

Christian Wenz
Tobias Hauser

Galileo Computing

ActionScript mit JavaScript



Tobias Hauser
Christian Wenz

ActionScript mit JavaScript

Liebe Leserin, lieber Leser,

wir freuen uns, dass Sie sich für ein Buch der Reihe Galileo Computing entschieden haben.

Galileo Computing bietet Titel zu allen wichtigen Sprachen, Tools und Techniken der Programmierung. Die Bücher der Reihe zeigen, wie es wirklich geht und warum – immer mit Blick auf die praktische Anwendung. Galileo Computing ist Fachliteratur für Experten von Experten. Kompakt in der Darstellung und benutzerfreundlich gestaltet. Wo wir Übersetzungen in unser Programm aufnehmen, garantieren wir deren herausragenden Informationswert. Hohe Qualität der Übertragung ist selbstverständlich.

Jedes unserer Bücher will Sie überzeugen. Damit uns das immer wieder neu gelingt, sind wir auf Ihre Rückmeldung angewiesen. Bitte teilen Sie uns Ihre Meinung zu diesem Buch mit. Ihre kritischen und freundlichen Anregungen, Ihre Wünsche und Ideen werden uns weiterhelfen.

Wir freuen uns auf den Dialog mit Ihnen.

Ihre Judith Stevens-Lemoine

Lektorat Galileo Computing

judith.stevens@galileo-press.de

www.galileocomputing.de

Galileo Press

Gartenstraße 24

53229 Bonn

Die Deutsche Bibliothek – CIP-Einheitsaufnahme
Ein Titeldatensatz für diese Publikation
ist bei der Deutschen Bibliothek erhältlich

ISBN 3-89842-263-1

© Galileo Press GmbH, Bonn 2002
1. Auflage 2002

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

Lektorat Judith Stevens-Lemoine, Bonn
Korrektorat Katrin Fischer, Bühl/Baden
Einbandgestaltung Barbara Thoben, Köln
Titelbild Getty Images **Herstellung** Iris Warkus
Satz Typographie & Computer, Krefeld
Druck und Bindung Bercker Graphischer Betrieb, Kevelaer

Das vorliegende Werk ist in all seinen Teilen urheberrechtlich geschützt. Alle Rechte vorbehalten, insbesondere das Recht der Übersetzung, des Vortrags, der Reproduktion, der Vervielfältigung auf fotomechanischem oder anderen Wegen und der Speicherung in elektronischen Medien.

Ungeachtet der Sorgfalt, die auf die Erstellung von Text, Abbildungen und Programmen verwendet wurde, können weder Verlag noch Autor, Herausgeber oder Übersetzer für mögliche Fehler und deren Folgen eine juristische Verantwortung oder irgendeine Haftung übernehmen.

Die in diesem Werk wiedergegebenen Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. können auch ohne besondere Kennzeichnung Marken sein und als solche den gesetzlichen Bestimmungen unterliegen.

Inhalt

Vorwort 13

Die Zielgruppe 13

Das Konzept 14

Hilfe?! 15

Danke! 16

Icons und typografische Konventionen 17

1 Einbau und Grundlegendes 19

1.1 Geschichte von JavaScript 19

1.2 Voraussetzungen 21

1.3 Einbau 23

1.3.1 Verwendung von `<script>` 23

1.3.2 Verwendung von `<script language>` 25

1.3.3 Exkurs: Kommentare 29

1.3.4 Externe Dateien verwenden 30

1.4 JavaScript-Links 31

1.5 Event-Handler 32

2 Spracheinführung 35

2.1 Variablen 35

2.1.1 Namensgebung 35

2.1.2 Numerische Variablen 36

2.1.3 Zeichenketten 36

2.1.4 Boolesche Variablen 37

2.1.5 Variablendeklaration 37

2.2 Operatoren 38

2.2.1 Arithmetische Operatoren 38

2.2.2 Boolesche Operatoren 41

2.3 Kontrollstrukturen: Schleifen 44

2.3.1 for-Schleifen 44

2.3.2 do-while-Schleife 46

2.3.3 while-Schleife 47

2.3.4 for-in-Schleife 48

2.3.5 Schleifensteuerung 48

2.4 Fallunterscheidung 49

2.4.1 if-Anweisung 49

2.4.2 switch-Anweisung 52

2.5 Datenspeicherung 53

2.5.1 Die eval-Methode 54

2.5.2 Arrays 55

2.6 Funktionen 57

2.7 Objekte 60

3 Wichtige Objekte 63

3.1 Das Objekt Date 63

3.1.1 Tagesdatum 64

3.1.2 Y2K 65

3.1.3 Mit Daten rechnen 67

3.1.4 Lokales Datumsformat 69

3.2 Das Objekt Math 69

3.3 Browser-Erkennung 71

3.3.1 Wann ist es ein Netscape Navigator? 73

3.3.2 Wann ist es ein Internet Explorer? 73

3.3.3 Versionsnummer erkennen 74

3.4 Weiterleitung 76

3.5 Event-Handler 77

4 Fenster 79

4.1 Modale Fenster 79

4.1.1 Warnfenster 79

4.1.2 Bestätigungsfenster 80

4.1.3 Fenster mit Rückgabewert 81

4.2 Fenster öffnen 82

4.2.1 Fenster öffnen und schreiben 82

4.2.2 Fenster öffnen und verlinken 83

4.2.3 Fenster öffnen und anpassen 84

4.3 Zeitversetzte Codeausführung 89

4.4 Auf Frames zugreifen 90

4.4.1 Zugriff 90

4.4.2 Immer Frames 91

4.4.3 Mehrere Frames gleichzeitig ändern 93

5 Grafiken 95

5.1 Image-Objekt einbinden 96

5.1.1 Grafiken austauschen 97

5.1.2 Vorladen 98

5.1.3 Rollover 99

5.1.4 Komplettes Beispiel 101

5.2	Vorladen per Funktion	102
5.3	Animierte JPEGs	105
5.3.1	Eine Animation mit JavaScript	106
5.3.2	Vorladen	107
5.4	Fazit	108
6	Formulare	109
6.1	Überprüfung auf Vollständigkeit	109
6.1.1	Allgemeiner Aufbau	112
6.1.2	Texteingabefelder	112
6.1.3	Radiobuttons	113
6.1.4	Checkboxen	114
6.1.5	Auswahllisten	114
6.1.6	Fehlermeldung ausgeben	118
6.1.7	Konstruktive Vorschläge	120
6.2	Automatische Überprüfung	124
6.2.1	Texteingabefelder	125
6.2.2	Radiobuttons	126
6.2.3	Checkboxen	127
6.2.4	Auswahllisten	127
6.2.5	Zusammenfassung	127
7	DHTML	131
7.1	Browserabhängiges DHTML	133
7.1.1	Der Ansatz von Netscape	133
7.1.2	Der Ansatz von Microsoft	135
7.1.3	Der Ansatz des W3C	136
7.2	Browserunabhängiges DHTML	137
7.3	Beispiel: Fliegendes Firmenlogo	141
7.4	Beispiel: Permanentes Werbebanner	148
7.5	Fazit	154
8	Basics	155
8.1	Einsatzgebiete von ActionScript	155
8.2	Flash-Basics	156
8.2.1	Windows und Mac	157
8.2.2	Koordinatensystem	159
8.2.3	Ereignisse	161
8.2.4	Symbole und Instanzen	165
8.2.5	Szenen	168
8.2.6	Textfelder	168

8.2.7	Film testen	169
8.2.8	Komponenten	170
8.3	ActionScript in Flash	171
8.3.1	Aktionen-Bedienfeld	172
8.3.2	Externer Code	183
8.3.3	Voreinstellungen	184
8.3.4	Ansichten	187
8.3.5	Referenz	187
8.3.6	Film-Explorer	188

9 ActionScript: Grundlagen 191

9.1	Variablen und Datentypen	191
9.1.1	Namenskonventionen	192
9.1.2	Gültigkeitsbereich	194
9.1.3	Datentypen	195
9.1.4	Automatische Typkonvertierung	201
9.1.5	Explizite Typkonvertierung	202
9.2	Operatoren	204
9.2.1	Arithmetische Operatoren	204
9.2.2	Vergleichsoperatoren	206
9.2.3	Logische Operatoren	209
9.3	Kontrollstrukturen	210
9.3.1	Fallunterscheidungen	211
9.3.2	Schleifen	219
9.4	Beispielanwendung – Taschenrechner	224
9.4.1	Vorbereitungen – die Flash-Datei	224
9.4.2	Variablendeklaration	229
9.4.3	Die Zifferntasten	229
9.4.4	Operatoren	230
9.4.5	Clear	231
9.4.6	Ausrechnen	231

10 Fortgeschrittene Programmierung 237

10.1	Funktionen	237
10.1.1	Eigene Funktionen	237
10.1.2	Parameter	242
10.1.3	Funktionen im Normalmodus	246
10.1.4	Code strukturieren	247
10.1.5	Vorgefertigte Funktionen	252
10.2	Arrays	275
10.2.1	Arrays erstellen	276
10.2.2	Datentypen in Arrays	276
10.2.3	Arrays auslesen	276

- 10.2.4 Bestehende Arrays ändern 279
- 10.2.5 In Arrays suche n280
- 10.2.6 Assoziative Arrays 281
- 10.2.7 Ausblick 282

11 Objekte, Eigenschaften und Methoden 283

- 11.1 Begriffe 283**
 - 11.1.1 Eigenschaften und Methoden 284
 - 11.1.2 Vererbung 285
 - 11.1.3 Polymorphismus 286
- 11.2 Eigene Objekte, Eigenschaften und Methoden 286**
 - 11.2.1 Klassen 287
 - 11.2.2 Der Konstruktor new 288
 - 11.2.3 Eigenschaften 288
 - 11.2.4 Methoden 290
 - 11.2.5 Alle Eigenschaften auslesen 293
 - 11.2.6 Vererbung 294
 - 11.2.7 Mehrfachvererbung mit Parametern 297
 - 11.2.8 Polymorphie 299
- 11.3 Vorgefertigte Objekte 300**
 - 11.3.1 Eigenschaften 300
 - 11.3.2 Allgemeines zu Objekten 305
 - 11.3.3 String 306
 - 11.3.4 Array 316
 - 11.3.5 Andere Datentypen 319
 - 11.3.6 Datum 320
 - 11.3.7 Mathematische Methoden und Konstanten 329

12 Movieclips 335

- 12.1 Movieclips 335**
 - 12.1.1 Ein Beispiel 338
 - 12.1.2 Zugriff 343
 - 12.1.3 Mit ActionScript erstellen 350
 - 12.1.4 Löschen 364
 - 12.1.5 with-Anweisung 366
 - 12.1.6 Ereignisse für Movieclips 367
 - 12.1.7 Ereignisprozeduren 369
 - 12.1.8 Eigenschaften 371
 - 12.1.9 Methoden 373
- 12.2 Externe Filme 378**
 - 12.2.1 Ein Beispiel 380
 - 12.2.2 Film laden 383
 - 12.2.3 Film löschen 386

13	Textfelder 389
13.1	Textfeld mit ActionScript erstellen 389
13.2	Die Eigenschaften eines Textfelds 390
13.3	Die Methoden und Ereignisse des Textfelds 394
13.4	Textfeld formatieren 395
13.5	Ein Beispiel 398
14	Sound und Video 401
14.1	Sound in Flash 401
14.1.1	Globale Steuerung für Sounds 403
14.1.2	Das Sound-Objekt 403
14.1.3	Fähigkeiten des Clients testen 409
14.2	Video 410
15	Debugging 415
15.1	Testen 415
15.2	trace etc 417
15.3	Debugger 419
15.4	Bandbreite 422
16	Preloader 425
16.1	Teile vorladen 426
16.2	Gesamten Film vorladen 432
17	ActionScript mit JavaScript 437
17.1	Von ActionScript nach JavaScript 438
17.1.1	Die Funktion fscommand() 438
17.1.2	Die Funktion getURL() 460
17.1.3	Anwendung: PHP-Session-Management 463
17.2	Von JavaScript nach ActionScript 469
17.2.1	Zugriff 470
17.2.2	Funktionen und Methoden 472
17.2.3	Beispiel: Preloader 476
17.2.4	Beispiel: Filmsteuerung 482
17.2.5	Beispiel: Flash-Variablen setzen 490
17.3	Fazit 495

18	XML	497
18.1	Was ist XML?	498
18.2	Das XML-Objekt	502
18.3	Häufige Anwendungen	510
18.3.1	Daten laden	510
18.3.2	Elemente finden	513
18.3.3	XML formatiert ausgeben	516
18.4	Beispiel: RSS-Newsbanner	524
19	Spiele programmieren	533
19.1	Grundlagen	533
19.1.1	Tastatur abfragen	533
19.1.2	Sounds abspielen	535
19.1.3	Kollisionstest	536
19.2	Der Meteoritenhagel	537
19.2.1	Startbildschirm	537
19.2.2	Die einzelnen Movieclip	538
19.2.3	Kollisionstest	540
19.2.4	Automatisches Hageln	541
19.2.5	Zählen	542
19.2.6	Endstand	544
19.2.7	Musikalische Untermalung	544
19.2.8	Veröffentlichen und spielen	545
19.3	Erweiterungen	548
20	Komponenten von Flash MX	551
20.1	Übersicht	551
20.2	Ansteuerung	553
20.2.1	CheckBox	553
20.2.2	ComboBox	554
20.2.3	ListBox	555
20.2.4	PushButton	557
20.2.5	RadioButton	557
20.2.6	ScrollBar	559
20.2.7	ScrollPane	560
20.3	Beispiel	561
20.3.1	Formular erstellen	561
20.3.2	Formular überprüfen	562
20.3.3	Formular verarbeiten	564
20.4	Fazit	567

21 Serverseitige Technologien 569

21.1 Voraussetzungen 569

21.2 Serverseitige Kommunikation 572

21.2.1 Die Funktion `getURL()` 572

21.2.2 Die Funktion `loadVariables()` 578

21.3 Anwendungen 584

21.3.1 Mailversand 584

21.3.2 Dynamischer News-Ticker 591

Anhang

A Unterschiede zwischen ActionScript und ECMAScript 603

B Wichtige Tastenkürzel 605

Index 607

Vorwort

*Spruchwörter sind ein öffentlicher Unterstützungsverein für
Leute ohne eigene Gedanken.*

– Wilhelm Raabe

Das Lieblingswort eines jeden Macromedia-Vertriebsmitarbeiters in diesen Tagen ist »Skip Intro«. Es vergeht keine Produktvorführung, keine Pressekonferenz, in der nicht dieses Schlagwort verwendet wird.

Was aber ist »Skip Intro«? Nun, die Macromedia-Vertriebsmannschaft hat ganz richtig erkannt, dass Flash heutzutage im World Wide Web hauptsächlich dazu eingesetzt wird, Animationen zu erstellen, die vor die eigentliche Homepage gestellt werden. Erst nach Ablauf der Animation offenbaren sich die eigentlichen Informationen. Viele Designer sind zumindest so gnädig, eine direkte Sprungmöglichkeit zur eigentlichen Hauptseite des Webangebots anzubieten. Diese wird landein, landaus mit »Skip Intro« betitelt – englisch für »Einleitung überspringen«. Die Qualität und/oder der inhaltliche Gehalt vieler dieser Flash-Filme führt dazu, dass die Einladung »Skip Intro« nur allzu gerne angenommen wird.

Die Zielgruppe

Mit Flash MX – eigentlich Flash 6, auch die interne Versionsnummer ist 6 – soll das Produkt neu positioniert werden. Programmierer werden augenscheinlich verstärkt angesprochen. Flash bietet in der neuesten Version in der Tat zusätzliche Funktionalität, mit der die Entwicklungsumgebung zu weit mehr fähig ist als zum Erstellen animierter Grafiken. Allerdings müssen Sie dazu ein wenig programmieren – und hier setzt dieses Buch an.

Programmierung ist in Flash schon seit langem möglich, mit der Flash-Version 5 jedoch hat dieser Aspekt noch mehr an Bedeutung gewonnen. Auf den ersten Blick ist das erschreckend – ein guter Designer ist in den meisten Fällen ein lausiger Programmierer, ein guter Programmierer ein ganz fürchterlicher Designer, so zumindest lautet das Klischee. Außerdem muss der klassische Designer in seinem Arbeitsalltag meist überhaupt nicht programmieren.

Diese Barriere ist jedoch keinesfalls unüberwindbar. Die Flash-eigene Programmiersprache ActionScript ist mächtig, aber trotzdem leicht zu

erlernen. Wichtig ist hierbei jedoch, dass Sie die Programmierung von ActionScript kennen lernen und verinnerlichen. Dadurch schaffen Sie eine solide Grundlage und können auch ohne Programmiererfahrung sogar komplexere Programme erstellen. Dafür wollen wir mit diesem Buch sorgen.

