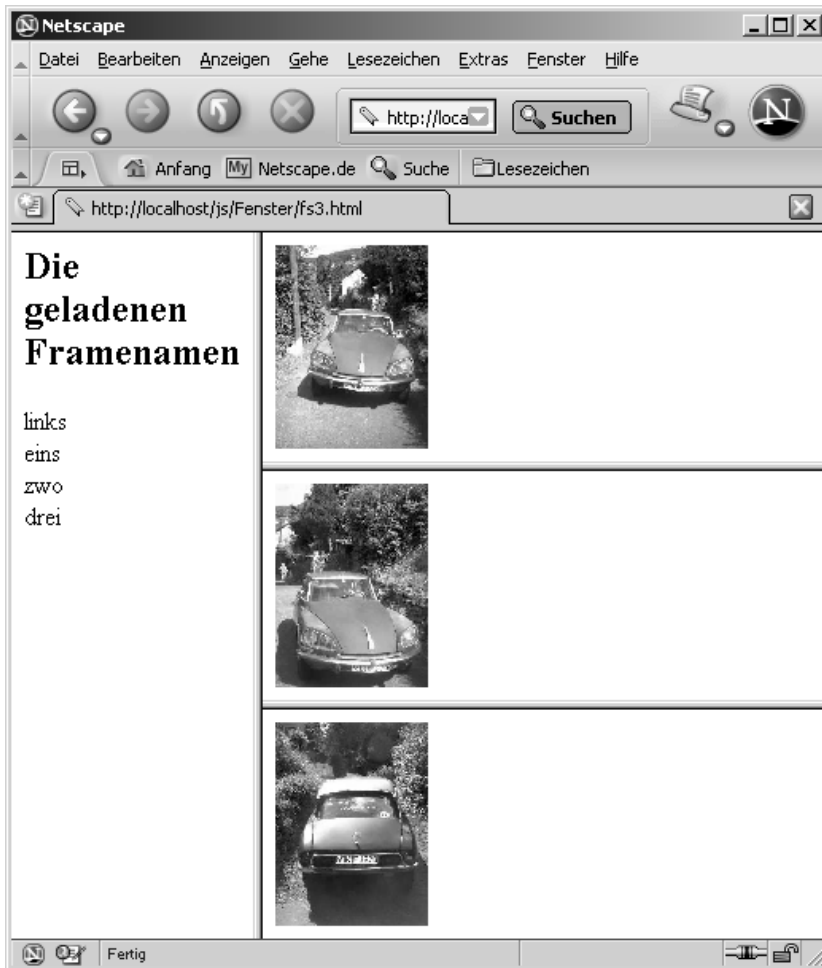


Abbildung 381: Im linken Frame werden die Namen der Frames ausgegeben.

Hinweis

Beachten Sie zum Umgang mit Frames auch das Rezept »Wie kann ich in ein Dokument schreiben?« auf Seite 671 und das Rezept »Wie kann ich ein Dokument explizit öffnen oder schließen?« auf Seite 9. Dort finden Sie unter anderem das Rezept, wie man aus einem Frame heraus direkt in ein anderes Frame schreiben kann.

Hinweis

Sie können den Namen eines Frames auch aus JavaScript festlegen.

233 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() {
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 576: Verhindern, dass eine Seite in einem Frameset angezeigt wird

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.

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

Animation und DHTML

Wir wollen in diesem Kapitel Animationen mit JavaScript bzw. den Begriff DHMTL (Dynamic HTML) im weitesten Sinn behandeln. Im Rahmen des Document Object Model (DOM) wird eine HTML-Seite nicht als statisch aufgebaute, fertige und nicht unterscheidbare Einheit, sondern 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. Dies ist eine Art dynamische Datenbank im Hauptspeicher des Rechners, die 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 für die Seite abgelegt, d.h., in der Denkweise einer relationalen Datenbank in den gleichen Datenbanktabellen eingetragen. Auf diese Weise hat der Browser nach dem Laden der Webseite genaue Kenntnis über alle relevanten Daten sämtlicher eigenständiger Elemente in der Webseite. Welche das jedoch sind und was er damit anstellen kann, das kann sich je nach Browser erheblich unterscheiden. Jedes Element (etwa ein bestimmter HTML-Tag) ist in einem eigenen Datensatz abgespeichert, der bei Bedarf auch während der Lebenszeit der Webseite upgedatet wird, etwa wenn mittels eines Scripts 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.

234 Was versteht man unter Dynamic-HTML?

Dynamic-HTML oder *DHTML* ist ein Begriff, der eng mit dem DOM zusammenhängt. Der Begriff ist jedoch weder eindeutig, standardisiert oder sonst irgendwie geschützt. Insbesondere legen verschiedene Interessengruppen den Begriff unterschiedlich aus. Es gibt überdies verschiedene Schreibweisen, die aber alle im Wesentlichen das Gleiche meinen. 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. Die konkrete Technik, wie die Veränderung dabei realisiert wird, ist irrelevant, obgleich DHTML sehr oft als die Verbindung von HTML, JavaScript und Style Sheets bezeichnet wird. 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, sofern man explizit bestimmte Möglichkeiten ausreizen will.

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 liegende Idee ab dem Internet Explorer 3.0 in Verbindung mit VBScript (wenngleich vollkommen unterschiedlich implementiert). Im ursprünglichen Netscape-Konzept wollte man die Beziehung von einigen Standardobjekten des Browsers zueinander über eine Hierarchie beschreiben. Das anfangs realisierte Konzept erlaubte noch keinen Zugriff auf einzelne Elemente innerhalb einer Webseite oder sonst welchen Komponenten innerhalb eines dieser Standardobjekte. 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 Browserkontrahenten verschiedene Strategien, weshalb es mittlerweile zu zahlreichen Inkompatibilitäten zwischen den Browsern kommt. 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 bleiben noch nicht alle Elemente. In das Konzept von Netscape fielen auch die Layer-Technik, und auch die allgemeine Ereignisbehandlung wird im DOM-Konzept geregelt.

Die Konzeption von Microsoft stellt zur Unterstützung der DOM-Philosophie ab dem Internet Explorer 4.0 u.a. ein eigenes Objekt zur Verfügung (das Objekt `all`), das bereits überall (nicht nur für die Bereiche der Style Sheets) in der Webseite den Zugang auf sämtliche Inhalte der Seite erlaubt. Das DOM-Modell von Microsoft kann seit geraumer Zeit nicht nur über VBScript, sondern auch andere Sprachen wie JavaScript angesprochen werden und beinhaltet zudem – im Gegensatz zu dem Netscape-Modell – eine konkrete und weitgehend vollständige Ereignisbehandlung für sämtliche (!) eigenständige Elemente einer Webseite. Netscape beschränkte die Ereignisbehandlung auf DOM-Ebene wie angedeutet auf wenige ausgewählte Objekte, aber mittlerweile kann im neuen Modell auch im Netscape-Modell auf alle Elemente in HTML-Containern zugegriffen werden. Zum Teil wird sogar das `all`-Objekt unterstützt.

235 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 allerdings nicht zur offiziell unter JavaScript unterstützten Objektmenge und wird hauptsächlich bei Internet Explorer voll unterstützt. Allerdings bieten mittlerweile mehr und mehr Browser – selbst der Netscape Navigator und Mozilla – (teilweise eingeschränkte) Unterstützung. Das `all`-Objekt verfügt über Methoden, mit denen Sie beispielsweise sowohl HTML-Tags in eine Webseite dynamisch einfügen als auch entfernen können. Vor allem von Interesse sind die Eigenschaften:

Eigenschaft	Beschreibung
<code>className</code>	Der Style-Sheet-Klassenname eines Elements. Darüber lässt sich in einigen Browsern aus JavaScript heraus die Verbindung zur Stilklasse 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.
<code>dataFormatAs</code>	Der Datentyp bei Datenanbindung.
<code>dataPageSize</code>	Die Anzahl Datensätze bei Datenanbindung.
<code>dataSrc</code>	Die Datenquelle bei Datenanbindung.
<code>id</code>	Die ID eines Elements.
<code>innerHTML</code>	Der Inhalt eines Elements in HTML-Form.
<code>innerText</code>	Der Inhalt eines Elements in Textform.
<code>isTextEdit</code>	Die Information, ob ein Element veränderbar ist.
<code>lang</code>	Die Sprache eines Elements.
<code>language</code>	Die Skriptsprache für ein Element.
<code>length</code>	Die Anzahl der Elemente in der Webseite.
<code>offsetHeight</code>	Die Höhe eines Elements.
<code>offsetLeft</code>	Der Wert der linken oberen Elementecke.
<code>offsetParent</code>	Die Position des Elternelements.
<code>offsetTop</code>	Der obere Wert der linken oberen Ecke des Elements.
<code>offsetWidth</code>	Die Breite eines Elements.
<code>outerHTML</code>	Der Elementinhalt sowie äußeres HTML.
<code>outerText</code>	Der Elementinhalt sowie äußerer Text.
<code>parentElement</code>	Das Elternelement.
<code>parentTextEdit</code>	Die Editierbarkeit des Elternelements.
<code>recordNumber</code>	Die Datensatznummer bei Datenanbindung.
<code>sourceIndex</code>	Die Angabe der Nummer eines Elements.
<code>tagName</code>	Der HTML-Tag des Elements.
<code>title</code>	Der Titel des Elements.

Tabelle 14: Die Eigenschaften von `all`

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 577: Zugriff auf eine Eigenschaft oder Methode des `all`-Objekts über `tags()`

Beispiel:

```
document.all.tags("h4")[0].outerHTML
```

Listing 578: 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 auch entsprechend dann als ID bezeichnet wird.

Die Syntax des Zugriffs aus JavaScript:

```
document.all.[name].[Eigenschaft/Methode]
```

Listing 579: Zugriff über den Namen

Die Syntax des Kennzeichnens im HTML-Tag:

```
<[HTML-Tag] id="[name]">
```

Listing 580: Die Kennzeichnung des HTML-Tags mit dem id-Parameter

Beispiel für den Zugriff über den Namen bzw. der ID:

```
document.all.abc.lang
```

Listing 581: Ansprechen eines Elements einer Webseite über die ID

Der Tag müsste so aussehen:

```
<h1 id="abc">
```

Listing 582: Die Kennzeichnung in HTML

Schauen wir uns in einem vollständigen Beispiel den Zugriff auf verschiedene Eigenschaften von `all` an.

Beispiel (*d9.html*):

```
01 <html>
02 <head>
03 <title>Ein Demo zu all</title>
04 </head>
05 <script language="JavaScript">
```

Listing 583: Abfrage verschiedener Eigenschaften von all

```

06 <!--
07   function alles() {
08       alert("Die Webseite:\n"
09           + "className: " + document.all.webseite.className + "\n"
10           + "id: " + document.all.webseite.id + "\n"
11           + "innerHTML: " + document.all.webseite.innerHTML + "\n"
12           + "innerText: " + document.all.webseite.innerText + "\n"
13           + "lang: " + document.all.webseite.lang + "\n"
14           + "language: " + document.all.webseite.language + "\n"
15           + "length: " + document.all.webseite.length + "\n"
16           + "offsetHeight: " + document.all.webseite.offsetHeight + "\n"
17           + "offsetLeft: " + document.all.webseite.offsetLeft + "\n"
18           + "outerHTML: " + document.all.webseite.outerHTML + "\n"
19           + "sourceIndex: " + document.all.webseite.sourceIndex + "\n"
20           + "tagName: " + document.all.webseite.tagName + "\n"
21           + "title: " + document.all.webseite.title);
22   }
23 //-->
24 </script>
25 <body onLoad="alles()" id="webseite">
26 <h1>Tom Bombadilis</h1>
27 <h2>Frau Goldbeere</h2>
28 </body>
29 </html>

```

Listing 583: Abfrage verschiedener Eigenschaften von all (Forts.)

In der Webseite finden Sie zwei Überschriften verschiedener Ordnung. Beim Laden der Webseite wird mit dem Eventhandler `onLoad` die Funktion `alles()` aufgerufen. Beim `<body>`-Tag ist die ID mit dem Wert `webseite` spezifiziert, und über diesen Namen greifen wir in der Funktion auf verschiedene Eigenschaften von `document.all` zu und zeigen diese in einem `alert()`-Fenster an.

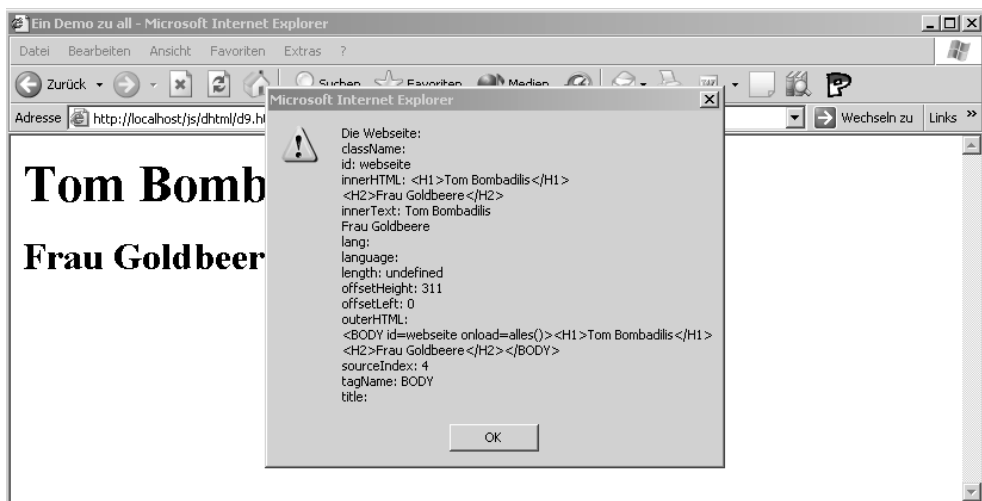


Abbildung 382: Zugriff auf verschiedene Eigenschaften von all über die ID

Einige (nicht alle) der Eigenschaften von `all` können Sie auch setzen, etwa die `title`-Eigenschaft.

Beispiel (*d10.html*):

```

01 <html>
02 <script language="JavaScript">
03 <!--
04   function alles() {
05       document.all.tags("h1")[0].title = "Der Gefährte";
06       status=document.all.tags("h1")[0].title;
07   }
08 //-->
09 </script>
10 <body onLoad="alles()">
11 <h1>Streicher</h1>
12 </body>
13 </html>

```

Listing 584: Setzen der Eigenschaft title

In Zeile 5 wird der Wert der `title`-Eigenschaft gesetzt und in Zeile 6 in der Statuszeile wieder ausgegeben.



Abbildung 383: In der Statuszeile sehen Sie die gesetzte title-Eigenschaft.

236 Wie kann ich das `style`-Objekt einsetzen?

Das Objekt `all` besitzt insbesondere selbst ein Unterobjekt mit Namen `style`, worüber alle Stilinformationen von sämtlichen Elementen in der Webseite abgefragt und meist auch geändert werden können. Dies kann zum Beispiel wieder über die folgende Syntax mit der `tags()`-Methode erfolgen:

```
document.all.tags["(Tagname)"][index].style.[Eigenschaft/Methode]
```

Listing 585: Veränderung der Stilinformationen eines Elements über das style-Objekt

Beispiel:

```
document.all.tags("h4")[0].style.color
```

Listing 586: In dem Beispiel sprechen Sie die Farbe der ersten Überschrift der Ordnung 4 in dem Dokument an.

Daneben können Sie auch wieder über eine ID arbeiten. Dies funktioniert über die Syntax:

```
document.all[id].style.[Eigenschaft/ Methode]
```

Listing 587: Ansprechen eines Elements einer Webseite über die ID

Die Syntax des Kennzeichnens im HTML-Tag:

```
<[HTML-Tag] id="[name]">
```

Listing 588: Die Kennzeichnung des HTML-Tags

Die Stileigenschaften vom `all`-Objekt unterscheiden sich in der Schreibweise ein wenig von der Schreibweise bei den CSS-Attributen, wie sie sonst im Web üblich sind. Dies führt oft zu Fehlern, da es nur geringe Unterschiede sind. Im Allgemeinen gilt, dass der Bindestrich bei CSS-Attributen bei den Eigenschaften des `all`-Objekts entfällt und der nachfolgende Buchstabe groß zu schreiben 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 `all`-Objektes 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>backgroundColor</code>	Eine Hintergrundfarbe für ein Element.
<code>backgroundImage</code>	Ein Hintergrundimage für ein HTML-Element.
<code>borderBottomWidth</code>	Die Breite von dem unteren Rahmen des HTML-Elements.
<code>borderColor</code>	Die Farbe von dem Rahmen des HTML-Elements.
<code>borderLeftWidth</code>	Die Breite von dem linken Rahmen des HTML-Elements.

Tabelle 15: Die wichtigsten Eigenschaften vom style-Objekt

Eigenschaft	Beschreibung
borderRightWidth	Die Breite von dem rechten Rahmen des HTML-Elements.
borderStyle	Den Stil vom Rahmen, wie einfach oder doppelt, um ein Blocksatz-HTML-Element.
borderTopWidth	Die Breite von dem oberen Rahmen des HTML-Elements.
clear	Die Seite des HTML-Elements, die umfließende Elemente erlaubt.
color	Die Farbe von dem Text eines HTML-Elements.
display	Überschreibt die normale Anzeige von Element und gibt an, ob ein Element in eine Zeile, als Blocksatzelement oder als Blocksatzlisteneintrag erscheinen soll.
fontFamily	Die Font-Familie, etwa Helvetica oder Arial, für ein HTML-Textelement.
fontSize	Die Fontgröße für ein HTML-Textelement.
fontStyle	Der Stil von der Schrift des HTML-Elements.
fontWeight	Die Dicke von der Schrift des HTML-Elements.
lineHeight	Der Abstand zwischen der Basislinie von zwei benachbarten Linien in einem Block.
listStyleType	Der Stil von dem Bullet in Listeneinträgen.
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.
paddingBottom	Spezifiziert, wie viel Leerraum zum Einfügen zwischen dem Boden von einem Element und seinem Inhalt, etwa Text oder ein Bild, gelassen wird.
paddingLeft	Spezifiziert, wie viel Leerraum zum Einfügen zwischen der linken Seite von einem Element und seinem Inhalt, etwa Text oder ein Bild, gelassen wird.
paddingRight	Spezifiziert, wie viel Leerraum zum Einfügen zwischen der rechten Seite von einem Element und seinem Inhalt, etwa Text oder ein Bild, gelassen wird.
paddingTop	Spezifiziert, wie viel Leerraum zum Einfügen zwischen der Spitze von einem Element und seinem Inhalt, etwa Text oder ein Bild, gelassen wird.
textAlign	Die Fließrichtung von einem HTML-Textelement in einem Block.