Das Konzept

Die Syntax von ActionScript lehnt sich an ECMAScript (genauer ECMA-262) an, einem im World Wide Web anerkannten Standard (siehe auch <http://www.ecma.ch/>). Anerkannt, weil es nämlich noch eine andere Sprache gibt, die sich an ECMAScript orientiert – die Rede ist von der clientseitigen Skriptsprache JavaScript, mit der HTML-Seiten neues Leben eingehaucht werden kann.

Zumindest von der Syntax – den Regeln der Sprache – her sind sich ActionScript und JavaScript sehr ähnlich. Wer für das World Wide Web produziert, benötigt beide Sprachen. Flash-Filme ohne ActionScript oder HTML-Seiten ohne JavaScript sind einfach nicht so funktional und mächtig wie solche mit Skriptcode.

Syntaktische Gemeinsamkeiten, gleiche Kontrollstrukturen und verwandte Funktionen, das sind nur drei der Dinge, die ActionScript und JavaScript einander so ähnlich machen. Oder, anders gesagt, beherrschen Sie die eine Sprache, lernen Sie die andere im Handumdrehen. Deswegen verfolgen wir in diesem Buch einen neuen Ansatz. Wir beginnen mit einer kurzen und prägnanten Einführung in JavaScript, um uns dann dem Hauptthema des Buchs – ActionScript – zu widmen.

Dazu haben wir das Buch in drei Teile aufgeteilt.

- ▶ Im ersten Teil erhalten Sie eine Einführung in JavaScript. Es handelt sich hierbei um einen speziell angepassten und erweiterten Extrakt aus den Titeln »JavaScript – Browserübergreifende Lösungen« und »JavaScript-Rezepte«, beide ebenfalls bei Galileo Press erschienen. Trotz der Kürze der Darstellung werden die wesentlichen Themen behandelt, inklusive Formularen und DHTML.
- ▶ Der zweite Teil führt Sie in ActionScript ein. Die grundlegenden Prinzipien von ECMAScript sind Ihnen bereits bekannt, was uns mehr Raum lässt für eine ausführlichere Beschreibung der Möglichkeiten der Sprache. Wir gehen dabei auch auf viele Flash-Spezifika wie beispielsweise Filmsteuerung und die dynamische Einbindung von Audio- und Video-daten ein.

- Im dritten Teil bringen wir eine Reihe von Praxisbeispielen. Nachdem wir bereits im ersten und im zweiten Teil etliche Beispiele gezeigt haben, wagen wir uns hier an komplette Anwendungen, die teilweise auch recht komplex werden. Außerdem werfen wir einen Blick über den Tellerrand und zeigen, wie Flash zusammen mit anderen Technologien (etwa der serverseitigen Programmierung mit PHP und ASP) eingesetzt werden kann oder wie mit Flash Spiele programmiert werden können. Im zentralen Kapitel dieses Buchteils schlagen wir eine Brücke zwischen den ersten beiden Teilen und zeigen, wie ein Flash-Film und eine HTML-Seite (mit eingebettetem JavaScript-Code) miteinander kommunizieren können.
- Im Anhang tragen wir die wichtigsten Unterschiede zwischen ActionScript und JavaScript zusammen, um Ihnen dabei zu helfen, Fehler und Verwechslungen zu vermeiden.

Hilfe?!

An dieser Stelle sind wir auch auf Ihre Hilfe angewiesen. Wie gefällt Ihnen das Konzept, wie gefällt Ihnen das Buch allgemein? Auf der Homepage des Verlags, <http://www.galileocomputing.de/>, können Sie Ihre Meinung loswerden, der Verlag leitet Ihre Zuschriften dann an uns weiter. Wenn Sie sich (kostenlos) registrieren, haben Sie in der Rubrik **MyGalileo** Zugriff auf spezielle Inhalte, beispielsweise Updates zum Buch oder zusätzliche Tipps und Artikel. Außerdem können Sie im Forum Fragen zum Buch und zum Thema im Allgemeinen stellen und so mit anderen Lesern und auch den Autoren in Kontakt treten.

Wir freuen uns natürlich über Lob, sind aber auch sehr an (konstruktiver) Kritik interessiert. Das nämlich ermöglicht uns, die späteren Auflagen dieses Titels noch besser zu machen. Wenden Sie sich über den Verlag an uns, oder besuchen sie <http://www.hauser-wenz.de/support/>. Bevor Sie uns schreiben, bitten wir Sie noch um Folgendes: Wenn Sie einen Fehler gefunden haben, überprüfen Sie bitte zunächst unter den obigen Webadressen, ob dieser Fehler unter Umständen bereits bekannt ist und eine Korrektur bereitsteht. Bei E-Mails gelten die üblichen Netiquette-Regeln: keine Priorität »hoch«, keine HTML-Formatierungen. Dafür erhalten Sie erfahrungsgemäß innerhalb von 48 Stunden eine Antwort, oftmals auch schneller. Lediglich bei akuter Arbeitsüberlastung oder Urlaub (soll ja mal vorkommen) kann es auch mal länger dauern. Ihre eigenen Programme und Flash-Filme (also Code, der nicht aus dem Buch stammt) können wir

nur in Ausnahmefällen untersuchen und nur, wenn die Frage samt Code im Forum gepostet wurde.

Also, machen Sie von dieser Möglichkeit Gebrauch und geben Sie uns Feedback!

Danke!

Als Autor darf man zwar seinen Namen an prominenter Stelle auf dem Cover abdrucken lassen, bekommt aber nur einen Bruchteil der Einnahmen. Das hat allerdings auch (innerhalb gewisser Grenzen :-)) seinen Sinn, da hinter der Buchproduktion und dem Vertrieb eine ganze Mannschaft steht, die zum Gelingen dieses Titels beigetragen hat und noch beiträgt. Besonders erwähnen möchten wir hier unsere Lektorin Judith Stevens-Lemoine, die das gesamte Projekt koordiniert hat und Termine auf magische Art und Weise so verschieben konnte, dass am Ende alles gepasst hat. Katrin Fischer hat das Korrekturat durchgeführt; sollten Sie noch auf einen Fehler stoßen, haben wir den sicherlich nachträglich eingebaut. Florian Wenz hat die Audiodaten für Kapitel 19, Spiele programmieren, komponiert und eingespielt.

Zu guter Letzt noch der obligatorische Dank an unsere Familien und Freunde, die (»wieder einmal«) ein Buchprojekt ohne vernehmbares Murren begleitet und damit entscheidend unterstützt haben.

Und nun »Skip Intro« und viel Vergnügen bei der Lektüre!

Christian Wenz und **Tobias Hauser**, im Juni 2002

Icons und typografische Konventionen

An vielen Stellen in diesem Buch setzen wir Icons ein, um Sie auf besondere Stellen im Text hinzuweisen.

Hinweise enthalten meist zusätzliche Informationen zum Thema oder Verweise auf die CD-ROM zum Buch, wie etwa die Tatsache, dass Sie alle Listings auf der CD finden. Für jedes Kapitel gibt es im Ordner **beispiele** ein Unterverzeichnis.



Auf mögliche Stolperfallen bei der Arbeit mit ActionScript, JavaScript oder Flash weisen wir Sie mit diesem Icon hin. Beispielsweise sind die Daten von der CD-ROM, wenn Sie sie auf Ihre lokale Festplatte kopieren, schreibgeschützt und können so nicht verändert werden.



Tipps, Tricks, Kniffe und alles andere, was Ihnen die tägliche Arbeit mit den in diesem Buch vorgestellten Programmiersprachen und Tools erleichtern kann, wird mit diesem Icon gekennzeichnet. Beispielsweise können Sie unter Windows den Schreibschutz für alle Dateien in einem Verzeichnis samt allen Unterverzeichnissen mit folgendem DOS-Befehl aufheben:



```
ATTRIB -R *.* /S /D
```

Außerdem setzen wir die folgenden Formatierungen für spezielle Textelemente und Absätze ein:

- Besondere Stellen im Text werden **fett** hervorgehoben. Dazu zählen auch Dateinamen, Webadressen (URLs), Menübefehle und Namen von Schaltflächen.
- Wenn innerhalb des Fließtextes Programmcode vorkommt, wird dieser in einer nichtproportionalen Schrift ausgezeichnet.

1 Einbau und Grundlegendes

Aller Anfang ist schwer. Höchstens das Aufhören ist manchmal noch schwerer.

– Victor Moritz Goldschmidt

Von immer wieder auftretenden Sicherheitslücken in der Programmierung aller Browser einmal abgesehen: Mit JavaScript können Sie die eher beschränkten Möglichkeiten von HTML erweitern. Es handelt sich hierbei um eine clientseitige Programmiersprache, das heißt alles läuft im Browser ab und es müssen keine besonderen Server-Voraussetzungen erfüllt sein. Dieser Punkt ist (noch) ein großer Nachteil von serverseitigen Sprachen wie ASP, ASP.NET, Perl, PHP und Konsorten, denn viele Hosters (und vor allem die günstigen Hosters) gestatten keine serverseitige Programmierung. Mit JavaScript ist das alles kein Problem.

1.1 Geschichte von JavaScript

Gegen Ende des Jahres 1995 stellte die Firma Netscape die neueste Version ihres Internet-Browsers vor. Der Versionssprung von 1.1 auf 2.0 war berechtigt, da so revolutionäre Dinge wie etwa die Unterstützung von Frames als neues Feature dazugekommen waren. Unter anderem war auch eine Sprache namens **LiveScript** eingebaut worden, mit der man auf HTML-Seiten Einfluss nehmen konnte. Die Syntax dieser Sprache lehnte sich an Java an, und wohl aus marketingtechnischen Gründen wurde die Programmiersprache schließlich in **JavaScript** umbenannt. Die Ähnlichkeit im Namen war gewollt und beabsichtigt; die Sprachen selbst haben aber rein gar nichts miteinander zu tun (was eine Reihe von Buchhändlern nicht davon abhält, Bücher zu beiden Themen in dasselbe Regal zu räumen).

Schon bald begann der Siegeszug dieser Programmiersprache. Zwar war die Implementierung im Browser eher mangelhaft (in der Netscape-Version 2.0 konnte man JavaScript nicht einmal deaktivieren, was zu einem bösen Sicherheitsproblem werden sollte), aber mit der Beta-Version des Netscape Navigator 3 wurde die JavaScript-Version 1.1 vorgestellt, die deutlich mehr Möglichkeiten bot. Microsoft wollte nun auch auf den Zug aufspringen und kündigte an, in Version 3 des Internet Explorers auch Skriptunterstützung anzubieten. Aus lizenzrechtlichen Gründen wurde

die Sprache **JScript** getauft, von der Syntax her war sie aber praktisch identisch mit JavaScript.

ECMAScript Seitdem läuft das Wettrennen zwischen Netscape und Microsoft. Während der Internet Explorer 3 praktisch nur JavaScript 1.0 unterstützte, beherrscht der Internet Explorer 4 schon JavaScript 1.1 – und einige Features aus dem Sprachschatz von JavaScript 1.2, das mit dem parallel zum »IE4« erscheinenden Netscape Navigator 4 eingeführt wurde. Seit dem Netscape Navigator 4.06 gibt es die JavaScript-Version 1.3, die zwar nur unbedeutende Verbesserungen bietet, sich aber an dem ECMA-262-Standard, ECMAScript (<http://www.ecma.ch/>), orientiert. Netscape hatte erkannt, dass man auf Standards setzen muss, und gleichzeitig behauptet, dass der Netscape Navigator 4.06 der Browser sei, der der Spezifikation von ECMAScript am nächsten komme. Microsoft wiederum hat mit dem Internet Explorer 5 den Rückstand weiter verkleinert, setzt aber inzwischen eher auf andere Standards wie **DOM (Document Object Model)**. Unter dem Codenamen Mozilla wurde mittlerweile die nächste Netscape-Version als Open Source entwickelt, das heißt der Sourcecode lag und liegt offen und jeder konnte an der neuen Version mitentwickeln. Nichtsdestotrotz wurde der Löwenanteil der Arbeit von Netscape-Angestellten geleistet – das »Linux-Wunder« funktioniert eben nicht überall.

Im Spätherbst 2000 begannen sich die Ereignisse zu überschlagen. Unter dem Druck des Internet Explorer 5.5 sowie des als Teil des neuen Microsoft-Betriebssystems XP angedrohten Internet Explorer 6 wurde flugs der aktuelle Zwischenstand des Open-Source-Projekts genommen, eine Installationsroutine dazugebastelt und das Ganze als Netscape 6 zum Download angeboten (bezeichnenderweise wurde zeitgleich eine fast identische Version auf der Mozilla-Website als Version 0.6 zum Download angeboten). Dieser Schritt stieß in der Fachwelt auf einige Kritik, denn die Version war noch weit davon entfernt, im Produktiveinsatz verwendet werden zu können. Und, aus Sicht der Webdesigner viel schlimmer, die neue Version erwies sich im Hinblick auf JavaScript als nicht abwärtskompatibel, wenn es um »DHTML« ging. So mussten (und müssen immer noch) eine Reihe von Skripten umprogrammiert werden. Die neue Version unterstützt übrigens JavaScript 1.5, das aber im Vergleich zu den Vorgängerversionen nur unwesentliche Änderungen mit sich bringt (es wird eine aktuellere Version von ECMAScript unterstützt).

Die Crux bei der Programmierung von Webseiten besteht darin, dass die beiden großen Browserhersteller immer wieder versucht haben, sich dadurch zu übertrumpfen, dass sie immer neue Features in ihre Browser

einbauen. Das ist innerhalb des eigenen Mikrokosmos' eine gute Sache, aber wenn die Technik inkompatibel zur Konkurrenz ist, schaut etwa die Hälfte der Websurfer (nämlich diejenigen, die den jeweils anderen Browser benutzen) bildlich gesprochen in die Röhre. Keine Firma kann es sich leisten, ein Web-Angebot nur für die Hälfte der Zielgruppe zu erstellen.

Es ist ja nicht so, dass nur der Netscape Navigator und der Internet Explorer JavaScript unterstützen. Die Nummer 3 im Browsermarkt, der Opera-Browser, unterstützt die JavaScript-Version 1.3, und das nicht einmal schlecht. Auch der in StarOffice integrierte Webbrowser sowie die Version 3 von Suns HotJava-Browser sind JavaScript-fähig, wenngleich nicht messbar verbreitet. Mit umsichtiger Programmierung lassen sich aber auch hier Fehler vermeiden.

Sonstige Browser

1.2 Voraussetzungen

Wenn Sie diesen Teil durcharbeiten wollen, sollten Sie zumindest Kenntnisse in HTML vorzuweisen haben – wir wollten dieses Buch nicht durch einen zusätzlichen HTML-Teil künstlich aufblähen. Ebenfalls sollten Sie sicher mit Ihrem Betriebssystem umgehen können und beispielsweise Editoren starten und Dateien speichern können. Apropos Editoren: Im Gegensatz zu HTML gibt es bei der JavaScript-Programmierung keinen großen Markt für Editoren, weil der JavaScript-Code größtenteils manuell erstellt wird. Die einfachen Texteditoren, die mit dem Betriebssystem mitgeliefert werden (beispielsweise **Notepad** oder **SimpleEdit**), reichen sogar schon aus. Wer mehr will, sollte einen der folgenden Editoren ausprobieren (sie sind aber Shareware und damit nach Ablauf einer Testperiode kostenpflichtig); die Reihenfolge entspricht meinen persönlichen Präferenzen:

HTML-Kenntnisse vorausgesetzt

Editoren

- ▶ Ultraedit (<http://www.ultraedit.com/>) – Sehr leistungsfähiger Editor inklusive Syntax-Highlighting (Windows)
- ▶ Programmer's File Editor (<http://www.lancs.ac.uk/people/cpaap/pfe/>) – Die Entwicklung wurde mittlerweile eingestellt, aber unter obiger Adresse ist dieser Editor immer noch erhältlich (Windows).
- ▶ NoteTab (<http://www.notetab.ch/>) – Mehrfach ausgezeichnete Klassiker unter den Texteditoren, auch in kostenfreier Light-Variante erhältlich (Windows). Fürs häufige Entwickeln sollte es aber eine der bezahlbaren kostenpflichtigen Versionen mit mehr Funktionalität sein.
- ▶ BBEEdit (<http://www.bbedit.com/>) – Der Macintosh-Klassiker, auch in einer Light-Version erhältlich.

Kompatibilität Außerdem sollten Sie möglichst viele Browser installieren. Sie sollten nämlich Ihre JavaScript-Programme auf möglichst vielen Zielplattformen testen, um sicherzustellen, dass die Programme eben nicht nur bei Ihnen laufen. Wenn Sie den Internet Explorer 5 oder 5.5 über den Internet Explorer 4 installieren, geben Sie bei der Installation unbedingt an, dass die alte Version (im so genannten **Kompatibilitätsmodus**) erhalten bleibt. Der Internet Explorer 3 lässt sich in der 32-Bit-Variante nicht parallel installieren, aber wenn Sie ein wenig abenteuerlustig sind, verwenden Sie die 16-Bit-Variante, das funktioniert (angeblich; alle Installationen gehen auf Ihr eigenes Risiko). Beim Netscape Navigator ist das nicht so schwierig, die Versionen 6, 4.x und 3 lassen sich problemlos parallel installieren. Den Opera-Browser sollten Sie ebenfalls installieren (<http://www.opera.com/>), da er auch eine gute JavaScript-Unterstützung hat.