Tabelle 15: Die wichtigsten Eigenschaften vom style-Objekt (Forts.)

Eigenschaft	Beschreibung
textDecoration	Spezifiziert Spezialeffekte wie Blinken u. Ä. für ein HTML-Textelement.
textIndent	Die Größe der Einrückung vor der ersten formatierten Zeile eines HTML-Textblock.
textTransform	Die Art von einem HTML-Textelement.
visibility	Mit dieser Eigenschaft können Sie bestimmen, ob ein Element angezeigt wird oder nicht. Der Wert hidden bedeutet, das Element wird nicht angezeigt, und der Wert visible, dass das Element angezeigt wird.
whiteSpace	Spezifiziert, ob nicht sichtbare Zeichen (so genannter White Space) innerhalb eines HTML-Elements wegoptimiert werden sollen oder nicht.
width	Die Breite von einem HTML-Blockelement.

Tabelle 15: Die wichtigsten Eigenschaften vom style-Objekt (Forts.)

Betrachten wir ein Beispiel, in dem über `tags()` mit Angabe des HTML-Tags und dem entsprechenden Index mit JavaScript eine Formatierung gesetzt wird.

Beispiel (*d6.html*):

```

01 <html>
02 <script language="JavaScript">
03 function lade(){
04     document.all.tags("h1")[0].style.color="red";
05     document.all.tags("h1")[1].style.color="blue";
06     document.all.tags("h3")[0].style.color="white";
07     document.all.tags("body")[0].style.backgroundColor="yellow";
08     document.all.tags("body")[0].style.color="green";
09     document.all.tags("div")[0].style.fontSize="42";
10 }
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 589: Einsatz von `document.all`

In den Zeilen 4 bis 9 werden gezielt bestimmten Tags in der Webseite (nach Typ und Nummer in der Seite) Layoutausprägungen zugewiesen. Laden Sie die Datei in verschiedene Browser. Alte Browser werden sich sehr pikiert zeigen, einen Fehler melden und nur HTML-formatierten Text anzeigen. Aber auch neue Browser werden recht »interessant« reagieren. Selbstverständlich kann der Internet Explorer mit dem Objekt umgehen. Aber auch der Opera-

Browser und der Konqueror kommen damit zurecht. Jedoch auch neuere Versionen vom Navigator und Mozilla werden aussteigen, obwohl sie teilweise das `all`-Objekt unterstützen.



Abbildung 384: Der Konqueror zeigt, wie die Seite aussehen sollte.



Abbildung 385: Der Navigator und der Mozilla werden in neuen Versionen zwar keinen Fehler anzeigen, aber auch keinerlei Wirkung.

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.all.h.style.color="red";
08                 document.all.h.style.backgroundColor="blue";
09                 break;
10         case 1 : document.all.h.style.color="blue";
11                 document.all.h.style.backgroundColor="green";
12                 break;
13         case 2 : document.all.h.style.color="green";
14                 document.all.h.style.backgroundColor="yellow";
15                 break;
16         case 4 : document.all.h.style.color="yellow";
17                 document.all.h.style.backgroundColor="red";
18                 break;
19         default : document.all.h.style.color="cyan";
20                 document.all.h.style.backgroundColor="gray";
21     }
22     ani();
23 }
24 function ani() {
25     setTimeout('lade()',250);
26 }
27 </script>
28 <body onLoad="ani()">
29     <h1 id="h">Homepage</h1>
30 </body>
31 </html>

```

Listing 590: Eine kleine Animation unter Verwendung von dem all-Objekt und dessen Unterobjekt style

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.all.h.style` über diese ID Stilinformatoren 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. Das Rezept benutzt zum Austausch der Stilinformatoren 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.

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

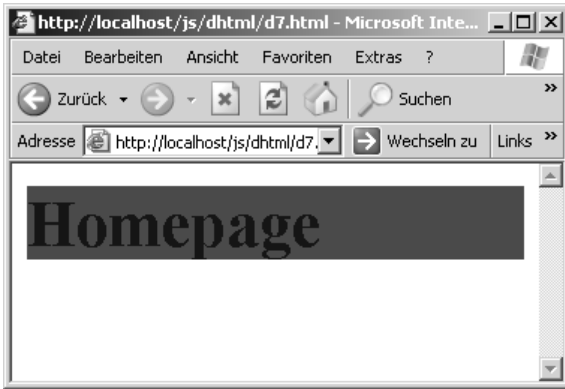


Abbildung 386: Dunkle Schrift

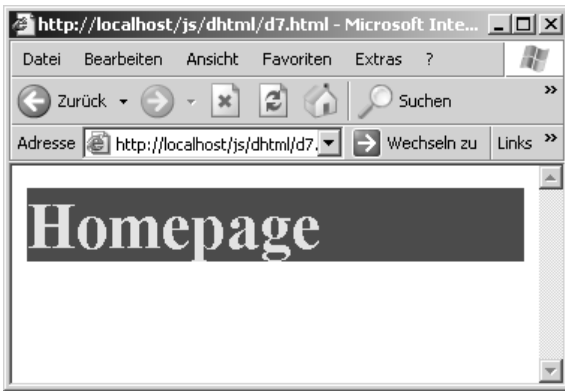


Abbildung 387: Eine helle Schrift und ein veränderter Hintergrund

237 Wie kann ich das style-Objekt so einsetzen, dass es in allen neueren Browsern funktioniert?

Da die bisherigen Beispiele auch in neueren Versionen des Netscape Navigators und Mozillas nicht funktionieren (obwohl das `all`-Objekt unterstützt wird), soll noch ein Beispiel mit einem Weg gezeigt werden, der auch in dieser Welt funktioniert. Dabei verwenden wir bei 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){
```

Listing 591: Eine Verwendung des style-Objekts, die auch bei neueren Versionen des Navigators und Mozillas funktioniert

```

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 591: Eine Verwendung des style-Objekts, die auch bei neueren Versionen des Navigators und Mozillas funktioniert (Forts.)

Hier wird bei der Überschrift der Ordnung 1 in Zeile 12 mit `this` eine Referenz auf das Element übergeben und in der jeweils aufgerufenen Funktion über die dort über `i` verfügbare Referenz auf das `style`-Objekt zugegriffen. Das funktioniert so dann in allen neueren Browsern.

Hinweis

Wenn Sie nicht bewusst Besucher aus der Netscape-/Mozilla-Welt sowie einige Exoten und alte Browser abhalten wollen, verzichten Sie bei unklaren Situationen auf den Einsatz von `document.all`. Es gibt Alternativen, die neutraler und weit reichender unterstützt werden. Diese verwenden direkt CSS, nutzen den Hover-Effekt bei Hyperlinks (siehe das Rezept »Wie kann ich den Hover-Effekt als Rollover-Effekt einsetzen?« auf Seite 794), arbeiten mit `document.getElementById()` oder arbeiten mit Image-Objekten (siehe das Rezept »Wie kann ich mit JavaScript dynamisch Bilder in einer Webseite erzeugen und darauf zugreifen?« auf Seite 770). Zu allen diesen Techniken finden Sie hier Rezepte.

238 Wie kann ich das style-Objekt in dem alten Netscape-Modell verwenden?

Obwohl alte Zöpfe irgendwann abgeschnitten werden müssen, soll zumindest kurz gezeigt werden, wie Sie das `style`-Objekt auch im alten Netscape-Modell verwenden können. Es ist beispielsweise über die Methode `document.contextual()` und die Eigenschaft¹ `document.tags` verfügbar.

Die Eigenschaft `tags` kreiert ein Style-Objekt, worüber die Stilangaben eines HTML-Tags spezifiziert werden können. Syntax:

1. Hier steht der Bezeichner `tags` und im Gegensatz zu den neueren Varianten eine Eigenschaft und keine Methode.

```
document.tags.[tagName]
```

Listing 592: Einsatz von document.tags

Dabei ist `tagName` der Name eines beliebigen HTML-Tags, beispielsweise `h1`.

Die Methode `contextual(xontext1, ...[contextN,] affectedStyle)` spezifiziert den Stil eines einzelnen HTML-Tags, wobei man verschiedene Tags schachteln kann, dass nur die so verschachtelten Tags betroffen sind. Die Parameter sind über die oben genannten Eigenschaften spezifizierte HTML-Tags, denen eine bestimmte Stilinformation zugewiesen wird, etwa wird so jedem `<span>`-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 593: 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 594: Einsatz von contextual() in JavaScript

Die Verwendung wird im Internet Explorer und kompatiblen Browsern einen Fehler auslösen, in neuen Navigator-Versionen und Konsorten ignoriert werden, aber etwa im Navigator 4.7 gut funktionieren. Schauen wir ein Beispiel zum Setzen von Stilinformationen an.

Beispiel (*d11.html*):

```
01 <html>
02 <head>
03 <title>Titel der Datei</title>
04 <style type="text/JavaScript">
05     contextual(document.tags.p, document.tags.i).color="blue";
06     contextual(
07         document.tags.h1, document.tags.span).backgroundColor="yellow";
```

Listing 595: Das Style-Objekt im praktischen Einsatz mit contextual()

```

08     contextual(document.tags.h2).backgroundColor="yellow";
09 </style>
10 </head>
11 <body>
12 <div>Gefahr <i>Feurio</i> Feinde</div>
13 <p>Wachet <i>auf</i>.</p>
14 <h1>Elbereth</h1>
15 <h1>Frodo <span>und </span> Sam</h1>
16 <h2>Streicher</h2>
17 </body>
18 </html>

```

Listing 595: Das Style-Objekt im praktischen Einsatz mit `contextual()` (Forts.)

In dem Listing verwenden wir die Methode `contextual()` des `document`-Objektes, um Stilinformationen anzugeben. Zum Teil sogar verschachtelt. Für alle kursiven Textpassagen in Absätzen legen wir in Zeile 5 fest, dass der Text dort blau dargestellt werden soll. Für alle Überschriften `h1` legen wir in Zeile 6 fest, dass dort eine gelbe Hintergrundfarbe verwendet wird. Ebenso werden alle Überschriften der Ordnung 2 grundsätzlich einen gelben Hintergrund haben (Zeile 8).



Abbildung 388: Die Anwendung im Navigator 4.7



Abbildung 389: In neueren Browsern ist allerdings keine Wirkung zu sehen oder gar ein Fehler.

239 Wie kann ich dynamisch Bilder 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 einsortiert wird. Dieses Datenfeld ist über `window.document.images` zugänglich. Ebenso ist es möglich, von Hand mit der Anweisung `new Image();` in JavaScript ein Bildobjekt zu erzeugen. Darüber kann per Skript bereits ein Bild geladen werden, das nicht sofort in der Webseite angezeigt wird. 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.

Tabelle 16: Die Eigenschaften eines Bildobjekts

Eigenschaften	Beschreibung
height	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.
hspace	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 hspace den Wert 0.
length	Diese Eigenschaft ist eine Besonderheit, da sie nicht einer Bildobjekt direkt zuzuordnen ist. Die Eigenschaft enthält die Anzahl der Grafiken in einem Datenfeld mit Grafiken. Das bedeutet insbesondere für eine Webseite, dass über <code>document.images.length</code> die Anzahl der Grafiken in der Webseite verfügbar ist.
lowsrc	Der URL der Vorschaugrafik. Korrespondierende Eigenschaft zu dem gleichnamigen HTML-Parameter.
name	Der Name der Grafik. Korrespondierende Eigenschaft zu dem gleichnamigen HTML-Parameter.
src	Der URL der Grafikdatei. Korrespondierende Eigenschaft zu dem gleichnamigen HTML-Parameter.
vspace	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 die vspace den Wert 0
width	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 16: Die Eigenschaften eines Bildobjekts (Forts.)

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 Pixel 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.

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 einen `<img>`-Tag dafür an, auch wenn dieser am Anfang nur eine unsichtbare Grafik referenziert. Eine solche unsichtbare Grafik kann eine vollkommen transparente Grafik sein oder eine Grafik, die nur ein Pixel groß ist.

240 Wie kann ich die Höhe und die Breite eines Bildes dynamisch verändern?

Die Antwort (und damit das Rezept) 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 (wie auch die folgenden Techniken mit `images`) 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>
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 596: Änderungen der Bildeigenschaften per JavaScript nach dem Laden der Webseite

In dem Beispiel sind zwei Grafiken in einer Webseite spezifiziert (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 390: Vor dem Klick auf einen Button



Abbildung 391: Nach Klick auf den Button 2

Wie kann ich ein Bild dynamisch austauschen?

Um ein Bild mit JavaScript dynamisch auszutauschen, setzen Sie einfach die `src`-Eigenschaft von einem Bild neu.

Beispiel (*d2.html*):

```
01 <html>
02 <script language="JavaScript">
03   function aender(i) {
04     document.images[i].src = "button12.gif";
05   }
```

Listing 597: Dynamische Änderungen von Bildern in einer geladenen Webseite

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

```
06 </script>
07 <body>
08   <br>
09   <form>
10     <input type="Button" value="Tausche" onClick="aender(0)">
11   </form>
12 </body>
13 </html>
```

Listing 597: Dynamische Änderungen von Bildern in einer geladenen Webseite (Forts.)

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 392: Vor dem Klick

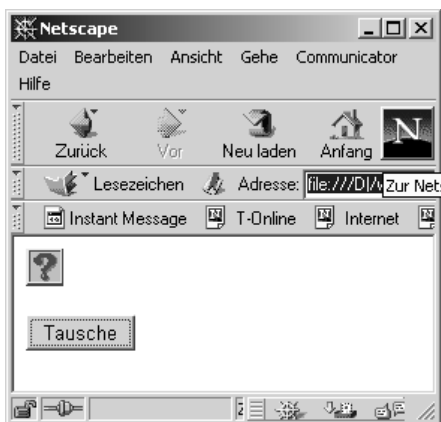


Abbildung 393: Nach dem Klick

241 Wie kann ich durch Setzen der src-Eigenschaft 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 hoch kompliziert aufgebaut werden. Einer der einfachsten und vor allem in nahezu allen Browsern unterstützte Wege führt darüber, einen Bild-URL beim Überstreichen mit dem Mauszeiger auszu-tauschen.

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. Im alten Netscape-Modell wurde mit so genannten *Layern* auf Seiten von HTML gearbeitet, die im Objektmodell ein Gegenstück in dem `layer`-Objekt fanden. Diese Layer kann man in diesem Modell dynamisch behandeln, ein- und ausblenden, verschieben usw. Da allerdings selbst die neuen Versionen des Navigators die Layer-technik nicht mehr unterstützen, ist es 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. Damit decken Sie zwar nicht alle ursprünglichen Möglichkeiten der Layer-Technik ab, aber einen weiteren Teil der mit der Alttechnik realisierten Effekte können sie vollkompatibel zu alten als auch neuen Browsern mit dem Hover-Effekt (*siehe das Rezept »Wie kann ich den Hover-Effekt als Rollover-Effekt einsetzen?« auf Seite 794*) realisieren. Dann benötigen Sie nicht einmal JavaScript. Ansonsten unterstützen nahezu alle modernen Browser mittlerweile die Philosophie des Microsoft-Konzepts bzgl. DHTML, und darauf kann man sich mittlerweile beschränken. Allerdings ist auch diese Art der Programmierung keinesfalls einfacher als die reine Verwendung von JavaScript und Bildobjekten. Es spricht sehr viel dafür, diese Techniken hier einzusetzen, wenn immer es geht.

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;
```

Listing 598: Ein einfacher Rollover-Effekt mit dem Austausch der src-Eigenschaft

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

```
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 598: Ein einfacher Rollover-Effekt mit dem Austausch der src-Eigenschaft (Forts.)

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 drei Mal die gleiche Grafik ist. Bei jedem `<img>`-Tag werden Sie zwei Eventhandler finden, die beides Mal 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 durch 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 Originaldatei durch die Datei, die über `tausch` referenziert wird, austauscht und beim zweiten Aufruf (`onMouseOut`) wieder zurücksetzt.

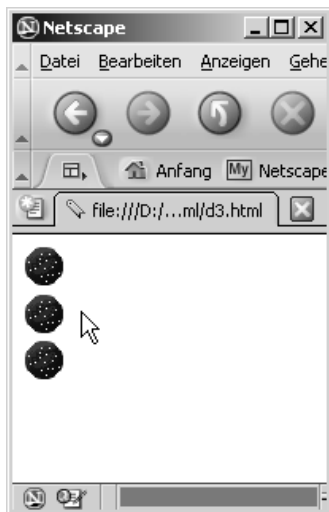


Abbildung 394: Der Mauszeiger befindet sich nicht im Bereich einer der Grafiken.

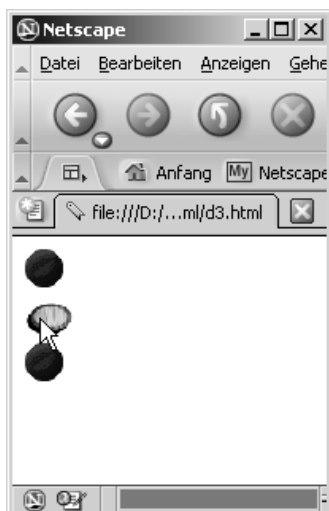


Abbildung 395: Der Mauszeiger befindet sich im Bereich der zweiten Grafik – diese wurde ausgetauscht.

242 Wie kann ich mit JavaScript und der src-Eigenschaft eine Animation realisieren?

Über die `src`-Eigenschaft können Sie hervorragend eine Animation realisieren, indem Sie zeitgesteuert Grafiken austauschen.

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

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 wie der Wert in `src` verwendet wird. In `document.images[i].src` wird der absolute URL verwendet, während `tausch.src` in dem Beispiel als relativer 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.