Bei aller Liebe zur Kompatibilität mit antiken Browsern – den Netscape Navigator 3 und auch den Internet Explorer 4 können Sie mittlerweile vernachlässigen.

Netscape 6 Den aktuellen Entwicklungsstand der nächsten Netscape-Version können Sie unter <http://www.mozilla.org/> begutachten. Das offizielle Release unter <http://www.netscape.com/de/> basiert auf der Version unter Mozilla, hat aber noch einige Bugs mehr. Beispielsweise basiert die zurzeit (Mai 2002) aktuellste Version 6.2.3 auf Mozilla 0.9.4, während zum selben Zeitpunkt bereits Mozilla 1.0 RC 2 (RC steht für »Release Candidate«) verfügbar ist.

Während der Entwicklung Ihrer Programme sollten Sie unbedingt primär den Netscape Navigator 4.x verwenden. Das hat nichts mit einer Glaubbenseinstellung oder Fanatismus zu tun, es ist nur so, dass die Fehlermeldungen des Netscape Navigator viel aussagekräftiger sind als die des Internet Explorers. Läuft ein Programm einmal im Netscape einigermaßen zuverlässig, dann sollten Sie natürlich sofort dazu übergehen, es im Internet Explorer zu testen.



Seit der Version 4.06 führt ein JavaScript-Fehler beim Netscape Navigator nicht mehr zu einem eigenen Warnfenster; stattdessen werden alle Fehlermeldungen in der JavaScript-Konsole angezeigt. Diese wird sichtbar, wenn Sie in der Adresszeile des Browsers `javascript:` eintippen (inklusive des Doppelpunkts) und auf  drücken. Im Netscape 6 funktioniert das mit dem Menübefehl **Task • Tools • JavaScript Console** bzw. **Aufgaben • Extras • JavaScript-Konsole**.

1.3 Einbau

JavaScript-Kommandos können an mehreren Stellen einer HTML-Datei untergebracht werden:

- ▶ zwischen den Tags `<script>` und `</script>`,
- ▶ in einer externen Datei,
- ▶ in Form eines HTML-Links,
- ▶ als Parameter von HTML-Tags.

In den folgenden Abschnitten werden die einzelnen Möglichkeiten der Reihe nach vorgestellt und beleuchtet.

Als Beispiel hierfür dient die Anweisung `document.write("ActionScript mit JavaScript")`, welche den Text "ActionScript" ausgibt. Warum dieser Befehl genau so funktioniert, erfahren Sie demnächst; fürs Erste müssen Sie uns einfach vertrauen.

1.3.1 Verwendung von `<script>`

Die naheliegendste Methode, JavaScript-Befehle auszuführen, besteht darin, das `<script>`-Tag zu verwenden. Folgender Code sorgt dafür, dass »ActionScript mit JavaScript« ausgegeben wird:

```
<script>
document.write("ActionScript mit JavaScript");
</script>
```

Befehle werden in JavaScript untereinander – einer pro Zeile – dargestellt. Außerdem sollten Sie jede Anweisung mit einem Semikolon abschließen. Browser akzeptieren zwar auch Anweisungen ohne Semikolon, die von uns empfohlene Vorgehensweise ist jedoch sauberer und weniger fehlerträchtig.

Wenn Sie mehrere Kommandos in einer Zeile unterbringen wollen, müssen Sie die Anweisungen unbedingt durch ein Semikolon voneinander trennen.

In der ersten Version der JavaScript-Sprachspezifikation war das Semikolon am Ende jeder Anweisung strikt vorgeschrieben. Inzwischen wurde aber davon Abstand genommen, und jeder JavaScript-Programmierer hat seinen eigenen Stil.



Die beiden folgenden Anweisungen sind also äquivalent. Einmal stehen die beiden Befehle in verschiedenen Zeilen, einmal in einer Zeile.

```
<script>
document.write("ActionScript ");
document.write("mit JavaScript");
</script>
```

und

```
<script>
document.write("ActionScript "); document.write("mit
JavaScript");
</script>
```

JavaScript-Code wird hierbei vom JavaScript-Interpreter des verwendeten Browsers ausgeführt. Betrachten Sie zum Beispiel folgendes HTML-Dokument:

```
<html>
<head>
<title>JavaScript</title>
</head>
<body>
<script>
document.write("ActionScript mit JavaScript");
</script>
</body>
</html>
```

Listing 11 Das erste funktionsfähige Beispiel

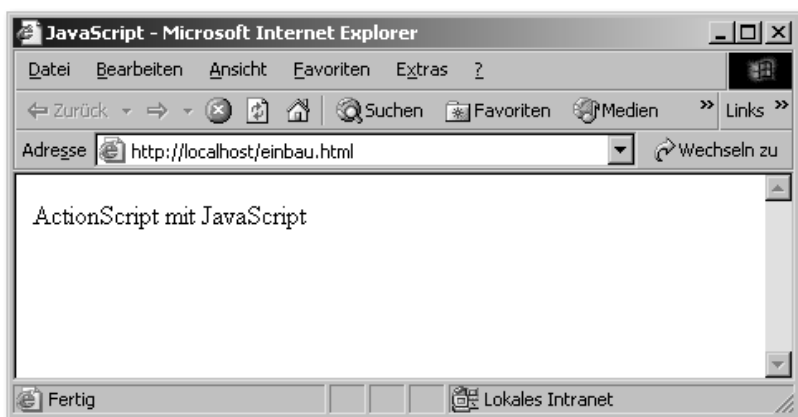


Abbildung 11 Der Text wird ausgegeben.

Wenn dieses Listing (auf der CD-ROM: **einbau.html**) vom Browser interpretiert worden ist und dieser Browser JavaScript unterstützt, verhält es sich so wie folgendes HTML-Dokument:

```
<html>
<head>
<title>JavaScript</title>
</head>
<body>
ActionScript mit JavaScript
</body>
</html>
```

Nehmen Sie es uns nicht übel, wenn die ersten Beispiele in diesem Kapitel nicht unbedingt die breite Funktionspalette von JavaScript demonstrieren.

1.3.2 Verwendung von `<script language>`

Das obige Beispiel ist streng genommen etwas unsauber. Der `<script>`-Tag eignet sich auch für andere Programmiersprachen, die in HTML-Dokumente eingebettet werden können, beispielsweise für Visual Basic Script (VBScript) oder JScript. Dazu dient das Attribut `language` des `<script>`-Tags. Ist es nicht gesetzt – wie im obigen Beispiel – so wird angenommen, dass die zwischen den Tags stehenden Kommandos in JavaScript verfasst wurden (deswegen funktioniert das obige Beispiel auch). Aber um auf Nummer Sicher zu gehen – es könnte ja sein, dass eine neue Version des Microsoft Internet Explorers als Standardsprache VBScript annimmt –, schreiben wir:

```
<html>
<head>
<title>JavaScript</title>
</head>
<body>
<script language="JavaScript"><!--
document.write("ActionScript mit JavaScript");
//--></script>
</body>
</html>
```

Listing 1.2 Die verwendete Skriptsprache wird angegeben.



Wie Sie sehen, folgt direkt auf `<script>` der Beginn eines HTML-Kommentars, `<!--;` direkt vor `</script>` steht `//-->`. Der Grund hierfür ist ein historischer; Browser, die überhaupt kein JavaScript beherrschen, sollen damit an der ungefilterten Ausgabe des JavaScript-Codes gehindert werden. Mittlerweile ist das zwar unnötig, aus reiner Sentimentalität behalten wir jedoch diesen Zusatz bei.

Wie Sie zuvor erfahren haben, gibt es mehrere Versionen von JavaScript. Sie können im `language`-Attribut auch explizit eine der Versionen angeben. Der folgende Code wird nur von Browsern ausgeführt, die die JavaScript-Version 1.5 unterstützen (das ist eigentlich nur der Netscape 6):

```
<html>
<head>
<title>JavaScript</title>
</head>
<body>
<script language="JavaScript1.5"><!--
document.write("ActionScript mit JavaScript");
//--></script>
</body>
</html>
```

Listing 1.3 Dieses Listing funktioniert nur mit JavaScript 1.5.

»Ältere« Browser, beispielsweise der Internet Explorer 6, ignorieren den JavaScript-Befehl und geben nichts aus.

Zur Zeit der Drucklegung (Juni 2002) waren die in Tabelle 2.1 aufgeführten Parameter gültig.

Parameter	Bedeutung
JavaScript	Jeder Browser, der JavaScript unterstützt
JavaScript1.1	Alle Browser, die mindestens die JavaScript-Version 1.1 unterstützen (ab NN3, IE4)
JavaScript1.2	Alle Browser, die mindestens die JavaScript-Version 1.2 unterstützen (ab NN4, IE5)
JavaScript1.3	Alle Browser, die mindestens die JavaScript-Version 1.3 unterstützen (ab Netscape Navigator 4.06, IE5 und IE6)

Tabelle 1.1 Die gültigen Parameter für `<script language="...">`

Parameter	Bedeutung
JavaScript1.4	Netscape 6
JavaScript1.5	Netscape 6

Tabelle 11 Die gültigen Parameter für `<script language="...">` (Forts.)

Mit der folgenden HTML-Seite (**version.html** auf der CD-ROM) können Sie überprüfen, welche JavaScript-Versionen der jeweilige Browser unterstützt.

```

<html>
<head>
<title>JavaScript</title>
</head>
<body>
<script language="JavaScript"><!--
document.write("Der Browser unterstützt JavaScript<hr />");
//--></script>
<script language="JavaScript1.1"><!--
document.write("Der Browser unterstützt JavaScript v1.1<hr />");
//--></script>
<script language="JavaScript1.2"><!--
document.write("Der Browser unterstützt JavaScript v1.2<hr />");
//--></script>
<script language="JavaScript1.3"><!--
document.write("Der Browser unterstützt JavaScript v1.3<hr />");
//--></script>
<script language="JavaScript1.4"><!--
document.write("Der Browser unterstützt JavaScript v1.4<hr />");
//--></script>
<script language="JavaScript1.5"><!--
document.write("Der Browser unterstützt JavaScript v1.5");
//--></script>
</body>
</html>

```

Listing 14 Die unterstützten JavaScript-Versionen werden angezeigt.

In Abbildung 1.2 sehen Sie beispielsweise, was der Internet Explorer 6 ausgibt.

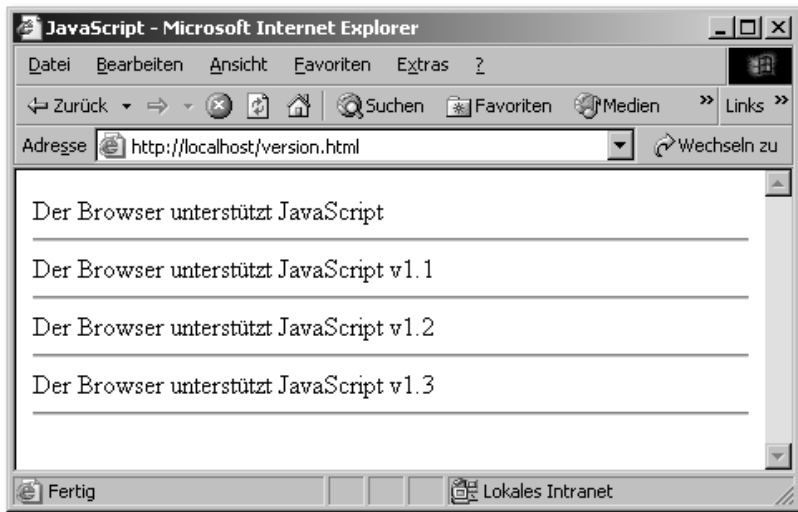


Abbildung 1.2 Die vom IE6 unterstützten JavaScript-Versionen



Beachten Sie bei obigem Beispiel, dass Sie mit `document.write()` insbesondere auch HTML-Code, in diesem Fall das `<hr>`-Element für eine horizontale Linie, ausgeben können.

In der folgenden Tabelle sehen Sie, welche Parameter ausgewählte Browserversionen erkennen bzw. unterstützen:

Browser	Unterstützte Parameter
Netscape 4.00-4.05	JavaScript1.0-JavaScript1.2
Netscape 4.06-4.76	JavaScript1.0-JavaScript1.3
Netscape 6	JavaScript1.0-JavaScript1.5
Internet Explorer 4	JavaScript1.0-JavaScript1.2
Internet Explorer 5-6	JavaScript1.0-JavaScript1.3

Tabelle 1.2 Die von ausgewählten Browsern unterstützten Parameter `<script language="...">`

Durch die Verwendung spezieller `language`-Attribute können Sie Fehlermeldungen vermeiden, die bei der Verwendung von zu modernen JavaScript-Kommandos erscheinen würden. Wenn Sie also Sprachelemente von JavaScript verwenden, die erst ab Version 1.1 unterstützt werden,

könnten Sie das `language`-Attribut auf `JavaScript1.1` setzen; ältere Browser ignorieren dann die Befehle völlig.

Das stimmt leider nicht immer. Einige Versionen des Netscape Navigator 3 beispielsweise haben einen Bug und versuchen, auch JavaScript-Blöcke mit `language="JavaScript1.2"` und `language="JavaScript1.3"` zu interpretieren. Auch der Internet Explorer führt manchmal Code aus, der nicht für ihn bestimmt ist.



Von diesem Ansatz, also der Angabe der JavaScript-Version, wird mittlerweile allerdings Abstand genommen. Stattdessen wird im `<language>`-Tag der `type`-Parameter angegeben; als Wert wird `"text/javascript"` verwendet:

```
<script type="text/javascript"><!--  
document.write("ActionScript mit JavaScript");  
//--></script>
```

1.3.3 Exkurs: Kommentare

Vermutlich haben Sie sich sowieso schon über die Schrägstriche `//` vor dem `-->` gewundert. Damit leiten Sie einen Kommentar im JavaScript-Code ein. Die Verwendung von Kommentaren ist bei der Programmierung sehr wichtig, damit Sie auch Wochen später noch wissen, was Sie damals eigentlich beabsichtigt haben, und damit auch andere Leute mit dem Code arbeiten können. Es gibt zwei Arten von Kommentaren:

- ▶ `//`: Hiermit wird ein einzeliger Kommentar eingeleitet; alles, was hinter den beiden Querstrichen in der jeweiligen Zeile steht, wird vom JavaScript-Interpreter ignoriert.
- ▶ `/*` und `*/`: Hiermit wird ein mehrzeiliger Kommentar eingeleitet; alles, was nach `/*` steht, wird vom JavaScript-Interpreter ignoriert, bis zur Zeichenfolge `*/`, die den Kommentar abschließt.

Im folgenden Listing (**kommentare.html** auf der CD-ROM) sehen Sie Beispiele für Kommentare.

```
<html>  
<head>  
<title>JavaScript</title>  
</title>  
<body>  
<script language="JavaScript"><!--  
// Hier wird auf generelle JavaScript-Unterstützung geprüft
```



```

document.write("Der Browser unterstützt JavaScript<hr />");
//--></script>
<script language="JavaScript1.1"><!--
/* Hier geht es um JavaScript 1.1 */
document.write("Der Browser unterstützt JavaScript v1.1<hr />");
//--></script>
<script language="JavaScript1.2"><!--
// JavaScript 1.2 wird überprüft
// und zwar gründlich
document.write("Der Browser unterstützt JavaScript v1.2<hr />");
//--></script>
<script language="JavaScript1.3"><!--
/* JavaScript 1.3 wurde
    mit Netscape Navigator 4.06 eingeführt */
document.write("Der Browser unterstützt JavaScript v1.3<hr />");
//--></script>
<script language="JavaScript1.4"><!--
document.write("Der Browser unterstützt JavaScript v1.4<hr />");
//--></script>
<script language="JavaScript1.5"><!--
document.write("Der Browser unterstützt JavaScript v1.5");
//--></script>
</body>
</html>

```

Listing 1.5 Das Listing wurde um Kommentare erweitert.

1.3.4 Externe Dateien verwenden

Insbesondere wenn Sie HTML-Code wiederverwenden möchten, ist es sinnvoll, häufig benötigte Codestücke in externe Dateien auszulagern. Der Vorteil liegt auf der Hand. Stellen Sie sich vor, es gibt ein Stück JavaScript-Code, das Sie auf allen Ihren Webseiten verwenden. Nun stehen Sie jedoch plötzlich vor der Situation, dass Sie diesen Code ändern müssen. Hätten Sie nun auf jeder einzelnen dieser Seiten den Code wie zuvor gezeigt eingebaut, müssten Sie alle Seiten öffnen und diesen Code ändern. Bei großen Websites ist das ein unvermeidbar hoher Aufwand.

Liegt dagegen der Code in einer externen Datei, können Sie in alle Seiten einen Verweis auf diese Datei einbauen. Dann muss nur noch diese eine Datei geändert werden, der Effekt macht sich auf allen Seiten bemerkbar.

Die Standard-Dateiendung für solche externen JavaScript-Dateien ist .js (was allerdings keine Pflicht ist). Betrachten Sie die folgende Beispieldatei, die sie auf der CD-ROM unter dem Dateinamen **extern.js** finden:

```
document.write("ActionScript mit JavaScript");
```

Sie sehen – diese Datei enthält lediglich JavaScript-Code, keine `<script>`-Tags oder ähnliches Beiwerk.

Um die Datei einzubinden, müssen Sie das `src`-Attribut des `<script>`-Tags auf den Dateinamen setzen (kompletter Code: **einbinden.html**):

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

Der Code aus der externen Datei wird an der angegebenen Stelle geladen und ausgeführt.

Wenn Sie das `src`-Attribut verwenden, wird alles zwischen `<script>` und `</script>` ignoriert! Sie können also pro `<script>`-Element **entweder** eine externe Datei einbinden **oder** direkt Code unterbringen, nicht jedoch beides.



1.4 JavaScript-Links

JavaScript-Befehle werden oft aufgrund von Benutzereingaben ausgeführt. Eine Möglichkeit besteht darin, aufgrund eines Mausklicks eine Aktion zu starten. Bevor wir ein paar Beispiele vorstellen, muss noch ein neuer JavaScript-Befehl eingeführt werden. Mit `window.alert("ActionScript mit JavaScript")` wird ein modales Fenster ausgegeben, das den Text »ActionScript mit JavaScript« anzeigt. Je nach verwendetem Browser sieht das etwas anders aus, in Abbildung 1.4 sehen Sie die Darstellung im Internet Explorer 6.



Abbildung 1.3 Das »Warnfenster« im Internet Explorer

HTML-Links können auf URLs mit den verschiedensten Protokollen verweisen, so zum Beispiel **http:**, **ftp:**, **news:** oder **mailto:**. Nun kommt ein

Protokolle

weiteres Protokoll hinzu, welches jedoch lediglich im Zusammenhang mit JavaScript eine Bedeutung hat: **javascript:**. Folgender Link gibt »ActionScript mit JavaScript« in einem modalen Fenster aus, wenn man ihn anklickt:

```
<a href="javascript:window.alert('ActionScript mit JavaScript');">xxx</a>
```

Beachten Sie die Anführungszeichen. In JavaScript ist es prinzipiell egal, ob Sie einfache oder doppelte Anführungszeichen verwenden – Hauptsache, Sie hören so auf, wie Sie angefangen haben.

Folgendes ist also völlig korrekt:

```
document.write("ActionScript ");  
document.write('<br />mit JavaScript');
```

Falsch ist dagegen:

```
document.write("ActionScript ");  
document.write('<br />mit JavaScript');
```

Obwohl man mit dem Pseudo-Protokoll **javascript:** bequem JavaScript-Befehle aufrufen kann, gibt es einen kleinen Nachteil: Browser, die kein JavaScript unterstützen, geben eine Fehlermeldung aus. Im folgenden Abschnitt wird diesem Missstand abgeholfen.

1.5 Event-Handler

Ohne anderen Kapiteln allzu sehr vorgreifen zu wollen – wenn Sie bestimmte Aktionen tätigen, beispielsweise die Maus bewegen oder einen Link anklicken, tritt JavaScript-intern ein so genanntes **Ereignis** (engl. **event**) ein. Manche dieser Ereignisse können Sie mit JavaScript abfangen und auf sie reagieren. Dazu benötigen Sie **Event-Handler**, die als Attribute mancher HTML-Tags in den HTML-Code eingebunden werden. Die wichtigsten Handler stellen wir Ihnen in den jeweiligen Kapiteln anhand eines Beispiels vor.

Grundsätzlich gilt Folgendes: Ein Event-Handler beginnt stets mit **on**. Der Event-Handler, der zuständig ist, wenn ein Link angeklickt wird, heißt **onclick**. Wenn der folgende Link angeklickt wird, wird ein modales Fenster geöffnet.

```
<a href="irgendwohin.html" onclick="alert('ActionScript mit JavaScript');">Galileo Press</a>
```

Vom Prinzip her wunderbar – Browser, die JavaScript unterstützen, führen den Code aus, folgen danach aber dem Link (später dazu mehr). Ältere Browser »sehen« nur den Link und würden versuchen, das Ziel des Links aufzurufen. Das ist aber in der Regel nicht erwünscht. Das `href`-Attribut des Links muss gesetzt werden, sonst wird er nicht angezeigt. Es gibt aber eine Möglichkeit, einen Link anzugeben, der keine neue Seite lädt:

```
<a href="#" onclick="alert('ActionScript mit Java-  
Script');">Galileo Press</A>
```



Diesen Trick sollten Sie sich gut merken!

Für diese Event-Handler hat ein bis dato allzu vernachlässigtes HTML-Element eine Art Revival gefeiert; die Rede ist von `<input type="button">`. Damit wird eine eigentlich funktionslose HTML-Schaltfläche definiert. Mit JavaScript jedoch hat sie durchaus eine Daseinsberechtigung. Auch dieses HTML-Element kennt `onclick`; auf Knopfdruck (bzw. Mausklick) kann JavaScript-Code ausgeführt werden.

Folgendes Beispiel demonstriert dies: Auf Mausklick wird wie zuvor `window.alert()` aufgerufen.

```
<html>  
<head>  
<title>Schaltflächen</title>  
</head>  
<body>  
<form>  
  <input type="button" value="Klick mich!"  
    onclick="window.alert('ActionScript mit JavaScript');" />  
</form>  
</body>  
</html>
```

Listing 1.6 Der JavaScript-Code gehorcht auf Mausklick.

Vergessen Sie nicht, um die Schaltfläche herum ein `<form>`-Element zu platzieren, sonst zeigen einige Browser die Schaltfläche gar nicht erst an!





Sie finden dieses Beispiel auf der CD-ROM zum Buch unter dem Dateinamen **schaltflaeche.html**.

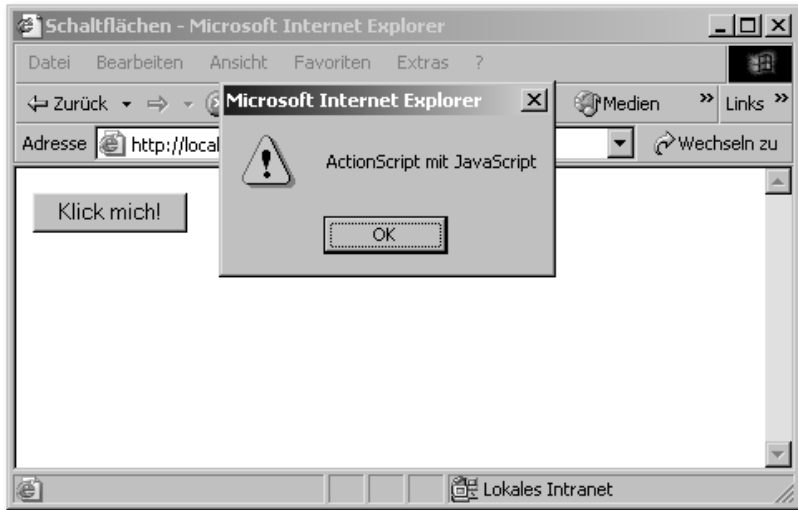


Abbildung 1.4 Auf Mausklick wird das Warnfenster geöffnet.

2 Spracheinführung

Die Sprache ist natürlich im ersten Moment immer ein Hindernis für die Verständigung.

– Marcel Marceau

In diesem Kapitel werden die Grundprinzipien der Programmierung im Allgemeinen und der Programmierung mit JavaScript im Besonderen erläutert. Es ist zwar arm an Praxisbeispielen, aber Sie werden die hier vorgestellten Techniken in den weiteren Kapiteln noch des Öfteren benötigen. Sie erhalten hier das Rüstzeug, das die Grundlage für alle folgenden Kapitel ist.

Gut gerüstet

Leser, die bereits Erfahrungen mit der einen oder anderen Programmiersprache haben, mögen diesen kleinen Exkurs verzeihen, aber so soll auch Neulingen die Chance geboten werden, in die Materie einzusteigen. Auf der anderen Seite sind die Erklärungen recht knapp gehalten und beschränken sich auf das Nötigste. Schließlich geht es ja darum, möglichst schnell brauchbare Anwendungen zu schreiben.

2.1 Variablen

Bei der Programmierung müssen immer wieder Daten zwischengespeichert werden. Hierzu bedient man sich so genannter **Variablen** (in manchen Büchern – sehr wenigen – auch als »Veränderliche« bezeichnet).

2.1.1 Namensgebung

Jede Variable wird über ihren Namen angesprochen. Bei der Namensgebung haben Sie größtenteils freie Hand. Ein Variablenname besteht aus einer Folge von Buchstaben, Ziffern und dem Unterstrich (_). Einzige Bedingung: Das erste Zeichen darf keine Ziffer sein. Außerdem wird zwischen Groß- und Kleinschreibung unterschieden. Die Bezeichner `ActionScript`, `actionscript` und `ACTIONSCRIPT` sind also verschiedene Variablen. Beispiele für Variablennamen sind etwa:

- ▶ `ActionScript_mit_JavaScript`
- ▶ `ActionScript`
- ▶ `ASJS2002`
- ▶ `_ActionScript_`

JavaScript-Schlüsselworte und -begriffe dürfen Sie nicht als Variablennamen verwenden. Ein Beispiel hierfür ist etwa `alert`, das Sie im vorangegangenen Kapitel schon einmal in Aktion gesehen haben.

Um einer Variablen einen Wert zuzuweisen, wird das Gleichheitszeichen verwendet. Links vom Gleichheitszeichen steht der Variablenname, rechts davon der neue Wert (oder eine andere Variable, deren Wert dann zugewiesen wird).

2.1.2 Numerische Variablen

Es gibt verschiedene Typen von Variablen. Zahlenwerte werden wie im amerikanischen Zahlensystem, also mit einem Dezimalpunkt statt mit einem Dezimalkomma, angegeben:

```
Pi = 3.14159265;  
Mauerfall = 1989;  
MinusHundert = -100;
```

2.1.3 Zeichenketten

Sehr oft werden Zeichenketten, auch **Strings** genannt, verwendet. Die zweite Bezeichnung kommt daher, dass es ein so benanntes Objekt in JavaScript gibt, aber dazu später mehr. In diesem Buch werden beide Begriffe äquivalent verwendet.

Ein String wird von Anführungszeichen eingeschlossen, entweder von einfachen (Apostrophen) oder von doppelten. Hierbei ist zu beachten, dass unbedingt gerade Anführungszeichen (`"` + `"`) und Apostrophe (`'` + `'`) verwendet werden. Im Gegensatz zu beispielsweise Perl ist es hier egal, ob einfache oder doppelte Anführungszeichen verwendet werden, Hauptsache, die Zeichenkette wird mit derselben Art von Anführungszeichen beendet, mit denen sie eingeleitet worden ist.

```
Hauptsprache = "ActionScript";  
AndereSprache = 'JavaScript';
```

Die verschiedenen Anführungszeichen haben unter anderem folgenden Sinn: Wenn Sie beispielsweise ein Apostroph in einer Zeichenkette verwenden wollen, können Sie diese Zeichenkette ja schlecht mit Apostrophen eingrenzen, da der JavaScript-Interpreter dann nicht weiß, wo die Zeichenkette aufhört. In diesem Fall müssen Sie die doppelten Anführungszeichen verwenden:

```
Frank = "There's no business like showbusiness";
```

Wenn man aber in der Verlegenheit ist, beispielsweise beide Arten von Anführungszeichen in einer Zeichenkette verwenden zu müssen, kommt man in Schwierigkeiten. Hier hilft der Backslash (\) weiter. Das dem Backslash folgende Zeichen wird »entwertet«, d. h. es hat in der Zeichenkette keine besondere Bedeutung. Beim Anführungszeichen oder Apostroph bedeutet das: Die Zeichenkette wird hiermit nicht beendet.

```
Frank = 'There\'s no business like showbusiness';
```

Wenn man nun den Backslash selbst in der Zeichenkette verwenden will, muss man auch ihn entwerten:

```
WindowsVerzeichnis = "C:\\WINDOWS";
```

Mit dem Backslash können auch einige besondere Zeichen dargestellt werden. Tabelle 2.1 zeigt eine Übersicht.

Ausdruck	Bedeutung
\r	Wagenrücklauf
\n	Neue Zeile
\t	Tabulator
\b	Backspace (Löschaste)
\f	Seitenvorschub

Tabelle 2.1 Sonderzeichen innerhalb von Zeichenketten

2.1.4 Boolesche Variablen

Oft ist man nicht an einem Zahlenwert oder einem String interessiert, sondern an einem Wahrheitswert, also wahr oder falsch. Man spricht hier auch von booleschen Variablen. Solche Variablen können als Wert nur `true` (wahr) oder `false` (falsch) annehmen. Später werden Sie sehen, dass boolesche Werte indirekt auch bei einigen anderen JavaScript-Konstrukten vorkommen.

```
Ist2000einSchaltjahr = true;  
Ist3000einSchaltjahr = false;
```

2.1.5 Variablendeklaration

Wie Sie bereits gesehen haben, kann mit dem Gleichheitszeichen einer Variablen ein Wert zugewiesen werden. Ein Variablenname kann auch öfter verwendet werden und ihm kann auch mehrmals ein Wert zugewie-

sen werden, es handelt sich dann jedoch immer um dieselbe Variable. Sehr oft findet man in der Praxis das Schlüsselwort `var` vor der ersten Verwendung einer Variablen. Dies dient der Initialisierung der Variablen und wird ab hier der Übersichtlichkeit halber in diesem Buch konstant verwendet. So sieht man auf den ersten Blick, welche Variable schon einmal deklariert worden ist und welche nicht. Später, bei der Einführung von Funktionen, wird das Schlüsselwort noch eine besondere Bedeutung erhalten.

Beachten Sie, dass Sie das Schlüsselwort `var` nur einmal pro Variable verwenden sollten. Richtig ist also Folgendes:

```
var FlashVersion = 5;  
// Flash MX ist erschienen  
FlashVersion = 6;
```

Nicht so gut ist dagegen Folgendes:

```
var FlashVersion = 5;  
// Flash MX ist erschienen  
var FlashVersion = 6;
```

In diesem und auch in einigen anderen Beispielen in diesem Buch sind gegenüber einem vorhergehenden Listing veränderte Programmzeilen durch Fettdruck hervorgehoben.

2.2 Operatoren

Durch Operatoren wird eine gewisse Anzahl von Variablen miteinander kombiniert. Beispiele für Operatoren sind die Grundrechenarten. Durch den Plus-Operator werden zwei Zahlen miteinander kombiniert, und als Ergebnis erhält man die Summe dieser beiden Zahlen. Man unterscheidet – auch je nach Typ der beteiligten Variablen – verschiedene Arten von Operatoren.

2.2.1 Arithmetische Operatoren

Diese Art von Operatoren arbeitet mit numerischen Variablen. Sie sollten also sicherstellen, dass auch wirklich Zahlenvariablen vorliegen, sonst könnten Sie eine Fehlermeldung erhalten. Tabelle 2.2 zeigt alle arithmetischen Operatoren anhand eines Beispiels.

Operator	Beschreibung	Beispiel	Ergebnis (Wert von a)
+	Addition	a = 7 + 4	11
-	Subtraktion	a = 7 - 4	3
*	Multiplikation	a = 7 * 4	28
/	Division	a = 7 / 4	1.75
%	Modulo (Restrechnung) ¹	a = 7 % 4	3
-	Negation	b = 7 a = -b	-7

Tabelle 2.2 Arithmetische Operatoren

Wollen Sie eine Variable um einen bestimmten Wert erhöhen, können Sie sich des folgenden Konstrukts bedienen:¹

```
FlashVersion = FlashVersion + 5;
```

Der Variablen `FlashVersion` wird als Wert der alte Wert dieser Variablen plus fünf zugewiesen, der Wert der Variablen wird also de facto um fünf erhöht. In der Praxis kommt es sehr häufig vor, dass der Wert einer Variablen um genau eins erhöht oder verringert werden soll; für diesen Fall sieht JavaScript eine Abkürzung vor:

- `FlashVersion++` erhöht den Wert der Variablen um eins.
- `FlashVersion--` verringert den Wert der Variablen um eins.

Die Operatoren `++` und `--` können auch direkt vor dem Variablennamen stehen. Der Unterschied liegt in der Reihenfolge, in der diese Operation im Vergleich zu anderen Operationen ausgeführt werden soll. Am Beispiel des Operators `++` soll das einmal durchexerziert werden; `--` verhält sich analog.