Beispiel (*d4.html*):

```

01 <html>
02 <script language="JavaScript">
03   dummy = new Image();
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 599: Animationen per JavaScript

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 von den sechs Bildern 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` geladen, das in Zeile 5 über das Setzen der `src`-Eigenschaft eine Grafik erhält. 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 für den 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 durch 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.

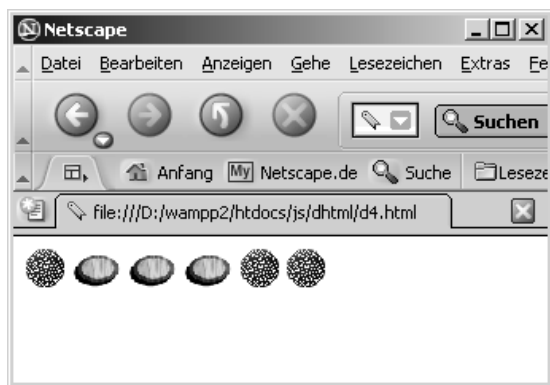


Abbildung 396: Die Animation ist am Laufen.

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, das 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 != tausche.src){
11         dummy.src = document.images[i].src;
12         document.images[i].src = tausche.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 600: Eine kleine Erweiterung der Animation

In Zeile 13 bis 18 wird das gerade zuvor ausgetauschte Bild wieder durch das Originalbild ausgetauscht. Wenn der Index `0` ist, wird das letzte Bild der Webseite ausgetauscht. Andernfalls immer das Bild zuvor.



Abbildung 397: Es wird immer genau eine Grafik ausgetauscht.

243 Wie kann ich mit JavaScript durch Veränderung der Größe einer Grafik eine Animation realisieren?

Eine verwandte Animationstechnik verändert die Größe einer Grafik zeitgesteuert und erzeugt damit einen Animationseffekt. Sie brauchen nur analog dem letzten Beispiel 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 601: 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 dem Navigator 7.1 seltsame Probleme (das finden Sie auch in anderen Rezepten in dem Buch beschrieben).

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

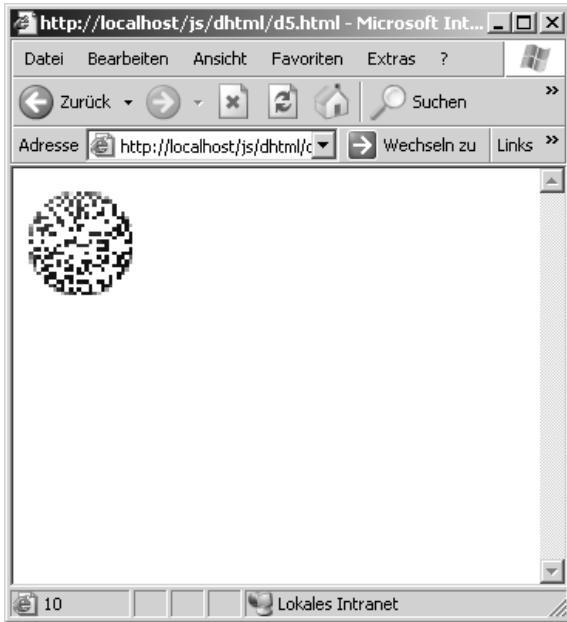


Abbildung 398: Die Grafik wird vergrößert und in der Statuszeile der Wert des Zählindex angezeigt – hier der Internet Explorer.

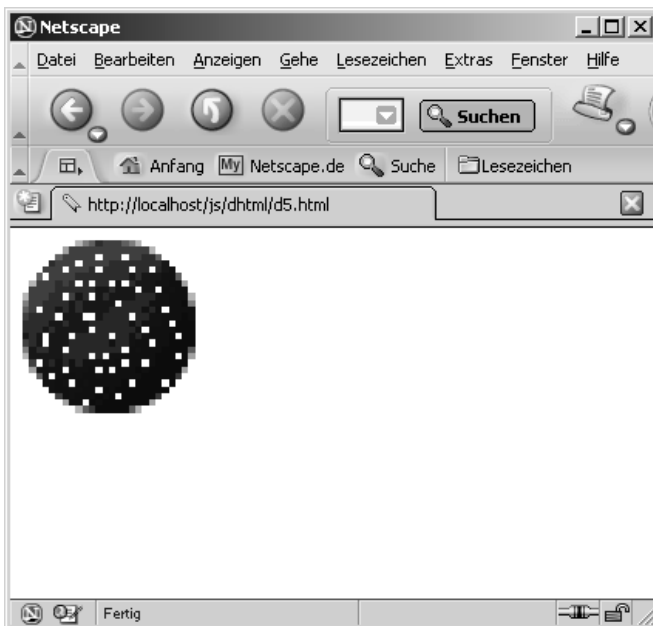


Abbildung 399: Im Navigator 7.1 wird zwar die Größe verändert, aber die Ausgabe in der Statuszeile macht die bekannten Probleme.



Abbildung 400: Die Ausgabe in der Statuszeile funktioniert im Navigator 4.7, aber die Größe der Grafik wird nicht verändert.

244 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.

Style Sheets sind keine eigene Sprache, sondern es gibt diverse Sprachen, mit denen Style Sheets erstellt werden können. Die genauen Regeln für die Arbeit mit Style Sheets werden je nach verwendeter Style-Sheet-Sprache etwas differieren, sind aber alle ähnlich. Es gibt nun verschiedene Versuche, Style Sheets zu standardisieren. Mittlerweile haben sich die Cascading Style Sheets (CSS), die bereits im Januar 1997 vom W3C eingeführt wurden, weitgehend im WWW durchgesetzt. Aber das ging nicht von heute auf morgen. Bezeichnend ist, dass – obwohl Cascading Style Sheets bereits seit 1998 in der 2. Version vorliegen und schon einige Jahre ins Land gegangen sind, der Internet Explorer 5.5 nicht einmal den auf den Webseiten des W3C zu findenden Acid-Test zur korrekten Darstellung von CSS 1.0 (!) bestand (<http://www.w3.org/>). Derzeit haben aber die neuesten Browser die Version CSS1 nahezu vollständig und die Version CSS 2 recht weitgehend umgesetzt. Es gibt nur noch marginale Abweichungen, die aber beachtet werden müssen².

245 Wie kann ich Style Sheets in eine Webseite einbinden?

Die Verbindung von Style Sheets mit einer HTML-Seite kann darüber geschehen, dass man Style Sheets in eine Seite einbettet oder aus einer externen Datei importiert. Zur tatsächlichen Definition von Style Sheets stehen Ihnen mehrere Standardtechniken zur Verfügung:

2. Die Details gehören aber in ein spezielles Buch zu Style Sheets.

- Direkte Notation im `<style>`-Container.
- Import externer Stilinformationen
- Die Inline-Referenz

Die wohl meistgebrauchte Technik zur Einbindung eines Style Sheets in eine Webseite erfolgt über die Notation eines `<style>`-Containers direkt in den HTML-Code. In dem Inneren des `<style>`-Containers werden – meist in HTML-Kommentare eingeschlossen, damit für ältere Browser die Anweisungen verborgen werden – alle HTML-Elemente bzw. Klassen (dazu kommen wir gleich) definiert, denen ein besonderes Layout zugewiesen werden soll. Man nennt diese Form der Verknüpfung »Eingebettete Formatvorlagen«. Wo das Style Sheet konkret in der HTML-Datei notiert wird, ist wie bei Skripten nicht festgelegt. Jedoch gelten wie bei Skripten die gleichen Argumente dafür, Style Sheets möglichst noch vor dem Laden des Body-Abschnitts bereitzustellen. Die konkrete Referenz erfolgt mit dem Tag `<style>` wie folgt (CSS-Style-Sheets):

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

Listing 602: Ein schematischer Style-Sheet-Container

Im Inneren werden die konkreten Stilinformationen notiert. Fehlt die Typ-Angabe des MIME-Typs, kann die Stilvereinbarung dennoch korrekt umgesetzt werden, wenn CSS die Default-Style-Sheets des Browsers sind. Das ist fast immer der Fall, aber darauf sollte man sich nicht verlassen.

Eine direkt in eine Webseite geschriebene Stilinformation hat natürlich auch einiges für sich, sonst gäbe es diese Variante nicht. Eine interne Lösung über eingebettete Style Sheets hält alle Informationen über eine Webseite zusammen und erlaubt schneller Änderungen. Außerdem hat man bei kleineren Seiten alles Notwendige direkt im Blick. Dennoch spricht in größeren Webprojekten viel für die Einbindung einer externen Variante. Eine externe Lösung hat bei Style Sheets wieder die gleichen Vorteile wie im Fall von externen JavaScript-Dateien:

- Gleiche Funktionalität kann in verschiedenen HTML-Dateien verwendet werden. Statt gleichen Code mehrfach in verschiedene Webseiten zu notieren, muss nur jeweils eine Referenz auf eine zentrale Datei angegeben werden.
- Nur eine zentrale Stelle ist bei Wartungen des Codes erheblich sinnvoller. Wenn die Funktionalität mehrfach verwendet wird, muss dort keine Veränderung vorgenommen werden. Das gilt bei der Verwendung von Style Sheets noch mehr als bei Skripten. Denken Sie nur an das Beispiel, wenn die Hintergrundfarbe in einem Projekt global geändert werden muss. Statt unter Umständen in Dutzenden von Webseiten den `<body>`-Tag anpacken zu müssen, kann eine einzige Änderung in der Style-Sheet-Datei das Problem lösen.

- ▶ Die Übersicht ist in einer reinen Style-Sheet-Datei besser.
- ▶ Insbesondere wird eine der Hauptintentionen für Style Sheets – die logische Entkopplung von Information und Layout – auch auf Dateiebene konsequent umgesetzt. Wenn das noch in Verbindung mit JavaScript gesehen wird, beinhaltet bei konsequenter Umsetzung die HTML-Datei die Information, die JavaScript-Datei die Logik, Steuerung und Dynamik der Webseite und die Style-Sheet-Datei das Layout (soweit es vollständig entkoppelt werden kann). In der HTML-Datei werden die Bereiche dann zusammengefügt.
- ▶ Die externe Style-Sheet-Referenz ist nur ein einzelner Tag und wird von inkompatiblen Browsern ignoriert. Und da es also keinen Container mit unverständlichen Anweisungen im Inneren gibt, kann es nicht zu Problemen kommen.

Eine externe Style-Sheet-Klartextdatei wird bei Stilformatvorlagen nach dem CSS-Standard meist die Erweiterung `.css` bekommen. Das ist jedoch wie die Erweiterung `.js` für JavaScript-Dateien optional.

Über die nachfolgende schematische Syntax kann ein Verweis auf ein externes Style Sheet erfolgen. Das nennt man dann ein verknüpftes Style Sheet bzw. eine Link-Notation. Die allgemeine Form sieht so aus:

```
<link type="[Style Sheet-Typ]" rel=stylesheet href="[URL einer Datei mit  
Stilinformationen]" title="[Name des Style-Sheets]">
```

Listing 603: Die schematische Einbindung einer externen CSS-Datei

Der Titel ist optional, die Typ-Angabe unter Umständen auch, aber das kann Probleme geben (s.o.). Die restlichen Attribute sind notwendig. Diese Zeile wird am sinnvollsten in den Header der HTML-Seite geschrieben und kennzeichnet eine Style-Sheet-Datei, von wo die in der HTML-Datei verwendeten Stilinformationen bei Bedarf hinzugebunden werden.

Hinweis

Beachten Sie, dass im Gegensatz zu dem `<style>`-Container hier kein Container definiert wird. Es gibt also keinen Ende-Tag.

Beispiel für eine CSS-Datei:

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

Listing 604: Eine Referenz auf eine externe Style-Sheet-Datei

Wie auch bei JavaScript gibt es bei Style Sheets die Möglichkeit einer Inline-Referenz. Dies bedeutet die Erweiterung eines gewöhnlichen HTML-Elementes um nur dort gültige Stilinformationen. Über ein zusätzliches Attribut – `style` (nicht mit dem Tag `<style>` zu verwechseln) – wird innerhalb des gewünschten HTML-Tags ein Stilattributwert gesetzt. Die Stilinformation gilt ausschließlich innerhalb des mit dem HTML-Befehl definierten Contai-

ners (wobei Container im Sinn von Wirkung des Tags zu verstehen ist – das kann auch ohne Ende-Tag gelten). Eine Inline-Deklaration ist dann sinnvoll, wenn ein Element nur ein- oder zweimal verwendet werden soll und nur wenige Stilformatierungen gesetzt werden sollen. Die abstrakte Syntax sieht so aus:

```
<[HTML-Tag] style="Stilinformationen">
```

Listing 605:

Optional kann wie bisher auch über den `type`-Parameter noch der Typ der Style Sheets definiert werden. Die Angabe `type="text/css"` spezifiziert wie oben bereits verwendet beispielsweise CSS.

246 Wie sieht eine CSS-Deklarationen grundsätzlich aus?

Dies ist hier kein Buch zu CSS, und für die genauen Details zur CSS-Syntax muss auf die entsprechende Literatur verwiesen werden. Nichtsdestotrotz soll die grundsätzliche Syntax vermittelt werden. Die Syntax von einer CSS-Deklaration sieht immer gleich aus. Es gibt jedesmal den folgenden Aufbau:

```
name : wert
```

Listing 606: Das grundsätzliche Schema einer CSS-Stilvereinbarung

Dabei gibt `name` das zu formatierende Element an und `wert` die konkrete Stilinformation. Das zu formatierende Element ist meist einer der existierenden HTML-Tags.

Beispiel:

```
h1 { color : white }
```

Listing 607: Ein Stilinformation für eine Überschrift der Ordnung 1

Bei Bedarf werden mehrere Deklarationen mit Semikolon getrennt. In vielen Situationen werden die Werte auch in geschweifte Klammern gesetzt.

Beispiel:

```
h1 {  
    background : yellow;  
    color : blue  
}
```

Listing 608: Mehrere Stile in einer Stilvereinbarung werden mit Semikolon getrennt.

Hinweis

Die Stilinformationen kann man natürlich auch in eine Zeile notieren. Eine Verteilung auch mehreren Zeilen ist oft nur übersichtlicher. Die letzte Stilangabe kann ohne abschließendes Semikolon notiert werden, aber aufgrund der Einheitlichkeit sollte es dennoch notiert werden.

Sie können auch mehrere Tags mit einer Deklaration formatieren (so genannte *Stilgruppen*). Die Tags werden durch Kommata getrennt.

Beispiel:

```
h1, h2, h3 {  
    background : yellow;  
    color : blue  
}
```

Listing 609: Mehrere Tags in einer Stilvereinbarung

Achtung

Beachten Sie unbedingt, dass die Wertzuweisung hier mit einem Doppelpunkt zu erfolgen hat. Die Geschichte ist deshalb gefährlich, weil man sowohl aus JavaScript als auch HTML ein Gleichheitszeichen gewohnt ist. Dazu kommt, dass der Internet Explorer auch das Gleichheitszeichen als korrekt wertet. Das scheint zwar auf den ersten Blick eine Erleichterung, disqualifiziert ihn aber im Grunde als Testbrowser, denn Sie übersehen unter Umständen Flüchtigkeitsfehler, die in anderen (korrekt interpretierenden) Browsern Probleme machen.

Ein HTML-Tag kann mit einem Punkt abgetrennt um die optionale Angabe einer Klasse erweitert werden. Die Syntax sieht so aus:

```
[HTML-Tag].[klasse]
```

Listing 610: Angabe einer Klasse eines bestehenden HTML-Tags

Dies ist die Festlegung der Eigenschaften für eine Instanz eines Elements. Dabei legt man eine Klasse fest und verwendet diese bei der Stilzuweisung. Für *klasse* kann jeder beliebige Name genommen werden. Damit ist es möglich, verschiedene Stile für ein und dasselbe HTML-Element zu erstellen. Der ursprüngliche HTML-Tag, welcher vor dem Punkt steht, ist das Elternelement zu dieser Instanz.

Beispiel:

```
h1.a {  
    background : yellow;  
    color : blue  
}
```

Listing 611: Mehrere Klasse von einem Tag

```
    }  
h1.b {  
    background : red;  
    color : green  
}
```

Listing 611: Mehrere Klasse von einem Tag (Forts.)

Die Angabe einer Klasse kann auch alleine in der Stildeklaration angegeben werden. In diesem Fall muss einem Klassenbezeichner in der Deklaration ein Punkt vorangestellt werden und bei der konkreten Verwendung ein HTML-Element um den so benannten neuen Parameter mit dem HTML-Parameter `class` erweitert werden. Die Syntax sieht so aus:

```
.[klasse]
```

Listing 612: Angabe einer Klasse ohne einen vorangestellten HTML-Tag

Die Verwendung in einem Tag geht so:

```
<[HTML-Tag] class="[klasse]">
```

Listing 613: Anwendung einer CSS-Klasse auf einen HTML-Tag

Beispiel:

```
.a {  
    background : red;  
    color : green  
}
```

Listing 614: Angabe einer Klasse ohne einen vorangestellten HTML-Tag

Die Verwendung in einem `<span>`-Tag geht so:

```
<span class="a">
```

Listing 615: Anwendung einer CSS-Klasse auf einen `<span>`-Tag

Hinweis

Die Festlegung von Klassen bei Stilvereinbarungen ist besonders deshalb interessant, weil man mittels Skripten über diese Angabe auf diese Stilvereinbarungen zugreifen kann.

Wenn Sie mit Style Sheets einen HTML-Tag formatieren, brauchen Sie in der Webseite bei den formatierten HTML-Tags keine Veränderung vornehmen. Sobald Sie einen Tag verwenden

den, wirken die Formatierungen über Style Sheet. Wenn Sie aber eine Klasse mit Style Sheets spezifizieren, müssen Sie diese in dem HTML-Tag mit dem Parameter `class` zuordnen. Die konkrete Syntax sieht so aus:

```
<[HTML]-Tag class="Stilklass">
```

Listing 616: Zuordnung einer Stilklasse

Wie verhalten sich CSS-Prioritäten?

Wenn mehrere Style Sheets in einer Seite vorhanden sind, kann es zu Konflikten kommen. Diese werden nach festgelegten Regeln aufgelöst:

- ▶ Stilvereinbarungen über HTML werden so lange unverändert weiterwirken, solange sie eine Style-Sheet-Vereinbarung nicht überlagert.
- ▶ Sofern sich zwei Stilinformationen der gleichen Art auf dasselbe Element beziehen und sich Formatierungsvorschriften widersprechen, wird von den meisten Browsern die jeweils zuletzt notierte Stileinstellung für ein bestimmtes Element verwendet. Das kann aber nicht garantiert werden, und so etwas sollte dringend vermieden werden.
- ▶ Ansonsten legt die Art der Einbindung die Priorität fest. Die Inline-Verknüpfung hat die höchste Priorität, gefolgt von der eingebetteten Formatvorlage. Die verknüpfte Formatvorlage hat die niedrigste Priorität.

Das Prioritätsschema kann auch dazu verwendet werden, über Style Sheets global definierte Elemente mit einer Inline-Deklaration wieder zu überschreiben.

247 Wie kann ich JavaScript auf Style Sheets anwenden?

Mit der Verbindung von Style Sheets und JavaScript lassen sich hervorragende dynamische Effekte erzielen. Wenn eine HTML-Seite vom Browser geladen wird, können im Rahmen vom DOM die eigenständigen Elemente als Objekte klassifiziert werden (abhängig von der genauen Umsetzung der Browserhersteller). Auf diese Weise kann der Browser jedes unterscheidbare Element einer Seite eindeutig wieder finden. Als Elemente fasst man in diesem Zusammenhang auch die unter CSS erstellten Stilvorlagen auf, die über Skripte auch angesprochen und verändert werden können.

Dies kann etwa direkt über Veränderungen der Webseite durch den Anwender (etwa über die Tastatur oder die Maus) geschehen oder aufgrund von Eingaben bei irgendwelchen Elementen als eine automatische Folgereaktion.

Im Mittelpunkt der dynamischen Veränderung von Webseiten über JavaScript steht das Attribut `ID`, mit dem Sie HTML-Tags erweitern können und über das ein Skript ein Element in einer Webseite identifizieren kann. Allerdings wird sich die weitere Vorgehensweise im Microsoft- und im (neueren) Netscape-Modell unterscheiden³.

3. Das alte Netscape-Modell ist noch ein ganz anderes Thema.

Wie erfolgt der Zugriff über die Eigenschaft `className`?

Im Rahmen des Microsoft-Modells verwendet man im Skript für den Zugriff auf ein HTML-Container die Eigenschaft `className` des Objekts von dem zu formatierenden Element, das über die ID bereitsteht.

Beispiel (*d12.html*):

```
01 <html>
02 <style TYPE="text/css">
03 <!--
04 .stil1 {
05   color:red;
06   font-style:italic;
07   font-size=18;
08   background-color = yellow
09 }
10 .stil2 {
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=="stil1") {
22     P1.className="stil2";
23     P2.className="stil1";
24   }
25   else {
26     P1.className="stil1";
27     P2.className="stil2";
28   }
29 }
30 //-->
31 </script>
32 <body>
33 <p class=stil1 id=P1>Mordor</p>
34 <p class=stil2 id=P2>Angmar</p>
35 <form>
36   <input type=button value="Farbwechsel" onClick="farbwechsel()">
37 </form>
38 </body>
39 </html>
```

Listing 617: Austausch von Stilinformationen per Button

Das Beispiel verwendet zwei Stilklassen (`stil1` und `stil2` in Zeile 4 bis 9 und 10 bis 15), die verschiedene Stilangaben definieren. Den beiden Absätzen in der Webseite wird über den



Abbildung 401: Originalaussehen

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 402: Die Absätze verändern dynamisch ihr Aussehen.

Wie erfolgt der Zugriff über getElementById()?

Auch mit dem neuen Netscape-Modell kann auf alle Elemente in Tag-Containern per JavaScript zugegriffen werden. Dazu ist wie beim Microsoft-Modell der ID-Parameter beim HTML-Tag wichtig. Allerdings kann dabei beim Zugriff über JavaScript nicht `className` zum Einsatz, sondern die Methode `getElementById()` des `document`-Objekts. Dieser Methode wird als Parameter der Wert des ID-Attributs des entsprechenden Tags übergeben. Der Rückgabewert der Methode ist eine Referenz auf das entsprechende Element. Darüber können Sie dann auch beispielsweise auf die `style`-Eigenschaft des Elements zugreifen.

Hinweis

Auch der Internet Explorer ab Version 5 unterstützt die Methode `document.getElementById()`. Und auch die meisten anderen modernen Browser. Damit bietet sich die Technik für den Zugriff an, wenn Sie möglichst breiter Unterstützung gewiss sein wollen.

Mit dem folgenden Code macht man beispielsweise den mit der ID `a` gekennzeichneten Container unsichtbar:

```
document.getElementById("a").style.visibility = "hidden"
```

Listing 618: Zugriff auf ein Element mit getElementById()

Schauen wir uns ein vollständiges Beispiel an, das zeitgesteuert einen Container erst unsichtbar und dann wieder sichtbar macht.

Beispiel (*d6a.html*):

```
01 <html>
02 <script language="JavaScript">
03 function lade(){
04     document.getElementById("a").style.visibility = "hidden";
05     setTimeout('document.getElementById("a").style.visibility = "visible"',2000);
06 }
07 </script>
08 <body onLoad="lade()">
09     <div align="center">
10         <h1>Willkommen</h1>
11         <h3 id="a">auf</h3>
12         <h1>meiner Homepage</h1>
13     </div>
14 </body>
15 </html>
```

Listing 619: Zeitversetztes Anzeigen von Teilen der Webseite

In Zeile 11 wird die Überschrift der Ordnung 3 mit der ID `a` versehen. Die beim Laden der Webseite über den Eventhandler `onLoad` aufgerufene `lade()`-Funktion greift in den Zeilen 4 und 5 mit `document.getElementById("a")` auf das `style`-Objekt zu und setzt zuerst den Con-

tainer auf unsichtbar (Zeile 4 – `document.getElementById("a").style.visibility = "hidden";`), um ihn dann nach zwei Sekunden auf sichtbar zu setzen (Zeile 5 – `setTimeout('document.getElementById("a").style.visibility = "visible";',2000);`).



Abbildung 403: Das Wort auf ist nicht zu sehen.



Abbildung 404: Nach zwei Sekunden taucht das Wort auf auf ;-)

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

248 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 große Tricks im Quelltext vornehmen, 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 620: Die grundsätzliche Syntax des Hover-Effekts

Beispiel:

```
a:hover {  
  color: #FF0000;  
  font-size: 14pt;  
  font-weight: bold  
}
```

Listing 621: 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 von dem Link 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   }  
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>
```

Listing 622: Der Hover-Effekt

```
16 </body>
17 </html>
```

Listing 622: Der Hover-Effekt (Forts.)

In dem Beispiel wird der normale Hyperlink in den Zeilen 3 bis 6 formatiert und der Hover-Effekt in Zeile 7 bis 11.

249 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 623: 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 Sinn.

Beispiel (*d14.html*):

```
01 <html>
02   <style TYPE="text/css">
03   <!--
04   .pos1 {
05     position:absolute;
06     left:60px;
07     top:40px;
08   }
09   .pos2 {
10     position:absolute;
11     left:100px;
12     top:60px;
13   }
14   .pos3 {
15     position:absolute;
16     left:140px;
17     top:75px;
```

Listing 624: Überlagerte Grafiken

Grund-
lagen

Core

Formu-
lare

Aus-
nahme

Arrays

Strings

Datum

Platt-
form

DHTML

Events

```

18  }
19  .pos4 {
20    position:absolute;
21    left:170px;
22    top:70px;
23  }
24  -->
25  </style>
26  <body>
27    <span class=pos1>
28      
29    </span>
30    <span class=pos2>
31      
32    </span>
33    <span class=pos3>
34      
35    </span>
36    <span class=pos4>
37      
38    </span>
39  </body>
40  </html>

```

Listing 624: Überlagerte Grafiken (Forts.)

In den Zeilen 4 bis 23 werden Positionen in der Webseite festgelegt. Die `<span>`-Container in den Zeilen 27 bis 38 werden entsprechend auf die Positionen geschoben. Unabhängig davon, wo diese Elemente per HTML in der Webseite positioniert sind. Die Elemente können auch überlagert werden (wie in dem Beispiel). Dabei definiert die Reihenfolge, wie die Elemente positioniert werden. Also welches Element welches andere überlagert.

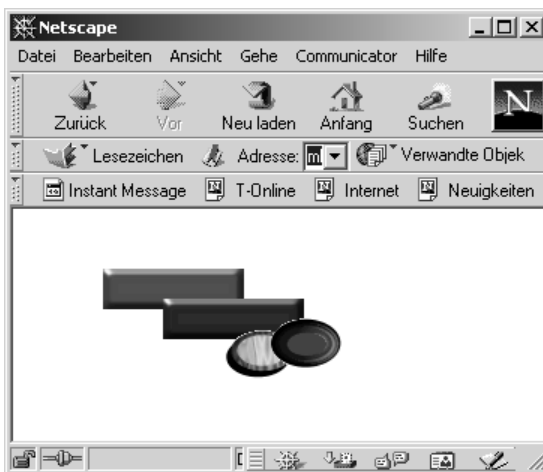


Abbildung 405: Selbst alte Browser wie der Netscape Navigator 4.7 kommen damit zurecht.

250 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;
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 625: Kein Klick möglich

Grund-
lagen

Core

Formu-
lare

Aus-
nahme

Arrays

Strings

Datum

Platt-
form

DHTML

Events

Das Beispiel beeinflusst dynamisch die Position von Elementen über Style Sheets. Dabei wird der HTML-`<span>`-Tag verwendet, der mit dem `id`-Attribut `P1` versehen wird (Zeile 29). Darüber greift die Funktion `poswechsel()` auf die Stilklasse zu. 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 Stilklasse gerade aktuell zugewiesen ist, und dann jeweils die alternative Stilklasse zugewiesen.



Abbildung 406: Die Originalposition des Hyperlinks



Abbildung 407: Der Mauszeiger wurde auf die Originalposition bewegt.

251 Wie kann ich anspruchsvollere Animationen mit JavaScript und Style Sheets durchführen?

Wir kommen zu Animationstechniken zurück und wollen mit JavaScript Positionsveränderungen einfließen lassen, die Animationseffekte erheblich erweitern können.

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 Stilklasse zu, die eine Position im sichtbaren Bereich des Browsers spezifiziert.

252 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 }
```

Listing 626: Eine Animation mit Bild-, Positions- und Größenveränderung

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DOM

Events

```
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);
```

Listing 626: Eine Animation mit Bild-, Positions- und Größenveränderung (Forts.)

```

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 626: 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-Positionsangaben 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 Zeilen 77 und 78), entsteht ein richtiges kleines Video. Die Handlung dieser Animation ist, dass sich der Hund dem Geschenk nähert.

Wenn die Positionen deckungsgleich sind, werden beide (!) Bilder zeitgleich (Zeilen 77 und 78) durch ein neues animiertes GIF ersetzt.

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

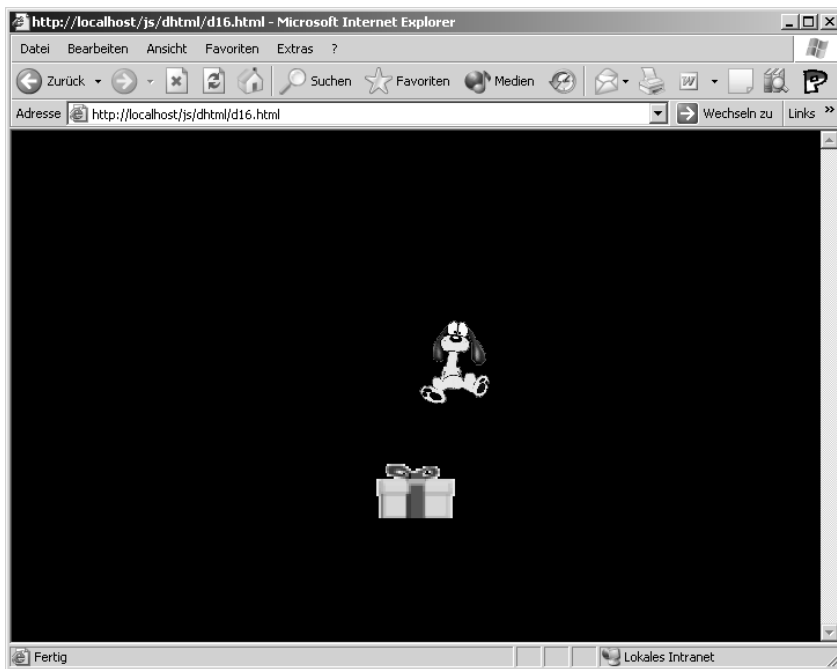


Abbildung 408: Der Hund nähert sich dem Geschenk.

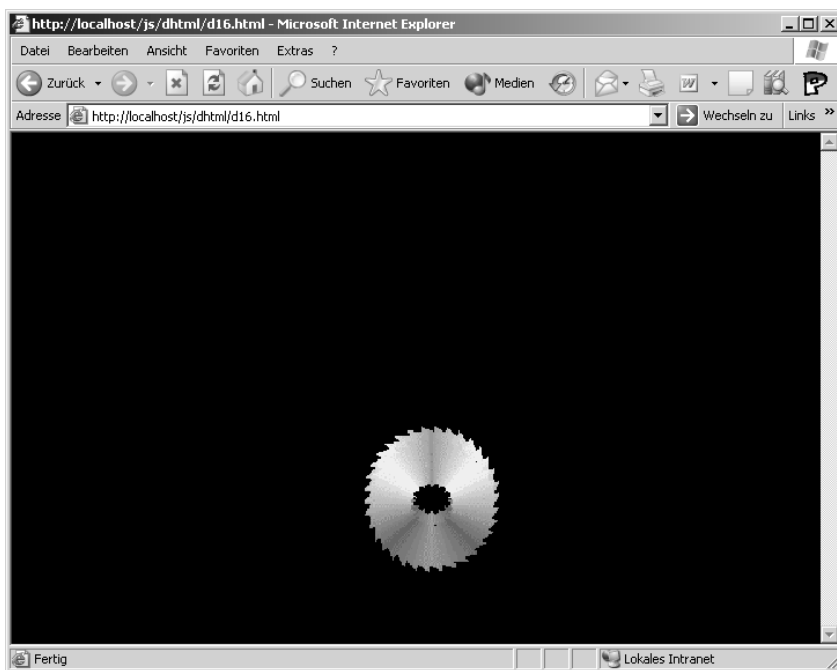


Abbildung 409: Zusammenstoß der Objekte

Die meisten Webanimationen beruhen auf einer ähnlichen Idee, wie wir sie gerade realisiert haben. Sie kann natürlich noch beliebig verfeinert werden.

253 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.

```
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' height=50
name='bat'></span>");
20     }
21     schreibePos();
22     ani();
23   //-->
24   </script>
25 </body>
26 </html>
```

Listing 627: Dynamisches Schreiben von Style Sheets

Das Beispiel verschiebt eine Grafik entlang Positionen, die mit Style Sheets angegeben werden. Dabei nutzen wir JavaScript, um beim Laden der Webseite dynamisch erst die Style Sheets zu schreiben. In Zeile 21 wird die Funktion `schreibePos()` aufgerufen. Diese generiert zuerst in Zeile 13 den Beginn vom Style-Sheet-Container. Dann wird mit einer `for`-Schleife (Zeilen 14 bis 17) die Positionsangabe 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 }");
```

Listing 628:

- ▶ 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 abschließend je nach Schleifendurchlauf `px; top: 110px },px; top: 120px }` etc.

Es wird also eine Reihe mit CSS-Positionsangaben erstellt, deren Anzahl sich aufgrund des Wertes der Variablen `anzahl` (Zeile 4) ergibt.

Abschließend schreibt die Funktion `schreibePos()` in Zeile 18 und 19 mit `document.write("</style><body><span class=pos1 id=P1><img src='79.gif' height=50 name='bat'></span>");` das Ende vom Stilcontainer und einen Body mit einer Bildreferenz (die verschoben werden soll).

Nach dem Schreiben der CSS-Positionsangaben wird die Funktion `ani()` in Zeile 22 aufgerufen. Dort wird zuerst eine String-Variable `befehl` eingeführt (Zeile 6). Diese wird in der nachfolgenden `for`-Schleife verwendet, um den Befehl für `setTimeout()` in Zeile 8 zusammenzusetzen. Dabei wird der Zugriff auf die Stilklasse 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()`. 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 für das Bild. In Zeile 9 wird zudem die Zeitspanne für `setTimeout()` mit der Zählvariablen `i` vergrößert, um die Intervalle zu generieren.

254 Wie kann ich ein Element in Abhängigkeit von der Mausposition positionieren?

Im Rahmen der Auswertung des `event`-Objekts zur globalen Ereignisbehandlung (siehe dazu das folgende Kapitel) steht Ihnen auch die Position des Mauszeigers zur Verfügung. Damit können Sie natürlich in Abhängigkeit 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' height=50
09       name='bat'></span>");
10   }
11   //-->
12 </script>
13 </html>
```

Listing 629: Positionieren einer Grafik an der Stelle, wo ein Mausklick erfolgt

In Zeile 7 wird für einen `<span>`-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 `<span>`-Container erfolgt, sobald ein Anwender in die Webseite klickt.

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

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.

255 Wie funktioniert Ereignisbehandlung grundsätzlich in JavaScript?

Allgemein haben wir in zahlreichen Rezepten in diesem Buch Ereignisbehandlung eingesetzt. Das ist im Grunde nicht anderes als die Festlegung, unter welchen Umständen ein fertig programmiertes Skript ausgeführt wird. Grundsätzlich ist das Laden einer Webseite nur eine von vielen Situationen, in denen die Ausführung einer Skriptfunktionalität sinnvoll ist. Sinnvolle andere Situationen können aber auch das sein:

- ▶ Das Verlassen einer Webseite.
- ▶ Der Klick auf einen Hyperlink.
- ▶ Der Klick auf einen Button in einem Formular.
- ▶ Der Doppelklick auf eine Grafik.
- ▶ Das Überstreichen einer Fläche mit dem Mauszeiger.
- ▶ Ein Betätigen einer Taste auf der Tastatur
- ▶ Das Verlassen eines Eingabefelds in einem Formular.

Es gibt noch zahlreiche weitere Ereignisse, bei deren Auftreten ein Skript gezielt ausgeführt werden kann. Diese weiter 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.

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

Seit HTML 4.0 gibt es universelle Eventhandler für Ereignisse in einer Webseite. Es handelt sich einfach um Attribute, die direkt in einen HTML-Tag geschrieben wird. Das W3-Konsortium hat Eventhandler offiziell in den HTML-Sprachstandard mit aufgenommen. Dabei wurde auch festgelegt, in welchen HTML-Tags welcher Eventhandler vorkommen darf¹. 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 630: Die grundsätzliche Syntax eines Eventhandlers

Unter `[HTML-Tag]` ist ein beliebiger HTML-Tag zu verstehen, welcher die Reaktion auf ein Ereignis unterstützt, `[ATTRIBUTE]` sind die normalen HTML-Attribute, welche den HTML-Tag bezüglich seiner üblichen HTML-Auswirkungen spezifizieren, `[onEventHandler]` soll der konkrete Name des Ereignisses sein, auf welches reagiert werden soll, und "Skriptanweisungen" sind die tatsächlichen Skriptaufrufe, welche bei Auftreten des spezifizierten Ereignisses ausgeführt werden sollen. Ein Eventhandler-Ereignis sieht aus wie jedes gewöhnliche HTML-Attribut. Gemeinhin beginnen die Namen von Ereignissen 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 Istgleichzeichen notiert, gefolgt von einer – meist zur Sicherheit in Anführungszeichen notierten – JavaScript-Anweisung oder -Funktion. In eine Eventhandler-Struktur können über Semikolon abgetrennt mehrere JavaScript-Anweisungen hintereinander geschrieben werden. Dies ist aber selten sinnvoll. Besser ist die Erzeugung einer eigenen JavaScript-Funktion, welche dann alle Anweisungen enthält, welche bei Auftreten eines Ereignisses ausgeführt werden sollen. Allgemein unterscheidet man die potenziell zu behandelnden Ereignisse nach Kategorien.

256 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 Skript-Anweisungen einfach in einen `<script>`-Container notiert werden.

Die Syntax beim Verwenden des Eventhandlers ist folgende:

```
<body onLoad="[Anweisung]">
```

Listing 631: Syntax von onLoad

1. Zumindest von der Theorie her – was die Browser-Hersteller daraus machen, ist ein anderes Thema.
2. Beziehungsweise Skriptaufrufen im Allgemeinen.

Die Variante mit dem Eventhandler hat die Vorteile, dass ein Aufruf einer Funktion erst dann erfolgt, wenn der `<body>`-Tag bereits geladen wurde – egal wo die aufgerufene Funktion notiert ist. Ebenso 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 HTML-Codes an einer Stelle Skriptfunktionalität benötigt. Ebenso gibt es einige Situationen, in denen der Eventhandler `onLoad` nicht richtig ausgeführt wird³.

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 den `<body>`-Tag integriert wird.

```
<body onLoad="[Anweisung]">
```

Listing 632: 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.

Beispiel (e1.html):

```
01 <html>
02   <script language="JavaScript">
03     <!--
04     function wk() {
05       alert("Willkommen");
06     }
07     function bey() {
08       alert("Ciao");
09     }
10     //-->
11   </script>
12   <body onLoad="wk()" onUnload="bey()">
13   </body>
14 </html>
```

Listing 633: Die Eventhandler onLoad und onUnload

Bei Laden der Webseite werden Sie mit einem Mitteilungsfenster begrüßt. Das öffnet die Funktion `wk()` (Zeilen 4 bis 6). Die Funktion `bey()` (Zeile 7 bis 9) wird aufgerufen, wenn die Webseite verlassen wird.

-
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, welche die Seite dynamisch schreibt, wird der Frame-Inhalt beim Verändern der Größe neu aufgebaut, die Funktion jedoch nicht mehr ausgeführt.

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 wichtige Aktionen dort hinein legen.

257 Wie kann ich beim Laden einer Grafik eine Anweisung aufrufen?

Der Eventhandler `onLoad` lässt sich auch beim `<img>`-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 634: Syntax von `onLoad` bei der Verwendung im `<img>`-Tag

258 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 `<img>`-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 Grafik geladen wird. Die grundsätzliche Syntax beim Verwenden des Eventhandlers ist folgende:

```

```

Listing 635: Syntax von `onAbort` bei der Verwendung im `<img>`-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 Program-

miersprachen kann damit nicht realisiert werden. Die grundsätzliche Syntax beim Verwenden des Eventhandlers ist folgende:

```

```

Listing 636: Syntax von onError bei der Verwendung im -Tag

259 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. Unter 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 den entsprechenden Tag mit dem Attribut onClick. Dabei gilt die allgemeine Syntax:

```
<[HTML-Tag] onClick="[Funktionsname]">
```

Listing 637: 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 638: 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>, <area>, <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 Eventhandlern durch alte Browser wie den Navigator vor 6 gilt leider nicht nur bei diesem Fall, sondern bei den meisten weiteren Eventhandlern. Und leider gilt auch für neuere Browser, dass die Unterstützung bei allen Tags nicht garantiert ist (obgleich die Inkonsistenz bei diesem Tag immer mehr abnimmt). Sie sollten aber grundsätzlich zur Sicherheit alle Browser testen, die Sie unterstützen wollen, wenn Sie einen Eventhandler bei einem bestimmten Tag 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 umgehen. Wenn man beispielsweise eine Grafik ohne Hyperlink-Kennzeichnung haben möchte, setzt man das Attribut `border=0`. Bei Texten kann man mit der zusätzlichen Anwendung von Style Sheets die Unterstreichung unterbinden. 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.

Beispiel (*e2.html*):

```
01 <html>
02   <body>
03     <h1 align="center" onClick="alert('Schau!')">
04       Klicken Sie überall hin
05     </h1>
06     
07     <hr noshade size="4" onClick="alert('Das kommt von einer Trennlinie')">
08     <a href="#" onClick="alert('So klappt es auch mit allen Browsern')">
09       
10     </a>
11   </body>
12 </html>
```

Listing 639: Einsatz von `onClick`

Bei allen Tags innerhalb des Bodys sind `onClick`-Eventhandler zu finden, die einfach eine Meldung ausgeben. Testen Sie die Seite in verschiedenen Browsern, und klicken Sie auf die Elemente.



Abbildung 410: Selbst der `<hr>`-Tag reagiert in den meisten modernen Browsern auf `onClick`.

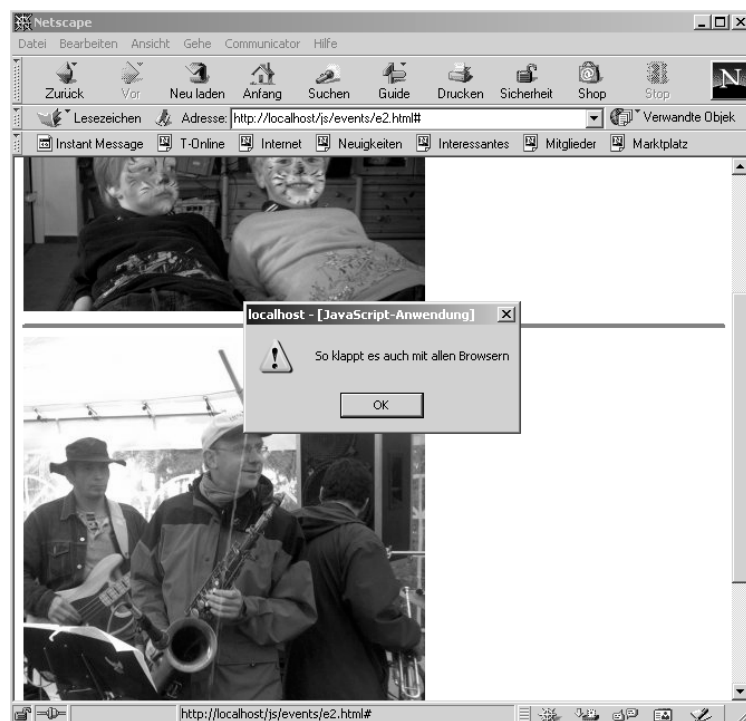


Abbildung 411: Mit `onClick` in einem `Hyperlink`-Tag kommen auch Browser wie der Netscape Navigator 4.7 zurecht.

Eng verwandt mit dem einfachen Klick auf ein sensitives Element ist die Reaktion auf Doppelklicks. Mit dem Eventhandler `onDb1Click` 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 `onDb1Click` ist so gut wie nie sinnvoll. Meist wird der einfache Klick ausgelöst, egal wie schnell der Doppelklick durch einen Anwender erfolgt. Das ist insbesondere deshalb ein Problem, weil man als Webseiten-ersteller natürlich keinerlei Einfluss auf die beim Anwender eingestellte Doppelklickgeschwindigkeit hat.

In den gleichen Kontext wie `onClick` und `onDb1Click` fallen auch die beiden Eventhandler `onMouseDown` und `onMouseUp`. Sie werden zur Überwachung des Drückens einer Maustaste und zu deren Loslassen verwendet. Sie werden wie das Ereignis `onClick` verwendet, nur halt bei der jeweiligen Teilaktion ausgelöst. 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.

260 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 Mitteilungsfenster oder der Statuszeile angezeigt werden. Dazu muss der 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.

Beispiel zur Ausgabe von Zusatzinformationen beim Überstreichen mit dem Mauszeiger

```
01 <html>
02   <body>
03     
05   </body>
06 </html>
```

Listing 640: Beim Überstreichen eines Bildes erscheinen in der Statuszeile Zusatzinfos

Beobachten Sie beim Test des Beispiels die Statuszeile. Beim Überstreichen wird dort jeweils eine Meldung zu jedem Bild angezeigt.

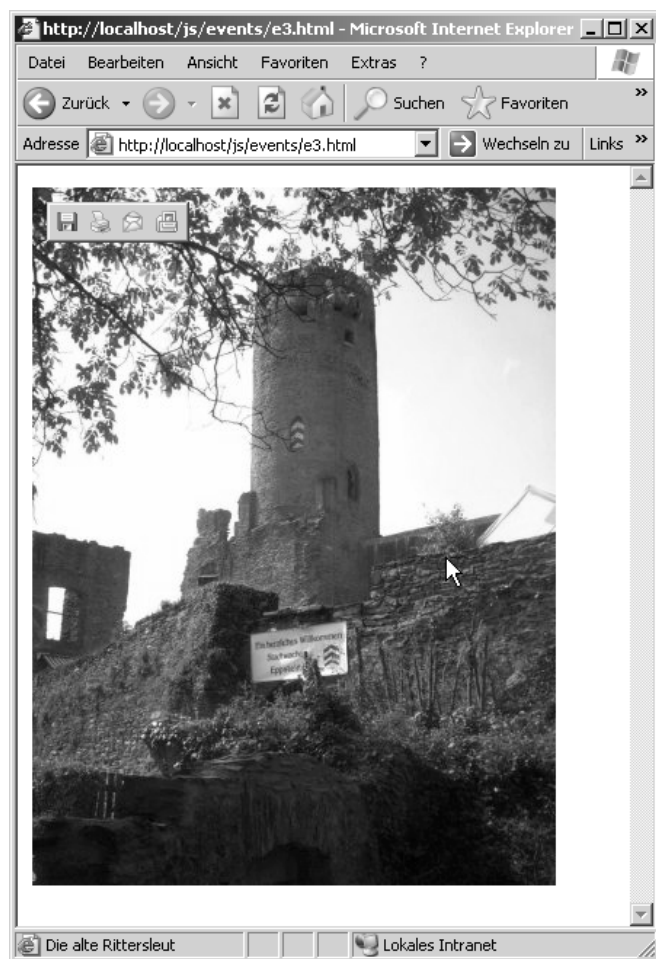


Abbildung 412: Beachten Sie die Statuszeile.

Achtung

Die Verwendung der beiden Eventhandler funktioniert auch bei einigen neueren Browsern (etwa neue Versionen vom Navigator als auch Mozilla) nur bei einem verweisen-sitiven Element (der Internet Explorer, Opera und der Konqueror kommen aber damit bei den meisten Tags zurecht). Wenn Sie allerdings die Eventhandler in einen Hyper-link-Tag notieren, sollten Sie keine Meldung in der Statuszeile (wie in dem Beispiel) ausgeben, denn diese Meldung wird unmittelbar durch die Anzeige des Hyperlink-URL überschrieben.

261 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

Grund-lagen

Core

Formu-lare

Aus-nahme

Arrays

Strings

Datum

Platt-form

DHTML

Events

Netscape unterstützt dieses Attribut bei HTML-Tags überhaupt nicht, sondern ursprünglich nur im Zusammenhang mit seinem Ereignis-Objekt (was aber in neuen Versionen nicht mehr unterstützt wird). Browser, die dem offiziellen Eventkonzept folgen, verstehen das Attribut jedoch bei allen gängigen HTML 4.0-Tags.

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.

262 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. Diese werden seit JavaScript 1.2 offiziell unterstützt. 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 Loslassens einer Taste bilden jeweils ein eigenes Ereignis. `onKeyPress`, `onKeyDown` und `onKeyUp` sind die jeweiligen Eventhandler. Wir verfolgen diese Eventhandler nicht weiter, da sie sowohl bei vielen Browsern problematisch sind, im Rahmen einer Webseite zu den absolut unüblichen Reaktionsvarianten zählen, als auch es schwierig ist zu bestimmen, auf welchem Element ein Tastaturereignis stattgefunden hat (es gibt ja keine x-/y-Koodinaten wie beim Mauszeiger). Von dem Einsatz dieser Eventhandler ist grundsätzlich abzuraten.

263 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 den `<area>`-Tag realisiert werden, oder Schaltflächen, die auch ohne Formulare funktionieren und extra für die Script-Ausführung vorhanden sind – der `<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 Ele-

menten reagieren. Dazu gibt es den Eventhandler `onBlur`. Für nähere Ausführungen als auch Beispiele zu diesen beiden Eventhandlern sei auf das Kapitel zu Formularen verwiesen.

264 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. Für nähere Ausführungen als auch Beispiele zu diesem Eventhandler sei auf das Kapitel zu Formularen verwiesen.

265 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`. Für nähere Ausführungen als auch Beispiele zu diesen Eventhandlern sei auf das Kapitel zu Formularen verwiesen.

266 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. Für weitere Ausführungen zu diesem Eventhandler sei auf das Kapitel zu Formularen verwiesen.

267 Wie kann ich einen Eventhandler explizit per JavaScript aufrufen?

Ein Eventhandler in der normalerweise verwendeten Form als HTML-Parameter gehört grundsätzlich zu HTML. Er kann aber auch als Funktionsreferenz über JavaScript ausgelöst werden. Schauen Sie sich bitte das nachfolgende Listing an.

Beispiel (*e4.html*):

```
01 <html>
02   <script LANGUAGE="JavaScript">
03     function ufktion() {
04       alert("Herzlich willkommen");
05     }
06   </script>
07 <body>
08   <form name="meinFormular">
09     <input type="button" name="meinButton" value="OK">
```

Listing 641: Expliziter Aufruf eines Eventhandlers aus JavaScript heraus als Funktionsreferenz

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

DHTML

Events

```
10 </form>
11 <script language="JavaScript">
12     document.meinFormular.meinButton.onclick = ufktion
13 </script>
14 </body>
15 </html>
```

Listing 641: Expliziter Aufruf eines Eventhandlers aus JavaScript heraus als Funktionsreferenz (Forts.)

Die in dem Beispiel verwendete Syntax beinhaltet eine Funktion (Zeile 3 bis 5), die in dem zweiten Script-Container 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 Scripts – das betrifft nicht die HTML-Attribute) muss vollständig in Kleinbuchstaben notiert werden (wie in dem Beispiel).

Hinweis

Bei den Rezepten zur globalen Ereignisbehandlung arbeiten wir in einigen Beispielen mit dieser Form der Eventhandler.

268 Wie kann ich globale Ereignisbehandlung in JavaScript realisieren?

Die Frage kann entweder ganz einfach beantwortet werden (lassen Sie es besser), oder aber man muss es richtig kompliziert aufziehen. Der Grund ist banal – im Grunde muss man für jeden Browser ein individuelles globales Eventhandling aufziehen und jeden Browser, den man unterstützen will, testen.

Es gibt im Konzept der Ereignisbehandlung ein spezielles Objekt mit Namen `event`, das eigens für die globale Ereignisbehandlung bereitsteht. Wenn man das tut, hat das weit reichende Konsequenzen. Die Ereignisüberwachung wird beispielsweise direkt in JavaScript programmiert. Die Verbindung von HTML (über die dort zugehörigen Eventhandler) und JavaScript in einer Zweckehe wird aufgelöst. Stattdessen wird von speziellen JavaScript-Routinen das Auftreten eines Ereignisobjekts überwacht. Je nach Art des Ereignisses kann der JavaScript-Interpreter darauf reagieren.

Wenn in einer Webseite ein Ereignis auftritt, wird dabei ein eigenes Objekt generiert – eben das `event`-Objekt, welches ein Unterobjekt von `window` ist. Das `event`-Objekt beinhaltet verschiedenste Informationen über ein aufgetretenes Ereignis in einer Webseite samt den Randbedingungen, welche dabei eine Rolle gespielt haben. 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.

In einer Webseite ist eine Situation, welche ein solches event-Objekt erzeugt, beispielsweise der Klick mit der Maus auf irgendeinen Bestandteil der Webseite. Ein Mausklick erzeugt ein event-Objekt, welches folgende Informationen enthält:

- Die verwendete Taste
- Eventuell gedrückte Zusatztasten (`[Strg]`, `[Alt]`, `[Alt-Gr]`, `[Shift]`)
- Die Koordinaten, wo der Klick erfolgte

Andere event-Objekte, die bei anderen Ereignissen erzeugt werden, beinhalten natürlich andere Informationen, welche dem Ereignis angepasst sind. So könnte etwa bei einem Tastendruck die gedrückte Taste als abzufragende Information bereitstehen.