Das Endergebnis des Stand-alone-Ausdrucks

```
FlashVersion++
```

ist völlig äquivalent zu dem von

```
++FlashVersion
```

Einen Unterschied stellt man jedoch fest, wenn der Ausdruck bei einer Zuweisung verwendet wird:

¹ Siehe Beispiel: 7 Modulo 4 liefert 3, weil 7 bei der Division durch 4 den Rest 3 lässt. Analog ist beispielsweise 11 Modulo 4 auch 3.

```
FlashVersion = 5;
var Version1 = ++FlashVersion;
var Version2 = FlashVersion++;
```

Welchen Wert hat `Version1`, welchen Wert hat `Version2`?

Die (vielleicht etwas überraschende) Antwort lautet: `Version1` hat den Wert 6, `Version2` hat auch den Wert 6. Betrachten Sie zunächst die zweite Zeile:

```
var Anzahl1 = ++FlashVersion;
```

Der `++`-Operator steht vor dem Variablennamen, das bedeutet hier, es wird zunächst diese Operation ausgeführt (`FlashVersion` um eins erhöhen) und dann wird der neue Wert (6) der Variablen `Anzahl1` zugewiesen.

Bei der dritten Zeile ist es genau andersherum:

```
var Anzahl2 = FlashVersion++
```

Zuerst wird der Variablen `Anzahl2` der (aktuelle) Wert von `FlashVersion` zugewiesen, dann wird der Wert von `FlashVersion` um eins vergrößert.

Wenn man den Wert einer Variablen nicht um exakt eins erhöhen oder verringern will, kann man sich einer anderen Abkürzung bedienen. Diese Abkürzung existiert für jede der vier Grundrechenarten sowie für den Modulo-Operator:

Operator	Bedeutung	Langform	Kurzform
<code>+=</code>	Addition	<code>a = a + b</code>	<code>a += b</code>
<code>-=</code>	Subtraktion	<code>a = a - b</code>	<code>a -= b</code>
<code>*=</code>	Multiplikation	<code>a = a * b</code>	<code>a *= b</code>
<code>/=</code>	Division	<code>a = a / b</code>	<code>a /= b</code>
<code>%=</code>	Modulo	<code>a = a % b</code>	<code>a %= b</code>

Tabelle 2.3 Abkürzungen für arithmetische Operationen

Auch in JavaScript gilt, dass Punktrechnung vor Strichrechnung geht, d. h. Multiplikationen und Divisionen werden vor Additionen und Subtraktionen ausgeführt. Der folgende Ausdruck liefert also 7 und nicht 9:

```
var PunktVorStrich = 1 + 2 * 3;
```

2.2.2 Boolesche Operatoren

Mit Logikoperatoren (oder booleschen Operatoren) können Sie Wahrheitswerte miteinander verknüpfen. Die Bedeutung der Operatoren ist die mathematische Bedeutung, nicht unbedingt die umgangssprachliche Bedeutung. Aus diesem Grund werden die einzelnen Operatoren explizit behandelt.

UND (&&)

Nur, wenn beide (bzw. alle) beteiligten Variablen den Wert `true` haben, liefert die Operation `true` zurück, ansonsten `false`.

```
var t = true;
var f = false;
var bool1 = t && f; //liefert false
var bool2 = t && t; //liefert true
```

ODER (||)

Ist eine der beteiligten Variablen `true`, so liefert die Operation `true` zurück. Das Ergebnis ist nur dann `false`, wenn alle Variablen den Wert `false` haben. Hier liegt ein Unterschied zur Alltagssprache vor, denn dort bedeutet »oder« eher »entweder-oder«: Nur dann ist das Ergebnis `true`, wenn genau eine der beteiligten Variablen den Wert `true` hat.

```
var t = true;
var f1 = false;
var f2 = false;
var bool1 = t || f1 || f2; //liefert true
var bool2 = f1 || f2; //liefert false
```

NEGATION (!)

Der Negationsoperator macht `true` zu `false` und `false` zu `true`.

```
var t = true;
var f = false;
var bool1 = !t; //liefert false
var bool2 = !f; //liefert true
```

Short Evaluation

Wie Sie bereits gesehen haben, genügt genau eine Variable mit dem Wert `true`, damit das Ergebnis einer Oder-Verknüpfung ganz sicher den Wert

`true` hat. Analog liefert eine Und-Verknüpfung auf jeden Fall den Wert `false`, wenn eine Variable den Wert `false` hat.

JavaScript benutzt hier das Prinzip der so genannten Short Evaluation (wörtlich: kurze Auswertung), zumindest gilt dies für die existierenden Implementierungen. Bei einer Und- bzw. Oder-Verknüpfung werden die beteiligten Variablen von links nach rechts durchgegangen. Sollte bei einer dieser Variablen aufgrund ihres Werts das Ergebnis der gesamten Operation schon feststehen, wird der Rest gar nicht mehr ausgewertet.

Vergleichsoperatoren

Vergleichsoperatoren werden meistens bei Zahlenwerten verwendet. Auch bei Zeichenketten sind diese Vergleiche möglich, hier richtet sich die Rangfolge der einzelnen Zeichen (welches Zeichen ist »größer« als ein anderes?) nach dem ASCII-Code des Zeichens.

Operator	Beschreibung	Beispiel	Ergebnis (Wert für a)
<code>==</code>	Gleich	<code>a = (3 == 4)</code> <code>a = ("ActionScript" == "JavaScript")</code>	false
<code>!=</code>	Ungleich	<code>a = (3 != 4)</code> <code>a = ("ActionScript" != "JavaScript")</code>	true
<code>></code>	Größer als	<code>a = (3 > 4)</code>	false
<code><</code>	Kleiner als	<code>a = (3 < 4)</code>	true
<code>>=</code>	Größer oder gleich	<code>a = (3 >= 4)</code>	false
<code><=</code>	Kleiner oder gleich	<code>a = (3 <= 4)</code>	true

Tabelle 2.4 Vergleichsoperatoren



Eine der häufigsten Fehlerquellen besteht in der Verwechslung der Zuweisung `=` mit dem Vergleichsoperator `==`. Ab JavaScript Version 1.3 gibt der Interpreter eine Fehlermeldung aus, wenn offensichtlich ein Vergleich durchgeführt werden soll, aber der Zuweisungsoperator verwendet wird.

String-Operatoren

Auch mit Zeichenketten können Sie »rechnen«; man kann zwei Zeichenketten aneinander hängen. Hierzu wird auch der Plus-Operator `(+)` verwendet:

```
var Sprachel = "ActionScript";
var Sprache2 = "JavaScript";
var Titel = Sprachel + " und " + Sprache2;
//liefert "ActionScript und JavaScript"
```

Ansonsten können Sie mit Zeichenketten nicht rechnen. Dennoch sollen an dieser Stelle noch drei Möglichkeiten vorgestellt werden, wie Sie mit Zeichenketten etwas anfangen können:

- ▶ `Zeichenkette.length`: Liefert die Anzahl der Zeichen in einer Zeichenkette zurück.
- ▶ `Zeichenkette.charAt(x)`: Liefert das Zeichen an Position `x` in der Zeichenkette zurück. Dabei beginnt die Zählung bei 0, das vierte Zeichen erhält man also mit `Zeichenkette.charAt(3)`.
- ▶ `Zeichenkette.substring(start, ende)`: Liefert eine Teilzeichenkette zurück, und zwar ab dem Zeichen an Position `start` (Zählung beginnt wieder bei 0) und bis zu dem Zeichen vor dem Zeichen an der Position `ende`.

Hierzu ein kleines Beispiel. In den Variablen `a` und `b` stehen das erste Zeichen der Zeichenkette und die folgenden Zeichen:

```
var z = "ActionScript";
var a = z.charAt(0);
var b = z.substring(1, z.length);
```

Umwandlung des Variablentyps

Die vorgestellten Operatoren können auch dazu verwendet werden, um die einzelnen Variablentypen umzuwandeln. JavaScript ist in Sachen Variablentyp nicht so strikt wie andere Programmiersprachen. Eine Variable kann ihren Typ auch während des Programmablaufs ändern. Beispielsweise werden Sie in einem späteren Kapitel feststellen, dass Formulareingaben stets als Zeichenketten vorliegen. Wenn Sie sich aber sicher sind, dass die Zeichenkette eine korrekt formatierte Zahl enthält, können Sie JavaScript dazu zwingen, die Variable als Zahlenwert zu betrachten. Der Trick besteht darin, die Variable mit eins zu multiplizieren (oder 0 zu addieren). Eine Multiplikation kann nur mit Zahlenwerten durchgeführt werden, so dass JavaScript die Variable in eine Zahl umwandelt – und eine Multiplikation mit eins ändert am Wert der Zahl auch nichts.

Außerdem ist es manchmal notwendig, eine boolesche oder numerische Variable in eine Zeichenkette umzuwandeln. Diesmal muss der

Konkatenationsoperator (Verkettungsoperator), das Plus, verwendet werden. Wenn eine Variable mit einer leeren Zeichenkette konkateniert (verkettet) wird, erhält man als Ergebnis eine Zeichenkette, ändert aber ansonsten den Wert der Variablen nicht.

```
var FlashVersion = "6";  
FlashVersion *= 1; //Zeichenkette in Zahl  
var wahrheitswert = true;  
wahrheitswert += ""; //Wahrheitswert in Zeichenkette  
var Version = 6;  
Version += ""; //Zahl in Zeichenkette
```

JavaScript führt zwar eine automatische Typkonvertierung durch, aber nicht immer in die gewünschte Richtung:

```
var summe = "47" + 11 //liefert "4711" und nicht etwa 58
```

2.3 Kontrollstrukturen: Schleifen

Hin und wieder kommt es vor, dass eine Anweisung mehrmals ausgeführt werden muss, beispielsweise bei einer Aufzählung. Hierzu bietet JavaScript mehrere Kontrollstrukturen an. Mit JavaScript Version 1.2 (sowie beim Internet Explorer 4) wurden neue Kontrollstrukturen eingeführt, die aber mit dem Befehlssatz von JavaScript 1.0 vollständig nachgebildet werden können.

2.3.1 for-Schleifen

Diese Art der Schleife führt eine Anweisung eine (in der Regel) bestimmte Anzahl von Malen aus. Die Syntax sieht dabei wie folgt aus:

```
for (Initialisierung; Bedingung; Befehlsfolge){  
    //Anweisungen  
}
```

Die `for`-Schleife hat drei Parameter:

- **Initialisierung:** Oft läuft bei einer Schleife eine Zählvariable mit, die die Anzahl der Wiederholungen zählt. Diese Variable kann hier initialisiert werden. Sollen mehrere Variablen initialisiert werden, so werden die einzelnen Anweisungen durch Kommata voneinander getrennt.
- **Bedingung:** Die `for`-Schleife wird so lange ausgeführt, bis diese Bedingung nicht mehr erfüllt ist.

- **Befehlsfolge:** Nach jedem Durchlauf der Anweisungen wird diese Befehlsfolge (in der Regel ein Befehl; mehrere Befehle werden durch Kommata voneinander getrennt) ausgeführt. Wenn die Schleife irgendwann enden soll, sollten hier in der Regel Befehle ausgeführt werden, die dafür sorgen, dass nach einer bestimmten Anzahl von Durchläufen die Bedingung (den zweiten Parameter) nicht mehr erfüllt werden kann.

Die geschweiften Klammern um den Anweisungsblock sind dann zwingend, wenn der Block aus mehr als einem Befehl besteht. Handelt es sich um nur einen Befehl, so können Sie die geschweiften Klammern weglassen. Sie sollten aber zumindest den Code einrücken, damit das Ganze übersichtlich und lesbar bleibt.

Der folgende Code gibt zehnmal `ActionScript` aus. Dabei wird eine Zählvariable mit 0 initialisiert und in jedem Schleifendurchlauf um eins erhöht. Die Abbruchbedingung prüft, ob die Zählvariable kleiner als zehn ist. Vor dem elften Durchlauf wird die Variable auf zehn erhöht und die Schleife wird verlassen.

```
for (var i=0; i<10; i++)  
    document.write("ActionScript<br />");
```

Sehr oft wird die Zählvariable auch direkt in der Schleife verwendet. Der folgende Code gibt alle Quadratzahlen von $0 = 0$ bis $10 = 100$ aus:

```
for (var i=0; i<=10; i++){  
    document.write("Das Quadrat von " + i + " ist: ");  
    document.write(i*i + "<br />");  
}
```

Sie sehen hier die Verwendung des Plus-Operators: Eine Zeichenkette und eine Zahl werden zusammengefügt. Das Ergebnis ist eine Zeichenkette, die Zahl wird also in eine Zeichenkette umgewandelt.

Beachten Sie ebenso, dass Sie einen Zeilenumbruch nicht mit `\r\n` angeben sollten, sondern mit dem entsprechenden HTML-Tag, `<br />`.

Wie bereits erwähnt wurde, können Sie auch mehrere Zählvariablen verwenden, die Sie dann durch Kommata voneinander trennen müssen. Anwendungen gibt es ziemlich selten, hier folgt ein sehr praxisfremdes Beispiel. Es werden zwei Zählvariablen verwendet, `i` und `j`. Die erste enthält eine Zahl, die zweite eine Zeichenkette. Der Zahlenwert wird in eine Zeichenkette umgewandelt und an `j` angehängt. Sobald `j` mehr als 15 Zeichen enthält, wird die Schleife verlassen.


```
for (var i=1, j="1"; j.length<=15; i++, j += i)
    document.write(i + " - " + j + "<br />");
```

Dieses Programm gibt Folgendes auf dem Bildschirm aus:

```
1 - 1
2 - 12
3 - 123
4 - 1234
5 - 12345
6 - 123456
7 - 1234567
8 - 12345678
9 - 123456789
10 - 12345678910
11 - 1234567891011
12 - 123456789101112
```

Vor dem nächsten Schleifendurchlauf würde an die Zeichenkette 13 angehängt, die Länge würde dadurch auf 17 Zeichen anwachsen, also wird die Schleife hier verlassen.

2.3.2 do-while-Schleife

Browser Nicht immer weiß man, wie oft ein Anweisungsblock hintereinander ausgeführt werden soll. Stattdessen will man den Block so lange ausführen, bis eine Bedingung nicht mehr erfüllt ist. Im folgenden Beispiel sollen in einer Zeichenkette alle As durch Bs ersetzt werden. Hierbei ist die Methode `Zeichenkette.indexOf(Teilstring)` nützlich; diese gibt nämlich zurück, an welcher Position in der Zeichenkette der Teilstring zum ersten Mal vorkommt. Ist dieser nicht in der Zeichenkette enthalten, wird -1 zurückgegeben. Das erste Zeichen steht, wie auch schon vorher, an Position 0. Diese Art von Schleife gibt es jedoch erst seit dem Netscape Navigator 4 und dem Internet Explorer 4.

Die Syntax ist die folgende:

```
do {
    //Anweisungsblock
} while (Bedingung)
```

Der Anweisungsblock wird ausgeführt, dann wird die Bedingung überprüft. Ist sie erfüllt, wird der Block erneut ausgeführt (und dann wieder die Bedingung geprüft); andernfalls wird die Schleife verlassen.

```

var Zk = "AXAYAZ"; // Zk steht für "Zeichenkette"
do{
    io = Zk.indexOf("A");
    Zk = Zk.substring(0, io) + "B" +
        Zk.substring(io+1, Zk.length);
} while (Zk.indexOf("A") > -1);

```

Nach dem Durchlauf enthält die Zeichenkette den Wert BXBYBZ.

Beachten Sie insbesondere, dass die Schleife auf jeden Fall einmal ausgeführt wird! Im Beispiel führt das zu einem Fehler, wenn die Zeichenkette zu Anfang überhaupt kein A enthält. Im Folgenden lernen Sie Methoden kennen, wie Sie dies vermeiden können.

Noch ein Wort zur Zuweisung: Wenn Sie bei einer Zeichenkette *z* das Zeichen an Stelle *x* in das Zeichen *x* ändern wollen (Zählung beginnt wie immer bei 0), kommen Sie mit folgender Anweisung weiter:

```
z = z.substring(0, x) + "X" + z.substring(x+1, z.length);
```

Alle Zeichen vor und hinter dem zu ändernden Zeichen bleiben durch die beiden `substring()`-Anweisungen erhalten.