Nun ist es sicher nicht Sinn und Zweck eines Ereignismodells, 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. Ein event-Objekt wird, wenn man unter JavaScript darauf reagieren will, an einen Mechanismus zum Behandeln von event-Objekten in Form einer Message (Botschaft) weitergereicht. Auf welche Art und Weise auf welches erzeugte Ereignisobjekt dann reagiert werden soll, wird durch ein abstraktes Modell geregelt – dem Ereignismodell. Darin sind sowohl die Verhaltensweisen auf Ereignisse festgelegt, die konkrete Syntax zur Umsetzung, aber auch erst recht diejenigen Ereignisse, welche überhaupt zur Kenntnis genommen werden.

Achtung

Die Ereignismodelle von Microsoft und Netscape sind vollkommen bis zum heutigen Zeitpunkt inkompatibel. Die meisten anderen Browser folgen dem Microsoft-Modell oder unterstützen sogar beide Varianten, aber ein ausführlicher Test ist dennoch zwingend, denn eine vollständige Unterstützung ist nicht gewährleistet.

269 Wie kann man das event-Objekt auswerten und darauf reagieren?

Da Ereignisobjekte während der Lebenszeit einer Webseite permanent im Hintergrund entstehen, stehen sie auch überall im Code zur Verfügung. Es gibt verschiedene Konzepte, auf welche Weise so ein event-Objekt angesprochen werden kann. Sie können es einfach über seinen Namen (`event`) ansprechen. Dabei müssen Sie noch angeben, von welchem Objekt aus das event-Objekt erzeugt wurde. Die Syntax dazu ist die in der objektorientierten Programmierung übliche Punktnotation:

```
[Objekt].event
```

Listing 642: Grundsätzliches Ansprechen des event-Objekts

Grundlagen

Core

Formulare

Ausnahme

Arrays

Strings

Datum

Plattform

HTML

Events

Beispiel:

```
document.event
```

Listing 643: Grundsätzliches Ansprechen des event-Objekts einer Webseite

Sie wollen bei einer Auswertung in der Regel dann natürlich eine ganz bestimmte Information eines Ereignisses verwenden. Dies sind bei einer objektorientierten Betrachtungsweise einfach die Eigenschaften des Objektes. So können Sie etwa einen Mausklick wie folgt ansprechen:

```
document.event.CLICK
```

Listing 644: Ansprechen eines Mausklicks

Diese Syntax steht vom Prinzip her sowohl im Ereignismodell von Microsoft als auch Netscape zur Verfügung. Dummerweise setzen sie aber die konkrete Implementierung völlig unterschiedlich um. Auch in der Syntax zur Überwachung von Ereignissen (wie schon bei der Unterstützung der Eventhandler) unterscheiden sich beide Browser. Und das, obwohl die grundsätzliche Logik gleich ist.

Sowohl das von Netscape entwickelte Objektmodell als auch das Microsoft-Objektmodell erlauben das globale Auffangen von `event`-Objekten. Sie können damit an zentraler Stelle in einer Webseite oder gar einem ausgelagerten Script behandelt werden. Dies ist sehr sinnvoll, wenn eine konsequente strukturelle Trennung einer Seite erreicht werden soll. Globales Eventhandling ist damit eine logische Konsequenz der Bemühungen, Layout, Inhalt und Skriptfunktionalitäten in einer Webseite zu entflechten.

270 Wie kann ich über das Netscape-Ereignismodell global auf Ereignisse reagieren?

Unter dem Netscape-Modell können 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 wird:

```
[Objekt].captureEvents([Ereignistyp])
```

Listing 645: 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 weitergeleitet – der zentralen Ereignisbehandlungsroutine. Dies ist in diesem Fall einfach eine Funktion, welche als Argument das erzeugte `event`-Objekt mit der gewünschten Eigenschaft

verwendet. Als Ereignistyp können die bei einem Objekt von Netscape unterstützten Eventhandler verwendet werden. Alternativ verwenden Sie einen Eventhandler als Funktionsreferenz.

Hinweis

Das Ereignismodell von Netscape ist in Zusammenhang mit einem allgemeineren Konzept zu sehen, mit dem Netscape dynamische Webseiten realisiert – dem Layer-Konzept (Schichten), das Netscape mit der Version 4.0 seines Browsers einführt und als Sprachelement von HTML versteht. Mit Hilfe des `<layer>`-Containers ist es möglich, beliebige Bereiche einer HTML-Datei exakt zu kontrollieren und Elemente zu positionieren. Neben der auf HTML beschränkten Funktion von Layern verwendet Netscape diese als Zielobjekt für Manipulationen unter JavaScript. Netscape hat jedoch ab der Version 6 vom Navigator sowohl die Unterstützung von Layern als auch ihr eigenes globales Eventmodell aufgegeben. Zwar meldet der Navigator beim Verwenden der alten Netscape-Technik keinen Fehler (wie etwa der Internet Explorer), aber er ignoriert die Anweisungen einfach. Leider muss daher grundsätzlich von der weiteren Verwendung der Technik abgeraten werden. Wir verfolgen das Layer-Konzept deshalb in dem Buch generell nicht weiter.

Allgemein können Sie im Netscape-Modell folgende Eigenschaften auswerten:

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 (Internet Explorer oder Opera) oder wird ignoriert (etwa im Mozilla und in neueren Navigator-Versionen).
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 außer dem Internet Explorer unterstützt.
screenX und screenY	Die Bildschirmkoordinaten. Die Eigenschaften werden in nahezu allen Browsern außer dem Internet Explorer unterstützt.
which	Der Tastaturcode bzw. der Maustastencode. Die Eigenschaften werden in nahezu allen Browsern außer dem Internet Explorer unterstützt.
type	Art des Ereignisses.

Tabelle 17: Eigenschaften, die im Netscape-Ereignismodell auszuwerten sind

Schauen wir uns zwei Beispiele an und wie verschiedene Browser darauf reagieren.

Beispiel (*e5.html*):

```
01 <html>
02 <script language="JavaScript">
03 window.onkeypress = taste;
04 function taste(ev)
05 {
06   status="Sie haben die Taste mit dem Wert " + ev.which + " gedrückt";
07 }
08 </script>
09 <body>
10 <h1>Drücken Sie eine Taste!</h1>
11 </body>
12 </html>
```

Listing 646: Globale Reaktion 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 (Zeile 3 – `window.onkeypress = taste;`). In der Funktion wird in der Statuszeile ausgegeben, welche Taste gedrückt wurde (der numerische Code über die Eigenschaft `which`).



Abbildung 413: In der Statuszeile wird der Wert der gedrückten Taste angezeigt – hier der Konqueror.

Schauen wir uns noch ein Beispiel mit der Auswertung anderer Eigenschaften des `event`-Objekts an.

Wie erwähnt, kommt der Internet Explorer mit dieser Art der globalen Ereignisbehandlung nicht zurecht.

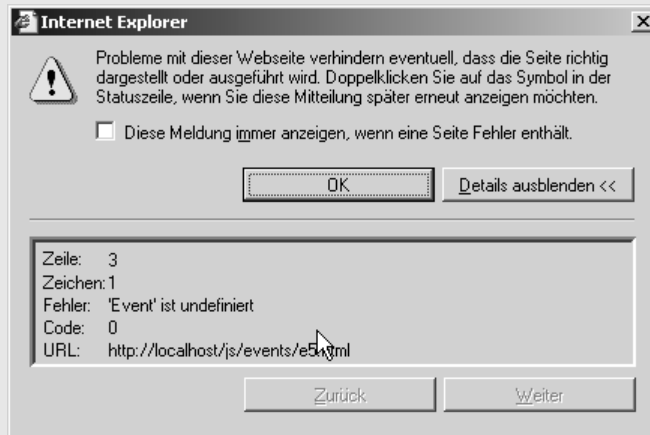


Abbildung 414: Der Internet Explorer bringt es einfach nicht.

Der Netscape Navigator, Mozilla, der Opera und der Konqueror kommen mit dem Beispiel im Prinzip zurecht.

Interessanterweise funktioniert das letzte Beispiel nicht im Netscape Navigator 7.x oder Mozilla. Das Problem ist aber nicht, dass das Ereignis nicht ausgewertet wird. Wenn Sie die Zeile 7 wie folgt austauschen, sehen Sie, dass die Browser das Ereignis schon wahrnehmen (*e5a.html*):

```
07 alert("Sie haben die Taste mit dem Wert " + ev.which + " gedrückt");
```

Listing 647: Austausch der Zeile durch eine alert()-Meldung

Das Problem ist die Ausgabe in der Statuszeile. Der Mozilla (nicht der Navigator) gibt in der Statuszeile aber einen Hinweis, wo das Problem liegt. Aber der nutzt Ihnen auch nicht viel. Bei der dynamischen Ausgabe in der Statuszeile macht der Netscape Navigator und sein Mozilla-Bruder allgemein Probleme.



Abbildung 415: In der Statuszeile des Mozillas deutet die seltsame Meldung das Problem bei der Ausgabe in der Statuszeile an – hier wurde die g-Taste gedrückt.

Beispiel (*e6.html*):

```

01 <html>
02 <script language="JavaScript">
03 function pos(ev)
04 {
05   alert("X-Koordinate: " + ev.pageX + ", Y-Koordinate " + ev.pageY +
06     "\nBildschirm: X-Koordinate: " + ev.screenX + ", Y-Koordinate " + ev.screenY +
07     "\nTyp: " + ev.type);
08 }
09 document.onmouseup = pos;
10 </script>
11 <body>
12 </body>
13 </html>

```

Listing 648: Eine weitere Variante mit globaler Reaktion nach dem Ereignismodell von Netscape

In Zeile 9 wird der Eventhandler `onmouseup` (JavaScript-Zugriff) überwacht und die Funktionsreferenz auf die Funktion `pos()` referenziert. In der Funktion werden verschiedene Ereignisse des Ereignisobjekts ausgegeben.

271 Wie kann ich über das Microsoft-Ereignismodell global auf Ereignisse reagieren?

Das Microsoft-Objektmodell erlaubt den unmittelbaren Zugriff auf das `event`-Objekt samt dessen Ereignissen, soweit diese im Microsoft-Ereignismodell verstanden werden. Netscape und Mozilla-Browser kommen explizit nicht mit dem Zugriff in der Form zurecht. Allerdings verstehen sowohl der Opera als auch der Konqueror auch diese Variante der globalen Ereignisbehandlung.

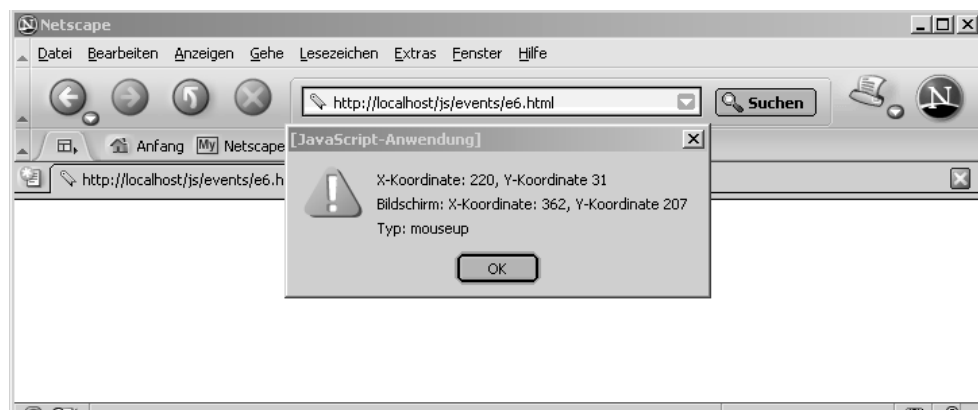


Abbildung 416: Verschiedene Eigenschaften des Ereignisobjekts – hier der Navigator 7.1

Eigenschaft	Beschreibung
altKey ctrlKey shiftKey	Die entsprechenden Sondertasten. Sie stehen für die Zusatztasten ALT , CRTL oder SHIFT . Wenn sie gemeinsam mit einer anderen Taste oder einem Mausklick gedrückt wurden, werden die jeweiligen Eigenschaften den Wert true enthalten.
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 koordinatenabhängiges Ereignis (etwa eine Mausektion) ausgelöst wurde.
keyCode	Der Tastaturcode. Die Eigenschaft speichert bei Tastaturereignissen den dezimalen Code (ASCII-/ANSI-Wert) der gedrückten Taste.
offsetX offsetY	Die Koordinaten relativ zum Objekt. Über die Eigenschaften kann auf die horizontalen Pixel (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 Pixel (x) und die vertikalen Pixel (y) der Cursorposition relativ zur oberen linken Ecke des Elternelements von dem Element, das ein Ereignis ausgelöst hat. Wenn ein absolut positionierter Bereich das Elternelement ist, ist dessen obere linke Ecke der Bezugspunkt. Wenn das auslösende Element sonst kein Elternelement hat, gilt die linke obere Ecke des Dokuments als Koordinatenursprung.

Tabelle 18: Eigenschaften, die im Microsoft-Ereignismodell auszuwerten sind

Schauen wir uns ein Beispiel an, in dem die Position des Mauszeigers beim Überstreichen von Grafiken ausgegeben wird.

Beispiel (e7.html):

```

01 <html>
02   <script language="JavaScript">
03     <!--
04     function pos() {
05       status="Bildschirmkoordinaten X-Position: " + event.clientX +
06         ", Y-Position: " + event.clientY;
07     }
08     function pos1() {
09       status="Position relativ zum Elternelement - X : " + event.x +
10         ", Y-Position: " + event.y;
11     }
12     function loesche() {
13       status="";
14     }
15     //-->
16   </script>
17 <body>
18   
19   
20 </body>
21 </html>

```

Listing 649: Einsatz von `onMouseOver` und Koordinatenangaben des `event`-Objekts

Das Beispiel gibt beim Überstreichen von Grafik eins in der Statuszeile die aktuelle x- und y-Position des Mauszeigers aus, wenn dieser in dem Bereich der Grafik bewegt wird. Einmal auf der ersten Grafik als Bildschirmkoordinaten (Zeile 6) und auf der zweiten Grafik als relative Koordinaten zum Elternelement. Verlässt der Mauszeiger die Grafik, wird die Statuszeile gelöscht (Zeile 8).

Abbildung 417: Die Auswertung von `clientX`, `clientY`, `x` und `y` – hier im Opera unter Linux

Die alternative globale Ereignisbehandlung im Microsoft-Ereignismodell erfolgt mittels der folgenden Syntax:

```
<script FOR=[Objekt] EVENT=[Ereignis] language="JavaScript">
    ...[Konkrete Anweisungen]...
</script>
```

Listing 650: Eine weitere Variante der globalen Ereignisbehandlung im Microsoftmodell

Als [Ereignis] geben Sie einfach die Eventhandler an (allerdings auf jeden Fall vollkommen kleingeschrieben!). 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 Internet Explorer kennt. Das gesamte Prozedere ist unkompliziert, solange kein Browser die Seite lädt, der damit nicht klar kommt.

Hinweis

Browser, die bei dieser Version vom Microsoft-Modell mit Fehlermeldungen reagieren, sind alte Netscape-Versionen oder alte Opera-Browser. Der Opera kommt aber in der Version 6 gut mit dem event-Objekt zurecht, wird aber bei der speziellen Erweiterung von dem <script>-Container Schwierigkeiten haben. Einmal versteht er ihn als normalen Container. Das bedeutet, alle darin enthaltenen Anweisungen werden beim Laden der Webseite ausgeführt. Zum anderen werden die Ereignisse selbst beim Auftreten nicht ausgeführt. Neue Netscape-Browser melden zwar keine Fehler, üben sich jedoch grundsätzlich in vornehmer Ignoranz oder führen ebenso Anweisungen beim Laden der Seite aus. Der Grund ist, dass die Angabe `FOR=document` für den Internet Explorer festlegt, dass die zentrale Behandlung für die gesamte Webseite gilt, `EVENT=onclick` spezifiziert etwa das Ereignis Mausklick. Beachten Sie bei der Spezifizierung des Ereignisses unbedingt die Groß- und Kleinschreibung! Für alle Browser außer dem Internet Explorer oder streng kompatibler Browser sind diese Angaben unbekannt und werden ignoriert. Dementsprechend wird der Container nicht als verschieden von einem normalen Scriptcontainer zu erkennen sein, und die darin enthaltenen Anweisungen werden abgearbeitet – als würde dort ein gewöhnliche Scriptcontainer stehen. Der Internet Explorer unterscheidet die Container – deshalb führt er die im Container notierten Anweisungen nicht beim Laden der Seite aus.

Schauen wir uns ein Beispiel an (*e8.html*).

```
01 <html>
02 <script FOR=document EVENT=onclick language="JavaScript">
03   alert("Shift: " + event.shiftKey + " Alt: " + event.altKey +
04     " Ctrl: " + event.ctrlKey);
05 </script>
06 <body>
07   
```

Listing 651: Auswertung der Zusatztasten beim Klick

```
08 </body>
```

```
09 </html>
```

Listing 651: Auswertung der Zusatztasten beim Klick (Forts.)

In dem Beispiel wird die gesamte Webseite überwacht, ob ein Klick erfolgt. Falls ja, zeigt das Beispiel alle dabei gedrückten Sondertasten an.



Abbildung 418: Alt und Shift waren gedrückt.

Stichwortverzeichnis

!

\$1

 RegExp 561

<a> 552

 552

<big> 552

<blink> 552

<body> 31, 677

<button> 184

<doctype> 34

 552

<form> 172

 enctype 180

 name 180

 target 180

<frame> 742

<frameset> 741

<head> 31

<i> 552

 onLoad 810

<input> 181

<layer> 821

<link> 785

<meta http-equiv= 91

<meta> 31, 663

<noframes> 743

<noscript> 84

<option> 200, 202, 204, 205, 251

<pre> 502

<script> 79

<select> 200, 202, 204, 205, 251

<small> 552

 436

<strike> 553

<style> 784

<sub> 553

<sup> 553

<textarea> 198

<title> 700

<tt> 552

@

 E-Mail-Test 360

 _blank 743

 _parent 743

 _self 743

 _top 743, 746

A

a

 hover 794

Ablaufdatum

 Inhalt einer Webseite 642

abweisende Schleife 56

action

 Formulare 172

 JavaScript-Zugriff 221

Active Server Pages 23

Adresszeile

 Zugriff per JavaScript 716

Aktivierung eines Elements 816

Aktualisierungsdatum

 dynamisch Schreiben 649

alert() 206, 702

align

 style 761

alink 677

alinkColor

 document 677

all 62, 95, 756

Altersbestimmung 635

anchor 62

anchor() 552

anchors 64, 682

Animationen 755, 765

 Austausch von Grafiken 777

 Bildverschiebung 799

 mit JavaScript generiert 803

 Veränderung der Größe einer Grafik 781

animierte GIFs 657

Anker 552

 Zugriff über JavaScript 682

annehmende Schleife 56

Anweisungen 54

appName 97

applet 62

- Applets 23
 - Zugriff über JavaScript 684
- applets 64, 684
- application/x-www-form-urlencoded 175, 180, 228
- appName 93
- appVersion 97
- Argumenten-Array
 - Funktionsparameter 146
- Argumentendatenfeld 145
- arithmetische Operatoren 48
- arithmetische Zuweisungsoperatoren 49
- Array 42, 62, 465
- ASCII-Kodierung
 - Zeichen 546
- ASP 23
- assoziative Datenfelder 472
- assoziierte Datenfelder 472
- Attribute 58
- Aufruf von JavaScripts 64
- Ausdrucksanweisungen 54
- Ausdrücke 47
- Ausnahmehandling 443
- Ausnahmen 444
 - mehrere Ausnahmen behandeln 451
 - selbstdefinierte 451
- Auswahlliste
 - Formulare 200
 - Mehrfachauswahl 204
 - Selektion von Einträgen 271
 - vorselektieren 205
 - Zugriff auf die Anzahl der Auswahlmöglichkeiten 259
 - Zugriff auf die Beschriftung 256
 - Zugriff auf einen Wert 254
 - Zugriff auf selektierten Eintrag 261
 - Zugriff aus JavaScript 251
- availHeight 100
- availLeft 100
- availTop 100
- availWidth 100
- B**
- back()
 - history 731
- backgroundColor
 - style 761
- backgroundImage
 - style 761
- Benutzereingaben 169
- Bezeichner
 - Variablen 38
- bgColor
 - document 677
- bgcolor 677
- big() 552
- Bilder
 - automatisch einbinden 657
 - beim Laden eine Anweisung ausführen 810
 - dynamisch austauschen 773
 - Erzeugen mit JavaScript 770
 - Höhe und Breite dynamisch verändern 772
 - Kontrolle, ob bereits geladen 810
 - Unterbrechung des Ladevorgangs 810
 - Veränderung der Größe einer Grafik 781
- Bildschirmauflösung 100
- bitweise Operatoren 53
- blink() 552
- Blockanweisungen 54
- blur() 264
- Body
 - Webseite 31
- bold() 552
- Boole'sche Werte 39
- Boolean 39, 62
- border
 - images 770
- borderBottomWidth
 - style 761
- borderColor
 - style 761
- borderLeftWidth
 - style 761
- borderRightWidth
 - style 762
- borderStyle
 - style 762
- borderTopWidth
 - style 762
- Botschaft 58
- break 55
- Breite
 - Webseite 693
- Browserabfrage
 - Spracheinstellung 99
 - Typ 93
 - Version 97

- Browserfenster
 - dynamisch in der Größe verändern 726
 - dynamisch positionieren 724
 - Inhalt dynamisch scrollen 727
 - öffnen 719
 - schließen 728
 - Test, ob geschlossen 729
- Browserweiche 112
- Build-In-Objekte 57
- button
 - <input> 184

C

- captureEvents() 69, 820
- Cascading Style Sheets 783
- case 56
- catch 446
- CDATA 34
- charAt() 358, 543
- charCodeAt() 358, 544
- Checkbox 190
 - Zugriff auf den Selektionszustand 250
- checkbox
 - <input> 190
- checked 250
 - Kontrollkästchen 268
 - Optionsfelder 267, 268
 - Radiobutton 192
- class 789
- className 790
 - all 757
 - Packages 689
- clear
 - style 762
- clearTimeout() 715
- click() 276
- close()
 - document 675
 - window 728
- closed 729
- color
 - style 762
- colorDepth 102
- compile()
 - RegExp 560
- complete
 - images 770
- concat() 358
 - Datenfelder 520, 523

- Strings 546
- confirm() 207, 704
- contextual() 767
- continue 55
- cookie
 - document 732
- cookie.dat 732
- Cookies 732
 - Test auf Aktivierung 736
- cookies.txt 732
- CSS 783
 - Prioritäten 789
- CSS-Deklarationen
 - grundsätzliche Syntax 786
- current
 - history 731
- CursorPosition 825

D

- dataFld
 - all 757
- dataFormatAs
 - all 757
- dataPageSize
 - all 757
- dataSrc
 - all 757
- Date 62, 597
- date
 - <input> 192
- Date() 597
- Dateiauswahlfenster 195
- Dateierweiterung
 - externe JavaScriptdatei 82
- Datenfeldelemente
 - Zugriff 470
- Datenfelder 42, 465
 - Anfügen von Elementen 514
 - anonymer Zugriff 469
 - Anzahl der Elemente 477
 - assoziative 472
 - assoziierte 472
 - Datenfeldelement entfernen 506
 - Elemente am Anfang einfügen 521
 - Elemente an einer beliebigen Stelle einfügen 525
 - erzeugen 466
 - Extrahieren von Teilen 516
 - Größe festlegen 482

- Inhalte zu einem String verbinden 505
- mehrdimensional 484
- Minimum oder Maximum 527
- multidimensional 484
- sortieren 490
- Sortierung umdrehen 503
- zufällig mischen 531
- Zugriff 468
- Zugriff auf die Elemente 469
- Zugriff über einen String-Index 472
- zusammenfügen 519
- Datenkapselung 685
- Datentypen 38
- Datum
 - Berechnung eines Termins in der Zukunft 646
 - Berechnung regelmäßiger Termine 647
- Datumsangaben
 - Differenz 635
 - rechnen 635
- Datumsfeld
 - Formular 192
- Datumsobjekt
 - Altersbestimmung 635
 - erzeugen 597
 - Summenberechnungen 646
 - vorgegebenes Datum 599
- Datumsoperationen 597
- defaultStatus 708
- Deklaration
 - Funktionen 45
- Deklarationsanweisungen 54
- Dekodieren
 - Zeichenketten 165
- Dekrement-Operator 49
- DeleteCookie() 733
- dependent
 - open() 723
- DHTML 431, 755, 783
- Dialogfenster
 - freie Benutzereingabe 705
- directories
 - open() 722
- display
 - style 762
- DNS 365
- document 62, 670
 - contextual() 767
 - Layoutinformationen 677
 - tags 767
- Document Object Model 211
 - document.anchors 682
 - document.applets 684
 - document.close() 675
 - document.cookie 732
 - document.getSelection() 701
 - document.height 693
 - document.lastModified 695
 - document.links 696
 - document.location 697
 - document.open() 675
 - document.plugins 699
 - document.title 700
 - document.URL 697
 - document.width 693
 - document.write() 671
- Dokument 669
 - explizit öffnen und schließen 675
- DOM 57, 59, 211, 756
- Domain 365
- domain
 - Cookies 733
- Domain Name System 365
- Doppelklick 814
- Dortch, Bill 733
- DOT-Notation 58
- do-while 56
- Drucken
 - Webseite 711
- DTD-Anweisungen 32
- Dynamic-HTML 755
- dynamisch positionieren
 - Browserfenster 724
- dynamisches Schreiben von Inhalten
 - datumsabhängig 651
- dynamisches Verändern des Layouts
 - mit Style Sheets
 - datumsabhängig 654
- E**
 - ECMA 25
 - ECMA-262-Norm 25, 36
 - ECMAScript 25, 36, 445
 - Eigenschaften 58
 - Einbindung 79
 - Eingabefeld
 - einzeilig 181
 - mehrzeilig 198
 - eingebettete Formatvorlagen 784

einzeiliges Eingabefeld 181
Eingabe eines Kalenderdatums 192
Eingabe eines URLs 194
Eingabe von Dezimalkommazahlen 193
Eingabe von ganzen Zahlen 193
maximale Anzahl an angezeigten
Zeichen 181
maximale Anzahl der einzugebenden
Zeichen 181
minimale Anzahl der einzugebenden
Zeichen 182
nur Werte größer als ein vorgegebener
Grenzwert 195
nur Werte kleiner als ein vorgegebener
Grenzwert 194
Vorgabewert 182, 334
elements 64, 235
else 55
E-Mail
Formulardaten versenden 173
Test auf Gültigkeit mit regulären
Ausdrücken 586
E-Mail-Adresse
Gültigkeit überprüfen 359
encoding 227
enctype 174, 227
Formulare 180
EndOfFile 178
Environment-Variablen
Formulare 177
EOF 178
Ereignisbehandlung 65, 807
global 67, 818
grundsätzliche Vorgänge 807
Ereignisse 807
Error 445
escape() 165, 733, 737
Escape-Sequenz 167
eval() 443
EvalError 445
event 62, 804, 818
auswerten und darauf reagieren 819
event.clientX 805
event.clientY 805
EventHandler 65
Aufruf aus JavaScript 817
expliziter Aufruf aus JavaScript 66
grundsätzliche Syntax 808
Eventhandling 65

event-Objekt 67
Events 807
Exceptions 443, 444
exec()
RegExp 560, 582, 584
expires
Cookies 732
Extensible HyperText Markup Language 32
Extensible Markup Language 32
externe JavaScript-Datei 79, 82

F

Fakultät 152
Fall-Through-Anweisung 56
false 39
Farbeinstellung 102
Fehlertoleranz
Prinzip 27
Fenster 669
Fensternamen
reservierte 743
fgColor
document 677
file
<input> 195
fixed() 552
float
<input> 193
focus() 266
Selektion von Schaltflächen 271
fontcolor() 552
fontFamily
style 762
fontSize
style 762
fontsize() 552
fontStyle
style 762
fontWeight
style 762
for 56
for...in 56
Datenfelder 475
form 62, 229, 245
Formatvorlagen 784
forms 64
Formular 169
abschicken 290
Anzahl der Elemente 214

- Aufruf von JavaScript-Funktionen 277
- Auswahlliste 200
- HTML-Grundlagen 170
- Kontrolle beim Verlassen 313
- mehrzeilige Auswahlliste 202
- Name 215
- nur plausibilisiert abzuschicken 439
- Reaktion auf das Absenden 817
- Reaktion auf das Zurücksetzen 817
- target 223
- Versandmethode 216
- Zugriff auf die Kodierung 227
- zurücksetzen 295
- Formularcontainer
 - HTML 172
- Formulardaten
 - abschicken 185
 - abschicken mit einer Grafik 186
 - abschicken ohne submit-Button 229
 - mailto 173
 - verschicken mit einer Grafik 230
 - zurücksetzen 189
 - zurücksetzen ohne Reset-Button 234
- Formularelement
 - aus JavaScript selektieren und deselektieren 267
 - Fokus entziehen 264
 - Fokus geben 266
 - JavaScript-Zugriff 235
 - JavaScript-Zugriff auf den Wert 238
 - Selektion 272
 - Simulation eines Anwenderklicks 276
 - Typbestimmung 247
 - verlassen 286
 - Zugriff auf das umgebene Formular 245
 - Zugriff auf den Namen 242
 - Zugriff über this 243
- Formularfeld
 - E-Mail-Test 359
 - Test auf einen Inhalt 359
 - versteckt 197
- Formularmailer 176
- Formularschaltfläche 184
- frame 62
- Frames 672, 741
 - gleichzeitig mehrere Frames aktualisieren 747
 - JavaScript anwenden 746
 - Namen per JavaScript abfragen 751
 - Schreiben in ein anderes Frame 752
 - Verweise zu anderen mit HTML 744
- frames 64
 - window 746
- Frameset
 - Formular 224
 - Verhindern, dass eine Seite in einem Frameset angezeigt wird 753
- freie Texteingabe
 - absichern mit dem Ausnahmekonzept 459
 - bestimmten Inhalt erzwingen 357
 - Eingabe eines Kalenderdatums 380
 - Eingabe eines URLs 371
 - maximale Anzahl an angezeigten Zeichen 324
 - maximale Anzahl der einzugebenden Zeichen 299
 - minimale Anzahl der einzugebenden Zeichen 317
 - nur die Eingabe von Dezimalkommazahlen 336
 - nur die Eingabe von Ganzzahlen 344
 - nur Werte größer als ein vorgegebener Grenzwert 353
 - nur Werte kleiner als ein vorgegebener Grenzwert 347
 - Verhindern bestimmter Zeichen 387
 - Vorgabewert 334
- Fremdinhalte einbinden 667
- fromCharCode() 547
- Function 62
- function 45
- Funktion 45
 - Defaultwerte für Parameter 148
 - Deklaration 45
 - rekursiver Aufruf 152
 - Rückgabewert 46
 - Werte übergeben 145
- Funktionsaufruf 47
- Funktionsreferenzen 67
- fußgesteuerte Schleife 56
- G**
 - Ganzzahlwerte 39
 - geklammerte Teile
 - reguläre Ausdrücke 584
 - GET 177
 - GetCookie() 733

- getDate() 602
- getDay() 604
- getElementById() 792
- getHours() 607
- getMinutes() 609
- getMonth() 619
- getProperty()
 - Java 692
- getSeconds() 611
- getSelection() 701
- Getter-Methoden 133
- getTime() 623
- getTimezoneOffset() 626
- getYear() 612
- Gleitkommawerte 39
- globale Ereignisbehandlung 818
- GMT 626
- grafische Submit-Schaltfläche
 - Formular 186, 190
- Greenwich Meridian Time 626
- Großbuchstaben
 - Strings 541
- Großschreibung
 - HTML 26
- Grundgerüst
 - Webseite 30
- Gültigkeitsbereich
 - Variablen 142

H

- Header
 - Webseite 31
- height 100
 - document 693
 - images 771
 - open() 721
- hidden
 - <input> 197
- History
 - Anzahl der Einträge 730
- history 62, 730
- history.back() 731
- history.length 730
- Höhe
 - Webseite 693
- home() 706
- hotkeys
 - open() 723
- hover

- Style Sheets 794
- Hover-Effekt 794
- href
 - location 718
- hspace
 - images 771
- HTML 22
 - Formular 170
 - Grundlagen 26
 - Kommentar 31
- HTML-Formatierungen
 - auf Strings anwenden 550
- HTML-Kommentar 80
- HTTP Request Header 177
- Hyperlinks
 - Zugriff über JavaScript 696

I

- id
 - all 757, 758
- IETF 624, 627
- if 55
- if-else-Operator 52
- image 62
 - <input> 186
- Image() 770
- images 64, 658, 770
- index
 - exec() 582
- indexOf() 95, 358, 556
- Infinity 125
- Information Hiding 58
- Initialwert
 - Arrays 467, 480, 483
- Inkrement-Operator 49
- Inline-Referenz 79, 83
 - Style Sheets 784
- innerHeight
 - open() 723
- innerHTML
 - all 757
- innerText
 - all 757
- innerWidth
 - open() 723
- input
 - exec() 582
- instanceof 457

- int
 - <input> 193
- Interaktion
 - JavaScript 206
- IP-Adresse 364
- IP-Nummer 364
- IPv4 364
- isFinite() 125
- isNaN() 127
- ISO 5589-1 390
- isTextEdit
 - all 757
- italics() 552
- J**
- Jahreszahl
 - extrahieren aus einem Datumsobjekt 612
 - setzen in einem Datumsobjekt 618
- Java 23
 - mit JavaScript verwenden 684
- java
 - LiveConnect 687
- Java Server Pages 23
- Java Servlets 23
- Java virtual Machine 104
- java.lang.System 692
- Java-Applets 23
 - Zugriff über JavaScript 684
- JSONArray
 - LiveConnect 687
- javaEnabled() 105
- JavaScript
 - LiveConnect 687
- JavaPackage
 - LiveConnect 687
- JavaScript
 - aufrufen 64
 - Geschichte 22
 - Grundlagen 34
 - Test, ob aktiviert 90
 - Version 84
 - Versionszyklen 25
- JavaScript-Kommentar 81
- Java-Unterstützung 104
- Java-Zugriff 687
- join() 505
- Jokerzeichen
 - Frames 742
- JScript 24
- JSObject 689
- JSP 23
- JVM 104
- K**
- Kalenderdatum
 - Gültigkeit 380
- Klasse 59
 - erweitern 159
 - Style Sheets 787
- Kleinschreibung
 - HTML 26
- Klick auf eine Referenz 811
- Kodieren
 - Zeichenketten 165
- Kodierung
 - Formular 227
- Kommentar
 - HTML 30, 80
 - JavaScript 81
- konditionaler Operator 52
- Konstruktor 59
 - Arrays 466
- Konstruktormethode 155
- Kontrollfelder
 - Formular 190
- Kontrollflussanweisungen 55
- Kontrollkästchen
 - Zugriff auf den Selektionszustand 250
- kopfgesteuerte Schleife 56
- L**
- Laden einer Webseite 808
- lang
 - all 757
- language 99
 - all 757
- lastIndexOf() 358, 558
- lastModified
 - document 695
- Laufschrift
 - Statuszeile 710
- Layer 756, 775
 - Ereignismodell 821
- layer 62, 775
- Layer-Konzept 821
- layers 95
- layerX 821
- layerY 821