2.3.3 while-Schleife

Schon seit der JavaScript-Version 1.0 gibt es eine weitere Form der Schleifen, und zwar `while`-Schleifen (ohne `do`). Die Syntax ist der von `do-while`-Schleifen sehr ähnlich, der Unterschied steckt im Detail:

```

while (Bedingung){
    //Anweisungsblock
}

```

Die Bedingung wird also hier **vor** dem Durchlaufen des Anweisungsblocks überprüft. Im Beispiel von oben, der Ersetzung aller As durch Bs, ist das sehr nützlich, da hier der Anweisungsblock nicht ausgeführt wird, wenn die Zeichenkette von Anfang an keine As enthält:

```

var Zk = "AXAYAZ";
while (Zk.indexOf("A")>-1) {
    io = Zk.indexOf("A");
    Zk = Zk.substring(0, io) + "B" +
        Zk.substring(io+1, Zk.length);
}

```

2.3.4 for-in-Schleife

Diese Schleife wird recht selten verwendet, und an dieser Stelle fehlt Ihnen noch das Grundwissen über Objekte, um Ihnen eine ausreichende Erklärung geben zu können. Prinzipiell sei gesagt, dass man hier mit der Schleife durch alle Eigenschaften eines Objekts laufen kann. Die folgende Schleife gibt alle Elemente eines Arrays (ein Feld aus mehreren Variablen) wieder.

```
var sprachen = new Array("ActionScript", "JavaScript");
for (e in sprachen)
    document.write(e + "<br />");
```

2.3.5 Schleifensteuerung

Eine Schleife muss nicht unbedingt so oft durchlaufen werden, wie vorgesehen ist. Angenommen, in einer Schleife wird eine Zeichenkette auf das Vorhandensein eines bestimmten Zeichens überprüft (das geht mit `indexOf()` sehr schnell, aber das soll uns an dieser Stelle nicht interessieren). Sobald das Zeichen gefunden worden ist, muss die Schleife nicht unbedingt weiter ausgeführt werden, denn das Ergebnis (»Das Zeichen ist in der Zeichenkette enthalten«) steht ja jetzt schon fest. Aus diesem Grund gibt es den Befehl `break`, der das sofortige Verlassen der aktuellen Schleife veranlasst. Der Interpreter fährt also hinter dem aktuellen Anweisungsblock fort.

Ebenso kann es bei Schleifen, insbesondere bei `for`-Schleifen, immer wieder vorkommen, dass man zum nächsten Schleifendurchlauf springen möchte, beispielsweise, wenn man genau weiß, dass dieser Durchlauf nicht das gewünschte Ergebnis bringt, und man den Rest der Schleife aus Effizienzgründen nicht ausführen lassen möchte. Der entsprechende Befehl heißt `continue`.

Auch hier ist es leider so, dass Ihr derzeitiges Wissen noch nicht ausreicht, um Ihnen ein sinnvolles Beispiel anzugeben. Am Ende dieses Kapitels werden Sie aber in der Lage sein, die Befehle `break` und `continue` einzusetzen.

Allgemein wird die Verwendung von `break` und `continue` als eher schlechter Programmierstil angesehen; zu sehr ähneln diese Befehle dem Spaghetti-Code aus alten BASIC-Zeiten². In der Praxis werden die beiden

² Bei den Ur-Versionen von BASIC wurde jede Programmzeile anhand ihrer Nummer identifiziert. Schleifen gab es nur in sehr eingeschränkter Form, so dass einer der häufigsten Befehle `GOTO` hieß, mit dem eine Zeilennummer direkt angesprungen werden konnte. Dadurch wurde der Quellcode sehr unübersichtlich und auch unsauber.

Befehle jedoch durchaus angewandt, und zumindest im JavaScript-Bereich ist dagegen unserer Meinung nach nichts einzuwenden.

2.4 Fallunterscheidung

Eines der Hauptelemente aller JavaScript-Programme ist die Fallunterscheidung. Schon im vorangegangenen Kapitel haben Sie gesehen, dass man mit den bereits vorgestellten Mitteln schnell an seine Grenzen stößt.

2.4.1 if-Anweisung

Gehen wir zu dem Beispiel zurück, in dem alle As durch Bs ersetzt werden sollen. Mit einer `while`-Schleife geht das, aber bei einer `do-while`-Schleife hatten wir das Problem, dass es eine Fehlermeldung gibt, wenn die Zeichenkette von Anfang an kein A enthält. Der Code sah wie folgt aus:

```
var Zk = "AXAYAZ";
do{
    io = Zk.indexOf("A");
    Zk = Zk.substring(0, io) + "B" +
        Zk.substring(io+1, Zk.length);
} while (Zk.indexOf("A") > -1);
```

Wenn man die Möglichkeit hätte, innerhalb der Schleife zu überprüfen, ob die Zeichenkette noch ein A enthält, hätte man kein Problem. Natürlich gibt es eine solche Möglichkeit. Man bedient sich einer `if`-Anweisung oder einer `if-else`-Anweisung.

Die Syntax sieht wie folgt aus:

```
if (Bedingung){
    //Anweisungsblock
} else {
    //Anweisungsblock
}
```

Ist die Bedingung erfüllt, wird der erste Anweisungsblock ausgeführt; falls nicht, wird der zweite Anweisungsblock ausgeführt. Der `else`-Teil ist hier optional, und wieder können die geschweiften Klammern weggelassen werden, wenn der Anweisungsblock aus genau einer Anweisung besteht.

Wenn der erste Anweisungsblock keine geschweiften Klammern enthält und das `else` in derselben Zeile steht, muss die Anweisung mit einem Semikolon beendet werden. Falsch ist also:



```
if (a==5) b=6 else c=7
```

Richtig ist dagegen:

```
if (a==5) b=6; else c=7
```



Bei einer `if`-Anweisung ist die Gefahr besonders groß, dass statt des Gleichheitsoperators `==` die Zuweisung `=` verwendet wird. Achten Sie hier besonders auf Ihre Syntax!

Aber zurück zum Beispiel. Es gibt hier mehrere Möglichkeiten. Die Überprüfung, ob die Zeichenkette ein A enthält, ist recht einfach darzustellen, bei den Konsequenzen hat man die Wahl. Am naheliegendsten ist folgende Variante:

```
var Zk = "AXAYAZ";
do{
    io = Zk.indexOf("A");
    if (io>-1)
        Zk = Zk.substring(0, io) + "B" +
            Zk.substring(io+1, Zk.length);
} while (Zk.indexOf("A") > -1);
```

Sie können auch bei Bedarf die Schleife einfach verlassen:

```
var Zk = "AXAYAZ";
do{
    io = Zk.indexOf("A");
    if (io == -1)
        break;
    Zk = Zk.substring(0, io) + "B" +
        Zk.substring(io+1, Zk.length);
} while (Zk.indexOf("A") > -1);
```

Dann braucht man die `while`-Bedingung aber gar nicht mehr und könnte beispielsweise folgende Variante verwenden:

```
var Zk = "AXAYAZ";
do{
    io = Zk.indexOf("A");
    if (io==-1)
        break
    Zk.charAt(io) = "B";
} while (true);
```

Die Bedingung `true` ist immer erfüllt, der Anweisungsblock kann also nur durch ein `break` verlassen werden. Ob der Programmierstil des obigen Beispiels besonders gut ist, steht auf einem völlig anderen Blatt.

Eine besondere, verkürzte Variante der `if-else`-Anweisung gibt es noch. Sie hat folgende Syntax:

```
(Bedingung) ? Wert1 : Wert2
```

Ist die Bedingung erfüllt, ihr Wert also `true`, so wird `Wert1` zurückgegeben, ansonsten `Wert2`. Meistens wird diese Schreibweise bei einer Variablenzuweisung verwendet, also in der Form

```
Sprache = (Flash == "MX") ? "ActionScript" : "JavaScript";
```

Es gibt somit eine weitere Möglichkeit, in einer Zeichenkette alle `As` durch `Bs` zu ersetzen: indem alle Zeichen überprüft und gegebenenfalls ersetzt werden. Auch hier gibt es mehrere Möglichkeiten, das Ganze auszuprogrammieren, aber unter Verwendung der gerade vorgestellten Kurzschreibweise bietet sich folgende Variante an:

```
var ca;
var Zk = "AXAYAZ";
for (var i=0; i<Zk.length; i++){
    ca = Zk.charAt(i);
    ca = (ca == "A") ? "B" : ca;
    Zk = Zk.substring(0, i) + ca +
        Zk.substring(i+1, Zk.length);
}
```

Ausgeschrieben hätte das Beispiel folgende, etwas längere Form:

```
var Zk = "AXAYAZ";
for (var i=0; i<Zk.length; i++){
    ca = Zk.charAt(i);
    if (ca == "A")
        ca = "B";
    Zk = Zk.substring(0, i) + ca +
        Zk.substring(i+1, Zk.length);
}
```

2.4.2 switch-Anweisung

Die Aufgabenstellung klingt einfach: In einer Variablen steht die Nummer eines Monats. Das Programm soll den Namen des Monats ausgeben. Mit `if`-Anweisungen kann das Ganze wie folgt erledigt werden:

```
if (monat == 1)
    document.write("Januar");
if (monat == 2)
    document.write("Februar");
if (monat == 3)
    document.write("März");
if (monat == 4)
    document.write("April");
if (monat == 5)
    document.write("Mai");
if (monat == 6)
    document.write("Juni");
if (monat == 7)
    document.write("Juli");
if (monat == 8)
    document.write("August");
if (monat == 9)
    document.write("September");
if (monat == 10)
    document.write("Oktober");
if (monat == 11)
    document.write("November");
if (monat == 12)
    document.write("Dezember");
```

Dieses Beispiel hat wenigstens den Vorteil, dass Sie keine `else`-Zweige verwenden müssen,³ dann würde nämlich der obige Code noch viel unübersichtlicher werden. Die Browser der Version 4 kennen jedoch eine Spracherweiterung, mit der Sie den Code etwas übersichtlicher gestalten können. Sie hat folgende Syntax:

```
switch (Ausdruck){
    case Wert1:
        // Programmblock
```

³ Das ist gleichzeitig auch ein Nachteil – bei obigem Code werden für jeden Wert von `monat` zwölf Überprüfungen durchgeführt.

```

        break;
    case Wert2:
        // Programmblock
        break;
    //usw.
    default:
        // Programmblock
}

```

Es dreht sich hierbei alles um `Ausdruck`, der in der Regel eine Variable ist. Hat dieser Ausdruck den Wert `Wert1`, wird der erste Programmblock ausgeführt, bei `Wert2` der zweite Programmblock und so weiter. Der `default`-Abschnitt wird ausgeführt, wenn keiner der vorherigen Werte zutrifft. Dieser Abschnitt ist optional. Jeder Programmblock muss mit dem Kommando `break` abgeschlossen werden.

Das Beispiel mit den Monaten lässt sich wie folgt umformulieren, und der Code ist deutlich kürzer und übersichtlicher:

```

switch (monat){
    case 1: var m = "Januar"; break;
    case 2: var m = "Februar"; break;
    case 3: var m = "März"; break;
    case 4: var m = "April"; break;
    case 5: var m = "Mai"; break;
    case 6: var m = "Juni"; break;
    case 7: var m = "Juli"; break;
    case 8: var m = "August"; break;
    case 9: var m = "September"; break;
    case 10: var m = "Oktober"; break;
    case 11: var m = "November"; break;
    case 12: var m = "Dezember"; break;
    default: var m = "Unbekannter Monat";
}
document.write(m);

```

2.5 Datenspeicherung

Wenn man viele Variablen braucht oder nicht genau weiß, wie viele Variablen man benötigt, ist das Konzept der Standardvariablen nicht flexibel genug. Sie sollten sich beispielsweise bei der Benennung der Variablen schon vorher ein Konzept zurechtlegen, um effizient auf die Variablen

zugreifen zu können. In diesem Abschnitt werden zwei Möglichkeiten vorgestellt, mehrere Variablen zu speichern.

2.5.1 Die eval-Methode

Wieder einmal sollen die Monatsnamen gespeichert werden. Anhand der Nummer des Monats soll auf den Monatsnamen zugegriffen werden. Der Variablenname setzt sich aus dem Wort »Monat« und der Zahl zusammen. Die Variablendeklaration sieht also wie folgt aus:

```
var Monat1 = "Januar";
var Monat2 = "Februar";
var Monat3 = "März";
var Monat4 = "April";
var Monat5 = "Mai";
var Monat6 = "Juni";
var Monat7 = "Juli";
var Monat8 = "August";
var Monat9 = "September";
var Monat10 = "Oktober";
var Monat11 = "November";
var Monat12 = "Dezember";
```

Die Frage lautet nun: Wenn (beispielsweise in einer Variablen) die Monatsnummer vorliegt, wie erhält man dann daraus den Monatsnamen? Natürlich können Sie eine Reihe von `if`-Anweisungen verwenden oder die `switch`-Anweisung, die jedoch nur ab der Browser-Version 4 (Netscape Navigator bzw. Internet Explorer) unterstützt wird. Schön wäre es jedoch, wenn Sie direkt den Variablennamen verwenden könnten. Folgender Code funktioniert natürlich nicht wie beabsichtigt:

```
document.write("Monat" + monat);
```

In `monat` steht die Monatsnummer, aber obige Zeile würde die Zeichenkette "Monat" konkateniert mit der Monatsnummer ausgeben, also den Namen der Variablen, nicht die Variable selbst. JavaScript bietet eine besondere Funktion, die es ermöglicht, Code auszuführen, der als Zeichenkette vorliegt. Diese Funktion heißt `eval`. Als Parameter wird eine Zeichenkette übergeben und `eval` führt den JavaScript-Code aus, der in der Zeichenkette steht. Der folgende Aufruf gibt den Wert der Variablen zurück, nicht ihren Namen:

```
eval("Monat" + monat);
```

Wollen Sie also den Monatsnamen auf dem Bildschirm ausgeben, können Sie folgende Zeile verwenden:

```
document.write(eval("Monat" + monat));
```

Alternativ dazu kann der `document.write`-Befehl auch innerhalb der `eval`-Klammern geschrieben werden:

```
eval("document.write(Monat" + monat + ")");
```

Steht in der Variablen `monat` beispielsweise der Wert 12, so würde der `eval`-Befehl nach dem Einsetzen der Variablen wie folgt aussehen:

```
eval("document.write(Monat12)");
```

Wenn in der Anweisung, die als Parameter an `eval()` übergeben wird, Anführungszeichen vorkommen, müssen Sie die herkömmlichen Regeln beachten, also entweder andere Anführungszeichen verwenden oder Anführungszeichen mit einem Backslash entwerthen. Oft ist es jedoch sinnvoll, den Aufruf von `eval()` möglichst weit innen zu platzieren.



```
document.write("Der Monatsname lautet " +  
    eval("Monat" + monat));
```

2.5.2 Arrays

Die obige Vorgehensweise funktioniert zwar tadellos, ist aber kompliziert, und auch hier werden die Befehle schnell unübersichtlich. In der Praxis werden zumeist so genannte Arrays verwendet. Das sind Variablencontainer, die mehrere Variablen beinhalten können. Auf eine einzelne Variable im Container greift man über den Variablennamen und eine Nummer zu. Die Nummer nennt man auch **Index**. Standardmäßig wird ein Array in JavaScript wie folgt definiert:

```
var a = new Array();  
a[1] = "Januar";  
a[2] = "Februar";  
//usw.
```

Mit der ersten Zeile, `var a = new Array()`, wird eine Arrayvariable deklariert. Sie können auch einen Parameter übergeben. Kleiner Hinweis am Rande: Wenn Sie mehr als einen Parameter übergeben, können Sie das Array vorbelegen:

```
var a = new Array("Januar", "Februar", "März", "April",  
    "Mai", "Juni", "Juli", "August")
```

```
"September", "Oktober", "November",  
"Dezember");
```

Der Index, über den auf ein Array-Element zugegriffen wird, steht in eckigen Klammern. JavaScript-Arrays beginnen, wie in vielen anderen Programmiersprachen und in Java übrigens auch, mit 0. Mit obigem Aufruf würde also `a[2]` auf März gesetzt. Sie müssen die Anweisung wie folgt modifizieren, damit `a[3]` den Wert »März« enthält:

```
var a = new Array("", "Januar", "Februar", "März", "April",  
"Mai", "Juni", "Juli", "August"  
"September", "Oktober", "November",  
"Dezember");
```

Die Anzahl der Elemente in einem Array können Sie über `length` bestimmen; hängen Sie dies, durch einen Punkt getrennt, an den Variablennamen für das Array an. Im obigen Beispiel würde `a.length` also 13 liefern, das Array enthält dreizehn Elemente.

Damit ist es Ihnen beispielsweise möglich, per Schleife alle Elemente eines Array auszugeben:

```
var a = new Array("", "Januar", "Februar", "März", "April",  
"Mai", "Juni", "Juli", "August"  
"September", "Oktober", "November",  
"Dezember");  
for (var i=0; i<a.length; i++)  
    document.write(a[i] + "<br />");
```

Beachten Sie, dass die Schleifenbedingung `i<a.length` lautet; die Schleife läuft also von 0 bis `a.length - 1`. Der Grund liegt wieder einmal in der Tatsache, dass das erste Element eines Arrays den Index 0 hat. Wenn also ein Array beispielsweise 13 Elemente hat, laufen die Indizes von 0 bis 12. Sie müssen die Arraylänge nicht explizit setzen, das macht JavaScript automatisch.