- Layoutinformationen
 - document 677
- length
 - all 757
 - Array 43, 44
 - Datenfelder 477
 - Formular 214
 - history 730
 - images 771
 - Strings 539
- lineHeight
 - style 762
- link 62
- link() 552
- linkColor
 - document 677
- links 64, 696
- listStyleType
 - style 762
- Literal 38, 129
- LiveConnect 25, 687
- LiveScript 23
- location 62, 716
 - document 697
 - open() 721
- location.href 92, 718
- location.reload() 718
- location.search 737
- logische Vergleichsoperatoren 51
- Lottospiel 531
- lowsrc
 - images 771
- M**
- mailto
 - Formular 173
- marginBottom
 - style 762
- marginLeft
 - style 762
- marginRight
 - style 762
- marginTop
 - style 762
- Maskieren
 - Zeichen in Strings 167
 - Zeichenketten 165
- Maskierung 738
- match() 588
 - reguläre Ausrücke 560
- Math 63, 491, 550
- Math.max() 528
- Math.min() 528
- Math.random() 125, 491, 515, 517, 520, 522, 530, 534
- Math.round() 126, 491, 504, 550
- Mausbewegungen 815
- Mausklick 814
- Mausposition
 - Element positionieren 804
- Mauszeigerposition 826
- max
 - <input> 194
- max() 527
- maxlength
 - <input> 181
- Mehrfachauswahl
 - Auswahlliste 204
- mehrzeilige Auswahlliste
 - Formulare 202
- mehrzeiliges Eingabefeld 198
- menubar
 - open() 721
- Meridian 626
- method
 - Formulare 177, 216
 - JavaScript-Zugriff 216
- Methoden 45, 58
- Microsoft
 - globales Ereignismodell 824
- Millisekunden
 - extrahieren aus einem Datumsobjekt 623
 - setzen in einem Datumsobjekt 625
- MIME-Typ
 - JavaScript 79
- Mime-Typ 33
- MIME-Type 109
- contentType 63, 64, 109
- min
 - <input> 195
- min() 527
- Minuten
 - Extrahieren aus einem Datumsobjekt 609
 - Setzen in einem Datumsobjekt 610
- Mitteilungsfenster anzeigen 702
- Mocha 23
- modifiers 821

- Modulooperator 48
- Monat
 - Extrahieren aus einem Datumsobjekt 619
 - Setzen in einem Datumsobjekt 622
- Monatstag
 - Extrahieren aus einem Datumsobjekt 602
 - Setzen in einem Datumsobjekt 603
- moveBy() 724
- moveTo() 724
- multiple
 - Auswahllisten 204
- N**
- Name
 - Fenster 699
- name 242
 - Formular 215
 - images 771
 - window 699
- NaN 127
- navigator 63, 93
 - navigator.appCodeName 97
 - navigator.appName 93
 - navigator.appVersion 97
 - navigator.javaEnabled() 105
 - navigator.language 99
 - navigator.mimeType 109
 - navigator.plugins 106
 - navigator.userAgent 585
- Netscape 23
 - globales Ereignismodell 820
- netscape
 - LiveConnect 687
- new 42, 59, 155
 - Datenfelder 466
- new Image() 770
- next
 - history 731
- null 39, 209
- Number 39, 63
- Number() 137
- numerischer Ausdruck
 - Test 127
- O**
- Oak 23
- Object 39
- Objekt 57
 - eigenes erzeugen 156
 - erweitern 159
 - Inhalt ausgeben 164
- Objektdaten 58
- Objektdeklaration 59
- Objektfelder 63, 465
- Objektinstanz 155
- Objektmethoden 58
- Oder-Verbindung
 - reguläre Ausdrücke 589
- Offscreen 657
- Offscreen-Bereich 435, 799
- offsetHeight
 - all 757
- offsetLeft
 - all 757
- offsetParent
 - all 757
- offsetTop
 - all 757
- offsetWidth
 - all 757
- oktale Darstellung 39
- onAbort 810
- onBlur 403, 817
 - Formulare 286
- onChange 289, 817
 - Formulare 289
- onClick 811
 - Formulare 277
- onDbClick 814
- onFocus 403, 816
 - Formulare 282
- onKeyDown 300, 816
- onKeyPress 300, 816
- onKeyUp 300, 816
- onLoad 808
- onMouseDown 814
- onMouseMove 815
- onMouseOut 814
- onMouseOver 814
 - Formulare 281
- onMouseUp 814
- onReset 295, 817
- onSelect 289, 817
- onSubmit 225, 290, 313, 817
- onUnload 809
- open() 719, 721, 729
 - document 675
 - Frames 748

- Operanden 47
- Operatoren 47
 - bitweise 53
 - Priorität 53
- Operatorprioritäten 53
- Option
 - Konstruktor 259
- options 64, 252
- Optionsfeld 191
 - Zugriff auf den Selektionszustand 250
- outerHTML 757
- outerText
 - all 757

P

- Packages
 - LiveConnect 687
- paddingBottom
 - style 762
- paddingLeft
 - style 762
- paddingRight
 - style 762
- paddingTop
 - style 762
- pageX 821
- pageY 821
- Parameter
 - HTML 29
- parent 671, 720
 - Frames 747
- parentElement
 - all 757
- parentTextEdit
 - all 757
- parse() 624
- parseFloat() 98, 132, 341, 345, 585
- parseInt() 98, 132, 341, 345, 585
- password
 - <input> 183
- Passwortfeld 183
- Passwortüberprüfung
 - Java 684
- path
 - Cookies 733
- Pattern
 - reguläre Ausdrücke 561
- Pflichtfeld
 - Webformular 355

- PHP 24
- Pipe-Symbol
 - reguläre Ausdrücke 589
- Plausibilisieren
 - Webformular 400
- Plug-In 24, 105
 - Zugriff über JavaScript 699
- plugin 63
- plugins 64, 106
 - document 699
- pop() 506
 - Datenfelder 506
- Positionieren
 - Elemente der Webseite 795
 - Verschieben mit JavaScript 797
- POST 177
- previous
 - history 731
- print() 711
- Prinzip der Fehlertoleranz 27
 - HTML 81
- Priorität
 - Operatoren 53
- prompt() 209, 705
 - Eingabe eines Kalenderdatums 380
 - Eingabe eines URLs 371
 - E-Mail-Adresse prüfen 360
 - maximale Anzahl der einzugebenden Zeichen 300
 - minimale Anzahl der einzugebenden Zeichen 317
 - nur die Eingabe von Dezimalkommazahlen 336
 - nur die Eingabe von Ganzzahlen 344
 - nur Werte größer als ein vorgegebener Grenzwert 353
 - nur Werte kleiner als ein vorgegebener Grenzwert 347
 - Vorgabewert 335
- prototype 160
- Prototyping 160
- Prozeduren 45
- Pseudo-URL 737
 - Formulare 177
- Punkt-Notation 58
- push()
 - Datenfelder 515

R

radio

Formular 191

Radiobutton 191

Zugriff auf den Selektionszustand 250

random() 125, 491, 515, 517, 520, 522, 530, 534

Random-Verfahren 531

RangeError 446

Reaktion

auf Änderungen 817

auf Textselektion 817

recordNumber

all 757

ReferenceError 446

refresh

<meta>-Tag 663

RegExp 63, 539, 561

reguläre Ausdrücke 99, 316, 344, 352, 358, 389, 560

Rekursion 765, 779

rekursiver Aufruf 152

reload() 718

replace() 358, 559

reguläre Ausdrücke 588

reguläre Ausrücke 560

Request

Formulare 177

reservierte Fensternamen 743

reset

<input> 189

reset() 234, 295

Reset-Schaltfläche

Formular 189

resizable

open() 721

return 55, 149

bedingtes 149

reverse() 493, 503

Rollover 775

round() 126, 491, 504, 550

Rückgabewert

Funktion 46, 149

S

Schaltflächen

Selektion 269

Schaltjahr

Berechnung 381, 632

schleifenlokale Variablen 143

Schlüsselwörter 35

Schreibkonventionen 21

scope

Variablen 142

screen 63, 100

screen.colorDepth 102

screen.height 100

screen.width 100

screenX 821

open() 723

screenY 821

open() 723

scrollbars

open() 721

scrollBy() 727

scrollTo() 727

search() 95, 358, 556

Reguläre Ausrücke 560

secure

Cookies 733

Sekunden

Extrahieren aus einem Datumsobjekt 611

Setzen in einem Datumsobjekt 611

select()

Schaltflächen 269

Textfelder 268

selected

Auswahlliste 205, 271

options 264

selectedIndex 261

Auswahllisten 271

Selektion von Text 289

self 671, 720

Servlets 23

SetCookie() 733

setDate() 603

setHours() 608

setMinutes() 610

setMonth() 622

setSeconds() 611

Setter-Methode 139

Setter-Methoden 133

setTime() 625

setTimeout() 277, 666, 674, 710, 714

Anhalten 715

setYear() 618

SGML 32

- shift()
 - Datenfelder 506
- size
 - <input> 181
 - <select> 202
 - Formulare 324
- Skriptcontainer
 - Einbindung 79
- Skriptsprachen 23
 - Aufbau 24
- slice() 358
 - Datenfelder 517
 - Strings 591
- small() 552
- Sondertasten 828
- Sonderzeichen
 - Strings 167
- Sonderzeichen in Strings 167
- sort()
 - Datenfelder 490
- sourceIndex
 - all 757
- Spezialisierung
 - Klassen 159
- splice()
 - Datenfelder 512, 526, 538
- split() 358, 466, 505
 - reguläre Ausrücke 560
 - Strings 594
- Spracheinstellung
 - Browser 99
- Sprunganweisung 46, 55
- src
 - images 771
- Startseite
 - aufrufen 706
- static 145
- status 707
 - open() 721
- Statuszeile 707
 - Standardanzeige 708
- Steueranweisungen
 - HTML 27
- Stilgruppen
 - CSS 787
- stop() 712
- strike() 553
- String 39, 63
 - Anzahl der enthaltenen Zeichen 539
 - Aufspalten in eine Datenfeld 594
 - Durchsuchen mit exec() 582
 - Ersetzen von Texten 559
 - HTML-Formatierungen anwenden 550
 - Suche eines Zeichens 556, 558
 - Suchen und Ersetzen mit regulären Ausdrücken 560
 - Teile extrahieren 591
 - Umwandlung in Großbuchstaben 541, 542
 - Unicode an einer bestimmten Stelle 544
 - Verbinden 546
 - Zeichen an einer bestimmten Stelle 543
 - Zeichen aus Unicode-Sequenz erstellen 547
- String() 164, 539
- Stringmanipulation 539
- Stringobjekt 95
- Stunden
 - Extrahieren aus einem Datumsobjekt 607
 - Setzen in einem Datumsobjekt 608
- style 766
 - all 760
 - altes Netscape-Modell 767
 - HTML-Parameter 785
- Style Sheets 654, 783
 - Einbinden in eine Webseite 783
 - externe Dateien 785
 - JavaScript-Anwendung 789
 - Positionieren von Elementen 795
- sub() 553
- Subklasse 160
- submit
 - <input> 185
- submit() 229, 290, 313, 439
- Submit-Schaltfläche
 - Formular 185
- substr() 592
- substring() 593
- sun
 - LiveConnect 687
- Sun Microsystem 23
- sup() 553
- Superklasse 160
- switch 55
- SyntaxError 446
- Systemdatum 597

T

- tagName
 - all 757
- Tags 27
- tags
 - document 767
- tags() 757, 760
- target
 - Formulare 180
 - Hyperlinks 744
 - JavaScript-Zugriff 223
- Tastaturereignisse 816
- TCP/IP 364
- Terminserie 647
- test()
 - RegExp 560, 586
- text 677
- textAlign
 - style 762
- textDecoration
 - style 763
- Textfelder
 - selektieren 268
- textIndent
 - style 763
- Textselektion 817
- textTransform
 - style 763
- this 156, 212, 229, 237, 253, 766
 - Formularelement 243
 - Formularrepräsentation 241
- this.form 212, 253
 - Formularelement 237
- throw 452
- title
 - all 757
 - document 700
- toGMTString() 627
- toLocaleString() 605, 620, 628
- toLowerCase() 358, 542
- toolbar
 - open() 721
- top 671, 720
 - Frames 747
- Toplevel-Domain 365
 - Test auf Gültigkeit 364
- toString() 165
- toUpperCase() 358, 541
- true 39

- try 446
- type 821
 - Formularelement 247
- TypeError 446
- typeof 52, 130, 132
- Typfestlegung 129

U

- Umgebungsvariablen
 - Formulare 177
- undefined 39
 - Arrays 42
 - Datenfelder 467
- Unendlichkeit 125
- unerreichbarer Code 151
- unescape() 166, 733, 739
- Unicode 544
- Unicode-Kodierung
 - Zeichen 546
- Unicode-Sequenz 167
- Uniform Resource Locator 372
- unshift()
 - Datenfelder 524
- Unterprogramm 45
- URIError 446
- URL 371, 372
- url
 - <input> 194
- URL-Codierung 177
- userAgent 585
- UTC() 624

V

- var 40, 142
- Variable 38
 - global 145
 - schleifenlokal 143
 - Test auf Definition 140
 - Typfestlegung 129
- Variablendeklaration 40
- Variablenwert
 - zu einer anderen Webseite weitergeben 736
- VBScript 23
- Vererbung 159
- Vergleichsoperatoren 50
- Verlassen einer Webseite 808
- Verlassen eines Elements 816
- Versandmethode
 - dynamisch umschalten 218

- Formular 216
- Version
 - Unterstützung im Browser 92
 - Unterstützung in verschiedenen Browsern 89
- Versionsangabe
 - JavaScript 84
- Versionszyklen
 - JavaScript 25
- verstecktes Formularfeld 197
- visibility 792, 793
 - style 763
- vlink 678
- vlinkColor
 - document 678
- Vorgabewert
 - einzeiliges Formularfeld 182, 334
- vspace
 - images 771

W

- W3C 22
- Webbrowser
 - Ermitteln der Version über reguläre Ausdrücke 584
 - Zugriff auf die History 730
- Webformular
 - HTML-Grundlagen 170
 - Pflichtfeld 355
 - plausibilisieren 400
 - Test auf gültige E-Mail mit regulären Ausdrücken 586
 - Zugriff aus JavaScript 211
- Webformularcontainer
 - HTML 172
- Webformulare
 - Gestaltung mit Tabellen 171
- Webseite
 - dynamisch schreiben 670
 - Elemente frei positionieren 795
 - Grundgerüst 30
 - Höhe und Breite einer Webseite abfragen 693
 - Inhalt ausdrucken 711
 - Ladevorgang abbrechen 712
 - letzte Änderung abfragen 695
 - neu laden 718
 - Suggestion einer Aktualität 649
 - Überstreichen einer Referenz 814
 - Zugriff auf den Titel 700
 - Zugriff auf den URL 697
 - Zugriff auf Hyperlinks 696
 - Zugriff auf Plug-Ins 699
- Weiterleiten
 - JavaScript 718
- Weiterleitung 91
- Wertebereich
 - festlegen 139
- Wertzuweisung 47, 49
- which 821
- while 56
- whiteSpace
 - style 763
- width 100
 - document 693
 - images 771
 - open() 721
 - style 763
- window 63, 671, 719
 - Unterobjekte 669
- window.alert() 702
- window.clearTimeout() 715
- window.close() 728
- window.closed 729
- window.confirm() 704
- window.defaultStatus 708
- window.document.images 770
- window.frames 746
- window.history 730
- window.home() 706
- window.location 716
- window.moveBy() 724
- window.moveTo() 724
- window.name 699
- window.open() 719, 729
- window.print() 711
- window.prompt() 705
- window.scrollBy() 727
- window.scrollTo() 727
- window.setTimeout() 714
- window.status 707
- window.stop() 712
- Wochentag
 - Ändern in einem Datumsobjekt 607
 - Extrahieren aus einem Datumsobjekt 604
- write() 671
- writeln() 671

X

XHTML 28, 32
 Grundlagen 26
XML 32
XML-Deklaration 34

Z

Zeichenketten
 Kodieren und Dekodieren 165
Zeit
 lokale Darstellung 628

Zeitspanne
 Berechnung 644
Zeitverzögerung 714
Zieladresse
 Formular 221
Zufallszahl 491, 504, 515, 517, 520, 522,
 530, 531, 534, 550
Zugangsbeschränkung 740
Zusatztasten 827
Zuweisungsoperatoren 49



... 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 collage displays several screenshots of the InformIT website interface. One screenshot shows a search bar with 'Suche' and 'GO' buttons, and a sidebar with 'Themenwahl' (Topic Selection) including categories like Betriebssysteme, Computer Hardware, and Softwareentwicklung. Another screenshot shows a 'Buchtipps' (Book Tips) section with a table of recommended books:

Titel	Preis	Status
VB.NET	49,95 EUR	Jetzt downloaden
VB.NET	49,95 EUR	Jetzt downloaden
VB.NET	49,95 EUR	Jetzt downloaden
VB.NET	49,95 EUR	Jetzt downloaden
VB.NET	49,95 EUR	Jetzt downloaden

A third screenshot shows a 'Meine ganz persönliche Bibliothek' (My very personal library) section with a list of books and their download status. A fourth screenshot shows a 'Mitglied werden' (Become a member) section with a login form and a 'Jetzt lernen' (Learn now) button. A fifth screenshot shows a 'Halle und herzlich willkommen bei InformIT' (Hall and a warm welcome to InformIT) section with a search bar and a 'Jetzt lernen' button.

wenn Sie mehr wissen wollen ...

www.InformIT.de



Copyright

Daten, Texte, Design und Grafiken dieses eBooks, sowie die eventuell angebotenen eBook-Zusatzdaten sind urheberrechtlich geschützt.

Dieses eBook stellen wir lediglich als **Einzelplatz-Lizenz** zur Verfügung!

Jede andere Verwendung dieses eBooks oder zugehöriger Materialien und Informationen, einschliesslich der Reproduktion, der Weitergabe, des Weitervertriebs, der Platzierung im Internet, in Intranets, in Extranets anderen Websites, der Veränderung, des Weiterverkaufs und der Veröffentlichung bedarf der schriftlichen Genehmigung des Verlags.

Bei Fragen zu diesem Thema wenden Sie sich bitte an:

<mailto:info@pearson.de>

Zusatzdaten

Möglicherweise liegt dem gedruckten Buch eine CD-ROM mit Zusatzdaten bei. Die Zurverfügungstellung dieser Daten auf der Website ist eine freiwillige Leistung des Verlags. Der Rechtsweg ist ausgeschlossen.

Hinweis

Dieses und andere eBooks können Sie rund um die Uhr und legal auf unserer Website



(<http://www.informit.de>)

herunterladen