Im ActionScript-Teil lernen Sie noch weitere Möglichkeiten zur Array-Steuerung kennen.

2.6 Funktionen

Wenn Sie eine Anweisung oder gar einen Block von Anweisungen häufig einsetzen, kann das zu viel Tipparbeit führen. Hier können Sie sich mit **Funktionen** behelfen. Unter einer Funktion versteht man einen Programmblock, der nicht sofort ausgeführt wird, aber explizit auf- bzw. abgerufen werden kann. So eine Funktion führt entweder ein paar Befehle aus (dann nennt man sie mitunter auch Prozedur), oder es wird ein Wert zurückgegeben. Eine Prozedur ist mit `document.write()` vergleichbar, während eine Funktion beispielsweise `Zeichenkette.length` entspricht.

Eine Funktion hat folgende Syntax:

```
function Funktionsname(Parameter1, Parameter2){  
    //Programmblock  
    return Wert  
}
```

Die Anzahl der Parameter ist beliebig. Es kann auch kein Parameter verwendet werden (dann ist die Klammer leer). Innerhalb der Funktion können die Parameter dann unter dem Namen angesprochen werden, unter dem sie im Funktionskopf eingeführt wurden.

Die Zeile mit `return` ist optional. Der Befehl `return` sorgt dafür, dass die Funktion sofort verlassen wird (wie `break`). Wird hinter dem `return` ein Wert angegeben, so ist dieser der Rückgabewert der Funktion.

Im Gegensatz zu manchen anderen Programmiersprachen muss die Funktion nicht vor ihrem ersten Aufruf deklariert werden. Erkennt der JavaScript-Interpreter, dass ein Funktionsaufruf vorliegt, so wird das Dokument nach einer Funktion dieses Namens durchsucht.

Diese theoretische Einleitung soll nun anhand von praktischen Beispielen verdeutlicht werden. Zunächst schreiben wir eine Funktion, die eine Abkürzung für `document.write()` darstellt.

```
function dw(s){  
    document.write(s);  
}  
//Test-Aufruf  
dw("ActionScript mit JavaScript");
```

Dieses Beispiel ist freilich arg simpel; Sie können aber auf diese Art und Weise umfangreicheren Code – beispielsweise die Ausgabe eines auf

jeder Seite wiederkehrenden Elements wie einer Fußzeile mit Copyright-Informationen – in Funktionen auslagern.

Richtig sinnvoll wird eine Funktion allerdings oft erst, wenn der `return`-Befehl verwendet wird. Die Funktion gibt also nichts direkt aus, sondern liefert einen Wert zurück, der dann im eigentlichen Programm weiterverwendet werden kann.

Als Beispiel verwenden wir hier eine Funktion, welche die Fakultät einer Zahl zurückliefert⁴.

```
function fak(n){
    var produkt = 1;
    for (var i=1; i<=n; i++)
        produkt *= i;
    return produkt;
}
document.write("10! = " + fak(10));
```

Es sind noch zwei Fälle zu beachten:

- Es werden weniger Parameter übergeben, als im Funktionsrumpf angegeben sind.
- Es werden mehr Parameter übergeben, als im Funktionsrumpf angegeben sind.

Der erste Fall lässt sich recht schnell abhandeln. In diesem Fall hat der entsprechende Parameter den Wert `null`.

Der zweite Fall liegt nicht ganz so einfach. Die gute Nachricht vorweg: Alle Parameter, die an eine Funktion übergeben werden, sind in einem Array gespeichert. Sie werden jedoch kaum intuitiv erkennen, wie auf dieses Array zugegriffen werden kann. Der entsprechende Bezeichner heißt `Funktionsname.arguments` – und hier sehen Sie gleich, dass man als JavaScript-Funktionsnamen keine reservierten Begriffe (etwa `document`, `write` etc.) verwenden darf.

Die folgende Funktion gibt gleich eine ganze Liste von Fakultäten aus (durch Kommata voneinander getrennt). In einer `for`-Schleife wird das Array `arguments` durchlaufen und alle Werte werden mit spitzen Klammern ausgegeben. Wie bereits oben angemerkt wurde, kann mit `Array-name.length` die Anzahl der Elemente des Arrays bestimmt werden, in obigem Fall also `Funktionsname.arguments.length`!

⁴ Die Fakultät der Zahl n , geschrieben $n!$, ist das Produkt aller Zahlen von 1 bis n .

```
function fak(){
    var produkt;
    var returnstring = "";
    for (var i=0; i<fak.arguments.length; i++) {
        produkt = 1;
        for (var j=1; j<=fak.arguments[i]; j++)
            produkt *= j;
        returnstring += ", " + produkt;
    }
    returnstring = returnstring.substring(2);
    // führendes ", " entfernen
    return returnstring;
}
```

Der Aufruf

`fak(4, 6, 8)`

hätte also folgende Zeichenkette zum Ergebnis:

"24, 720, 40320"

Auf der CD-ROM zum Buch finden Sie eine lauffähige Variante dieses Beispiels unter dem Dateinamen **fak.html**. Die Ausgabe dieses Skripts sehen Sie in Abbildung 2.1.



Abbildung 2.1 Die Funktion wird mit mehreren Parametern aufgerufen.

Natürlich gibt es weitere sinnvollere Einsatzgebiete für eine Funktion außer der Bereitstellung von mathematischen Zusatzfunktionen. Kommen wir noch einmal zu den Schleifen zurück. Es war ja immer etwas

umständlich, ein Zeichen durch ein anderes auszutauschen. Mit einer Funktion können Sie das Ganze etwas übersichtlicher gestalten:

```
function ersetzeZeichen(Zeichenkette, Position, Neu){
    //Zeichenkette ist die betrachtete Zeichenkette
    //Position ist die Pos. des Zeichens, das ersetzt wird
    //Neu ist das neue Zeichen
    var neueZ = Zeichenkette.substring(0, Position);
    neueZ += Neu;
    neueZ += Zeichenkette.substring(Position+1,
        Zeichenkette.length);
    return neueZ;
}
```

Bei der Wahl der Variablenamen müssen Sie Vorsicht walten lassen. Existiert die Variable auch außerhalb der Funktion, so wird keine neue Variable erzeugt, sondern auf die alte Variable zugegriffen. Umgekehrt können Sie auf Variablen, die nur innerhalb einer Funktion auftreten, nicht von außen zugreifen. Man unterscheidet hier in **globale** (außerhalb und innerhalb einer Funktion gültige) und **lokale** (nur innerhalb von Funktionen gültige) Variablen. Wollen Sie innerhalb einer Funktion eine lokale Variable erzeugen, deren Namen schon eine globale Variable trägt, so müssen Sie der ersten Verwendung ein `var` voranstellen.

Folgendes Beispiel verdeutlicht das eben Beschriebene:

```
var a = 0;
var b = 0;
function test(){
    a++; //Zugriff auf globale Variable a
    var b=1; // Neue lokale Variable; hat mit der
        // globalen Variablen b nichts zu tun
    var c=0; //Neue lokale Variable
}
```

2.7 Objekte

Objektbasiert JavaScript gehört zu den so genannten objektorientierten Programmiersprachen (oder, um genauer zu sein, zu den objektbasierten Sprachen). Das Konzept der objektorientierten Programmierung (OOP) wird im Folgenden sehr stark vereinfacht erklärt. Der interessierte oder vorgebildete Leser möge diese Vereinfachung verzeihen, aber für das Verständnis von

JavaScript ist nur ein gewisser Überblick über das Konzept der Objekt-orientierung in JavaScript nötig.

In JavaScript ist alles, worauf man zugreift (außer Variablen), ein Objekt. Ein Objekt ist der Versuch, die reale Welt in einer Programmiersprachen-umgebung abzubilden. Ein Standardbeispiel für ein Objekt ist etwa ein Auto. Das Auto an sich (als abstrakter Begriff) kann als Objekt angesehen werden, ein einzelnes Auto wird als **Instanz** des Objekts Auto bezeichnet. Sie haben so etwas in diesem Kapitel schon einmal gesehen – `Array` ist so ein Objekt, und mit `new Array()` wird eine Instanz des `Array`-Objekts, also ein konkretes `Array`, erzeugt.

Ein Auto, oder ein Objekt im Allgemeinen, wird durch gewisse Parameter spezifiziert. Bei diesen unterscheidet man Methoden und Eigenschaften.

- Eine **Eigenschaft** kann als Variable angesehen werden, also als ein Wert, der fest mit dem Objekt verbunden ist und gelesen und geschrieben (gesetzt) werden kann. Bei einem Auto ist das beispielsweise die aktuelle Geschwindigkeit oder die aktuelle Menge Benzin im Tank. Eine Eigenschaft kann mit einem Funktionsaufruf verglichen werden, der immer einen Wert zurückgibt.
- Eine **Methode** ist eine Funktion, die ebenfalls fest mit dem Objekt verbunden ist. Im Gegensatz zur Eigenschaft wird hier aber nicht immer ein Wert zurückgegeben. Um auf das Beispiel des Autos zurückzukommen: Eine mögliche Methode wäre eine Methode `vollbremsung()`, die die (Eigenschaft) `Geschwindigkeit` abrupt auf 0 setzt. Eine andere Möglichkeit wäre eine Methode `beschleunigen()`; je nach Definition könnte man als Parameter angeben, auf welche Geschwindigkeit oder um wie viele Stundenkilometer die Geschwindigkeit erhöht werden soll. An der Schreibweise sieht man schon eine verwendete Konvention: Methoden werden immer durch nachgestellte Klammern gekennzeichnet; dies soll verdeutlichen, dass es sich hier nicht um eine Eigenschaft, sondern um einen »echten« Funktionsaufruf handelt.

Wie Sie vielleicht schon bei den vorhergehenden Beispielen in diesem Kapitel gesehen haben, ruft man eine Eigenschaft oder eine Methode auf, indem man an den Namen des entsprechenden Objekts den Namen der Methode oder Eigenschaft durch einen Punkt getrennt anhängt. Methoden werden dabei immer mit Klammern geschrieben; sollten keine Parameter übergeben werden, so werden leere Klammern verwendet. Bei Eigenschaften werden keine Parameter übergeben. Um auf die vorherge-

henden Beispiele zurückzukommen: `document.write()` ist eine Methode des `document`-Objekts, `Arrayname.length` ist eine Eigenschaft des Array-Objekts.

In den folgenden Kapiteln werden Sie die wichtigsten JavaScript-Objekte kennen lernen. Einige davon, wie beispielsweise das Array-Objekt, erleichtern die Programmierung, haben aber mit Webseiten nichts zu tun. Man kann von diesen Objekten Instanzen erzeugen und diese dann verwenden. Andere Objekte wiederum, wie beispielsweise die Objekte `window`, `document` oder `location`, haben direkt etwas mit der entsprechenden Webseite zu tun und enthalten beispielsweise Informationen über die URL der aktuellen Seite oder die Anzahl der Links im HTML-Dokument. In den folgenden Kapiteln werden diese Objekte in praxisnahen Beispielen vorgeführt.

3 Wichtige Objekte

Objektivität: Alles hat zwei Seiten. Aber erst wenn man erkennt, dass es drei sind, erfasst man die Sache.
– Heimito von Doderer

Am Ende des vorhergehenden Kapitels haben wir Ihnen bereits kurz erläutert, was ein Objekt überhaupt ist. Im »Lieferumfang« von JavaScript ist eine Reihe von Objekten enthalten. In diesem Kapitel werden wir die wichtigsten dieser Objekte kurz vorstellen und im Einsatz zeigen.

3.1 Das Objekt Date

Beim `Date`-Objekt handelt es sich um ein Objekt, von dem man eine Instanz erzeugen kann, und zwar – analog zu Arrays – wie folgt:

```
var d = new Date();
```

Die Variable `d` ist eine Datumsvariable und enthält das aktuelle Datum (inklusive Uhrzeit) auf dem lokalen Rechner (nicht auf dem Server), und zwar das Datum zu dem Zeitpunkt, an dem die Instanz erzeugt worden ist. Wenn also nach der Generierung der Variablen `d` eine lange Schleife folgt, die mehrere Sekunden dauert, ändert `d` seinen Wert nicht.

Bei der Erzeugung der Instanz können Sie die Datumsvariable auch direkt mit (anderen) Werten belegen. Dazu gibt es zwei Möglichkeiten:

```
var d = new Date(Jahr, Monat, Tag);
```

bzw.

```
var d = new Date(Jahr, Monat, Tag, Stunde, Minute, Sekunde);
```

Bitte beachten Sie die unten stehenden Hinweise, insbesondere die für den Monatswert!



Sie können mit geeigneten Methoden den Wert der Variablen `d` ändern, beispielsweise `Tag`, `Monat` und `Jahr`. Natürlich hat das überhaupt keine Auswirkungen auf die Systemzeit des Clients oder des Servers, das wäre ja aus Sicherheitsgründen völliger Wahnsinn, sondern es ändert sich lediglich der Wert der Variablen.

Zu den Standardanwendungen des `Date`-Objekts gehören die Anzeige des aktuellen Datums sowie die Berechnung des Zeitunterschieds zwischen

zwei Daten. Im Folgenden sollen diese beiden Möglichkeiten anhand von Beispielen erläutert werden.

3.1.1 Tagesdatum

Haben Sie erst einmal eine Instanz des `Date`-Objekts erzeugt, können Sie das genaue Datum mit entsprechenden Methoden abfragen. Wie unter UNIX generell üblich, speicherte JavaScript 1.0 jedes Datum als die Anzahl der Millisekunden seit dem 1. Januar 1970; in neueren Versionen ist das nicht mehr so. Da es ein wenig aufwändig wäre, aus diesem Wert die interessanteren Daten wie etwa die Jahreszahl zu berechnen, gibt es hierfür vorgefertigte Methoden. Der folgenden Tabelle können Sie die wichtigsten dieser Methoden entnehmen.

Methode	Beschreibung
<code>getDate()</code>	Aktueller Tag der Datumsvariablen (1 ... 31)
<code>getMonth()</code>	Aktueller Monat der Datumsvariablen minus eins (0 ... 11)
<code>getFullYear()</code>	Aktuelles Jahr der Datumsvariablen; für Wertebereich siehe Abschnitt »Y2K«
<code>getHours()</code>	Aktuelle Stunde der Datumsvariablen (0 ... 23)
<code>getMinutes()</code>	Aktuelle Minute der Datumsvariablen (0 ... 59)
<code>getSeconds()</code>	Aktuelle Sekunde der Datumsvariablen (0 ... 59)
<code>getMilliseconds()</code>	Aktuelle Millisekunden der Datumsvariablen (0 ... 999)

Tabelle 3.1 Die wichtigsten Methoden des `Date`-Objekts

Hiermit ist es recht einfach, das aktuelle Datum auszugeben. Will man auf Monatsnamen nicht verzichten, muss man diese wieder in einem Array speichern. Im folgenden Listing – Sie finden es auf der CD-ROM unter **tagesdatum.html** – wurde dies umgesetzt.

```
<html>
<head>
<title>Tagesdatum</title>
<script type="text/javascript"><!--
var monatsnamen = new Array(
    "Januar", "Februar", "März", "April",
    "Mai", "Juni", "Juli", "August",
    "September", "Oktober", "November", "Dezember");
```

```

function tagesdatum(){
    var d = new Date();
    var datumsstring = d.getDate();
    datumsstring += ". " + monatsnamen[d.getMonth()];
    datumsstring += " " + d.getYear();
    document.write(datumsstring);
}
//--></script>
</head>
<body>
<h1>Tagesdatum</h1>
Guten Tag, heute ist der
    <script type="text/javascript"><!--
        tagesdatum();
    //--></script>
!
</body>
</html>

```

Listing 3.1 Tagesdatum mit JavaScript



Abbildung 3.1 Das aktuelle Tagesdatum wird ausgegeben.

3.1.2 Y2K

Auch JavaScript ist vom ominösen Jahr-2000(Y2K)-Problem betroffen, allerdings ist das hier nicht so schlimm wie damals in weiten Teilen der Wirtschaft. Die Crux liegt in der Methode `getYear()`. Je nach Browser werden manche Jahre anders interpretiert. Tabelle 3.2 zeigt auf, welche

Rückgabewerte die verschiedenen Browser bei den jeweiligen Zeiträumen haben. Dabei bedeutet »volle Jahreszahl« eine vierstellige Jahreszahl (außer natürlich bei Jahreszahlen vor 1000); »kurze Jahreszahl« bedeutet eine zweistellige Jahreszahl im 20. Jahrhundert, also etwa »99« statt »1999«.

Zeitraum	JS 1.0	JS 1.1/1.2	JS 1.3 und höher
<1970	Nicht unterstützt	Volle Jahreszahl	Jahreszahl minus 1900 (1800 wird durch -100 dargestellt)
1970 ... 1999	Jahreszahl minus 1900	Kurze Jahreszahl	Jahreszahl minus 1900
≥2000	Jahreszahl minus 1900	Volle Jahreszahl	Jahreszahl minus 1900

Tabelle 3.2 Die Jahreszahlbehandlung der verschiedenen JavaScript-Versionen

Tabelle 3.2 wirkt erschreckend, aber in der Praxis gibt es einen relativ einfachen Ausweg. Üblicherweise liegen alle verwendeten Daten im Bereich von 1970 bis ins 21. Jahrhundert hinein. Eine Jahreszahl kleiner als 200 ist also um 1900 zu niedrig; andere Jahreszahlen sind – nehmen wir einfach einmal an – korrekt. Man kann also eine Funktion schreiben, die das Y2K-sichere Jahr zurückgibt:

```
function getYearY2K(d) {
    var y = d.getFullYear();
    if (y < 200)
        return y + 1900;
    else
        return y;
}
```

Ab JavaScript 1.3 (also insbesondere ab dem Netscape Navigator 4.06) gibt es eine Methode `getFullYear()`, die grundsätzlich kein verkürztes Datum zurückgibt, sondern stets die volle Jahreszahl. Alle Browser, die ab dem heutigen Zeitpunkt erscheinen, sollten diese Methode unterstützen. Mittlerweile ist wohl die Verbreitung dieser Browser so groß, dass Sie den »zurückgebliebenen Rest« vernachlässigen können.

3.1.3 Mit Daten rechnen

In der JavaScript-Version 1.0 wurden Datumswerte intern als Zahl der seit dem 1. Januar 1970 verstrichenen Millisekunden gespeichert. Diese auf den ersten Blick merkwürdig anmutende Vorgehensweise bietet einen großen Vorteil: Rechenoperationen werden dadurch ermöglicht, die ansonsten nur sehr schwer und kompliziert zu implementieren gewesen wären.

Die Hauptanwendung ist hier die Berechnung der Differenz zweier Datumswerte. Wie viele Tage oder Stunden dauert es noch bis zum 1.1. 2001? Wie viele Tage bin ich alt? Um wie viele Stunden ist mein Bruder/meine Schwester/mein Hund älter/jünger als ich?

Sie nutzen dazu die Methode `getTime()` und ganz wenig Mathematik. Die Methode `getTime()` gibt die Zeit in der Datumsvariablen im Format »Millisekunden seit 1.1. 1970« zurück. Wenn zwei Datumsvariablen vorliegen, deren zeitliche Differenz berechnet werden muss, kann der Millisekundenwert beider Variablen voneinander abgezogen werden, und man hat das Ergebnis. Dieses liegt dann zwar in Millisekunden vor, aber die Umrechnung in handlichere Werte ist recht einfach. Eine Division durch 1000 ergibt die Sekunden (eine Sekunde dauert 1000 Millisekunden); die Division durch $1000 * 60$ ergibt die Minuten (60 Sekunden sind eine Minute) und so fort. Die Division durch $1000 * 60 * 60 * 24$ ergibt den Zeitunterschied in Tagen.

Noch wissen Sie nicht, wie man Benutzereingaben mit JavaScript abfängt; aus diesem Grund muss im folgenden Beispiel zum Testen an einer Variablen »herumgeschraubt« werden. Das Beispiel mit den Geburtsdaten ist heikel, wenn man vor dem 1. Januar 1970 geboren ist (aus Gründen der Abwärtskompatibilität), also wird nun die zeitliche Differenz des aktuellen, lokalen Datums zum nächsten Neujahr, also (zumindest zum Zeitpunkt der Drucklegung) dem 1.1. 2003, berechnet:

```
<html>
<head>
<title>Differenz</title>
<script type="text/javascript"><!--
var d = new Date(89, 9, 3); // 3.10. 1989
var d2 = new Date(2003, 0, 1); // 1.1. 2003

function differenz(){
    var dtime = d.getTime();
```

```

var d2time = d2.getTime();
var diff = d2time - dtime;
var diffstring = (diff / 1000) + " Sekunden";
diffstring += " oder " + (diff/(1000*60)) + " Minuten";
diffstring += " oder " + (diff/(1000*60*60))
                + " Stunden";
diffstring += " oder " + (diff/(1000*60*60*24))
                + " Tage";

document.write("Zeitunterschied: " + diffstring);
}
//--></script>
</head>
<body>
<h1>Zeitunterschied</h1>
<script type="text/javascript"><!--
    differenz();
//--></script>
</body>
</html>

```

Listing 3.2 Die Differenz zwischen zwei Datumswerten wird berechnet.

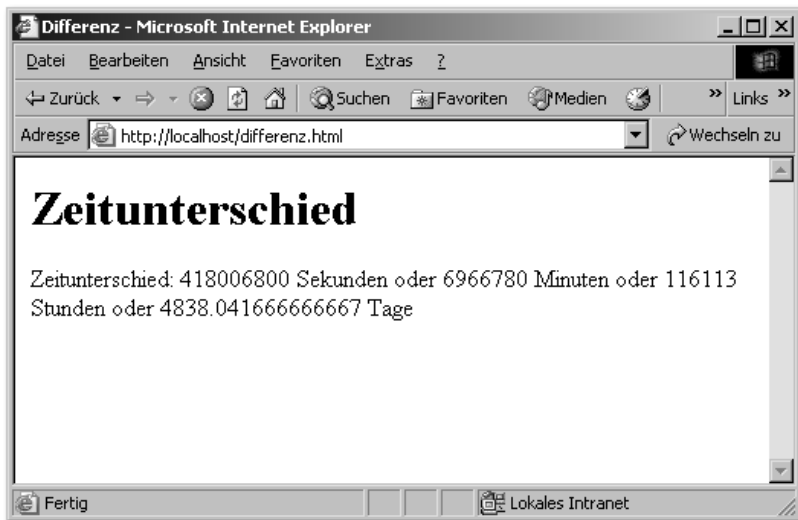


Abbildung 3.2 Die Differenz wird berechnet und ausgegeben.

Dieses Skript finden Sie auf der CD-ROM zum Buch unter dem Dateinamen **differenz.html**.



3.1.4 Lokales Datumsformat

Mit der Methode `toLocaleString()` können Sie einen Datumswert an die lokalen Einstellungen (das heißt an die Sprachversion des Betriebssystems und an lokale Spracheinstellungen) anpassen. So recht will das aber erst mit dem Internet Explorer 5.5 gelingen, der die entsprechenden Informationen direkt aus den Daten in der Systemsteuerung holt (unter Windows, bei anderen Betriebssystemen stehen diese Informationen an anderer Stelle). In der nachfolgenden Tabelle sehen Sie, wie drei Browserversionen das aktuelle Datum nach Anwendung von `toLocaleString()` ausgeben – bei identischen Einstellungen in der Systemsteuerung.

Browser	Ausgabe (am 31. Dezember 2000, 12 Uhr mittags)
Netscape Navigator 4.7x	Sunday, December 31, 2000 12:00:00
Internet Explorer 5.0x	12/31/2000 12:00:00
Internet Explorer 5.5/6	Sonntag, 31. Dezember 2000 12:00:00

Tabelle 3.3 Die Ergebnisse von `toLocaleString()`

3.2 Das Objekt Math

Mit diesem Objekt, von dem man im Übrigen keine Instanz erzeugen muss, können mathematische Berechnungen ausgeführt werden. Dabei sprechen wir nicht von elementaren Funktionen wie etwa den vier Grundrechenarten (die, wie bereits bekannt, durch entsprechende Operatoren abgedeckt sind), sondern von den trigonometrischen Funktionen (Sinus, Kosinus etc.) oder den Exponentialfunktionen (x »hoch« y, Logarithmus etc.). In der Praxis kommen diese Funktionen jedoch eher selten vor. Viel interessanter ist da schon die Generierung von Zufallszahlen.

Das `Math`-Objekt hat (neben vielen anderen) auch die Methode `random()`, die eine Zufallszahl zwischen 0 und 1 zurückliefert:

```
var z = Math.random();
```

Natürlich verwendet der Netscape Navigator einen Berechnungsalgorithmus für die Erzeugung der »Zufallszahl«, und auch dieser Algorithmus ist von der aktuellen Zeit abhängig. Auf den ersten Blick ist diese Zahl jedoch nicht besonders nützlich, da beispielsweise in obiger Anwendung eine

Zahl zwischen null und fünf benötigt wird. Mit einem kleinen Trick können Sie sich aber hier behelfen. Wird nämlich die Zufallszahl mit sechs multipliziert und dann abgerundet, so erhält man eine ganze Zahl zwischen null und fünf:

```
var z = Math.floor(6 * Math.random());
```



Mit `Math.floor()` wird eine Zahl abgerundet. Diese Zahl wird als Parameter übergeben.

Allgemein kann man das auch in einer Funktion schreiben. Die Funktion `random2(n)` erzeugt eine ganze Zahl zwischen 0 und n:

```
function random2(n){
    return Math.floor((n + 1) * Math.random());
}
```

Hiermit können Sie ein Beispiel erstellen, das aus einer Liste von Links zufällig einen auswählt und ausgibt:

```
<html>
<head>
<title>Zufallslink</title>
</head>
<body>
<h1>Zufallslink</h1>
<script type="text/javascript"><!--
var urls = new Array();
urls[0] = "http://www.galileo-press.de";
urls[1] = "http://www.spiegel.de";
urls[2] = "http://www.focus.de";

// ab hier: nichts mehr verändern! -Der Webmaster
var zufallsurl = urls[random2(3)];
document.write("<a href='" + zufallsurl + "'>");
document.write(zufallsurl);
document.write("</a>");

// Hilfsfunktion
function random2(n){
    return Math.floor((n + 1) * Math.random());
}
--></script>
```

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

Listing 3.3 Ein zufälliger Link wird ausgegeben.

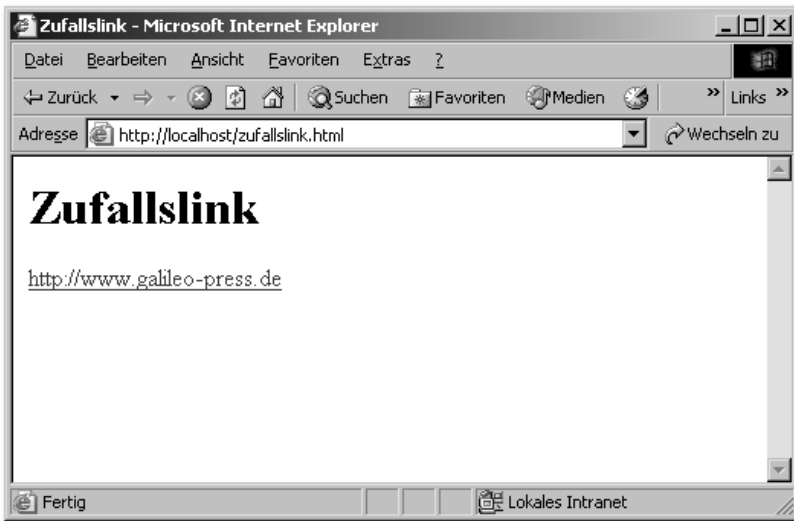


Abbildung 3.3 Ein Link wird zufällig ausgewählt und ausgegeben.

Dieses Skript hat auf der CD-ROM zum Buch den Dateinamen **zufalls-link.html**.



3.3 Browser-Erkennung

In diesem Abschnitt wird eine spezielle Lösung für das Problem der Inkompatibilität der einzelnen Browser zu- und untereinander aufgezeigt. Sie können nämlich versuchen, den Browser-Typ und die Browser-Version des Benutzers zu erkennen und gewisse Browser auf spezielle Seiten, die für diesen Browser »optimiert« (ein schreckliches Wort) worden sind, umzuleiten. Wenn Sie also auf Ihren Seiten die neuesten Effekte verwenden, sollten Sie auf Ihrer Homepage den Browser des Besuchers erkennen und nur Anwender mit neuen Browsern auf Ihre effektbeladene Seite weiterleiten; der Rest darf eine schlankere, kompatiblere Version der Seiten (die Sie hoffentlich erstellt haben!) betrachten.

Die Browser-Erkennung wurde mittlerweile von Netscape praktisch perfektioniert; der »Client Sniffer« ist ziemlich bekannt und erkennt fast alle Unterversionen der meisten bekannten JavaScript-fähigen Browser. In der Praxis braucht man das freilich nicht so genau. Oft können Sie abfragen,

ob ein Browser ein Objekt unterstützt (dazu in späteren Kapiteln mehr), oder es gibt eine Möglichkeit, die Funktionalität auch mit der Urversion, also JavaScript 1.0, nachzubilden (siehe beispielsweise Arrays). Ebenso können Sie mit den verschiedenen Werten für das `language`-Attribut des `<script>`-Tags operieren. Wenn Sie jedoch mehrere Seiten in Ihrer Anwendung haben, die erst ab JavaScript 1.1 funktionieren, und dann Benutzer älterer Browser schon auf der (JavaScript-1.0-kompatiblen) Homepage auf einen alternativen, schlankeren Bereich weiterleiten wollen, kann sich eine Browser-Erkennung lohnen. Star Division setzt so etwas beispielsweise ein – ohne jedoch eine schlankere Version der Seiten parat zu haben.

Für die Erkennung der Browser-Version gibt es ein eigenes Objekt, `navigator`. Man braucht davon keine Instanz zu erzeugen, sondern kann direkt darauf zugreifen. Das Objekt hat drei interessante Eigenschaften. Bei zwei Browsern, einem Netscape 4.78 unter Windows 98 und einem Internet Explorer 6 unter Windows 2000, haben diese Eigenschaften die in Tabelle 3.4 aufgeführten Werte.

Eigenschaft	Netscape 4.78 deutsch	Internet Explorer 6 deutsch
<code>navigator.appName</code>	Netscape	Microsoft Internet Explorer
<code>navigator.userAgent</code>	Mozilla/4.78 [de] (Win98; U)	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
<code>navigator.appVersion</code>	4.78 [de] (Win98; U)	4.0 (compatible; MSIE 5.01; Windows NT 5.0)

Tabelle 3.4 Werte des `navigator`-Objekts bei zwei exemplarischen Browsern

Sie sehen insbesondere, dass man **eigentlich** mit `navigator.appName` schon einmal zwischen Netscape und Internet Explorer unterscheiden kann. Doch leider ist bei einigen früheren Versionen des Internet Explorers als `navigator.appName` auch Netscape angegeben. Daher müssen Sie auf `navigator.userAgent` zurückgreifen. Diese Eigenschaft hat den Aufbau `Mozilla/Versionsnummer` und dahinter plattformspezifische Informationen. Bei der Versionsnummer ist der Internet Explorer so ehrlich und gibt (in vielen Versionen) zu, meistens in Bezug auf JavaScript eine Version hinter dem Netscape-Pendant zurückzuliegen. Der Internet Explorer 5 ist in Bezug auf JavaScript in etwa auf Höhe des Netscape Navigator 4, also fängt `navigator.userAgent` mit `Mozilla/4.0` an. Sie können also von der Zeichenkette hinter `Mozilla/` keine direkten Rückschlüsse auf die Browserversionsnummer ziehen!

Ausführliche Informationen zur Browser-Erkennung finden Sie unter http://devedge.netscape.com/docs/examples/javascript/browser_type.html. In diesem Abschnitt werden nur die Versionsnummern von Netscape Navigator und Internet Explorer bestimmt. Unter der angegebenen URL finden Sie noch detailliertere Informationen, auch für Browser wie etwa den Opera.

3.3.1 Wann ist es ein Netscape Navigator?

Wie Sie bereits gesehen haben, kann man mit `navigator.appName` nicht auf den Browser-Hersteller schließen, auch dann nicht, wenn `navigator.userAgent` mit `Mozilla` beginnt. Stattdessen muss man zusätzlich feststellen, ob man **nicht einen anderen** Browser hat. Andere Browser erkennt man an den folgenden Kriterien.

- ▶ `navigator.userAgent` enthält die Zeichenkette `compatible` (typisch für den Internet Explorer).
- ▶ `navigator.userAgent` enthält die Zeichenkette `Opera` (Opera-Browser).
- ▶ `navigator.userAgent` enthält die Zeichenkette `WebTV` (WebTV-Browser).

Da es zwischen Groß- und Kleinschreibung Unterschiede geben kann, empfiehlt es sich, vor den Überprüfungen `navigator.userAgent` in Kleinbuchstaben umwandeln. Hierzu verwendet man die Methode `Zeichenkette.toLowerCase()`.

```
var ua = navigator.userAgent.toLowerCase(),
var netscapebrowser = (ua.indexOf("mozilla")!=-1 &&
    ua.indexOf("compatible")==-1 && ua.indexOf("opera")==-1),
```

3.3.2 Wann ist es ein Internet Explorer?

Der Internet Explorer ist viel einfacher zu erkennen. Wie Sie bereits gesehen haben, schreibt der Internet Explorer in `navigator.userAgent` seine Versionsnummer in der folgenden Form: `MSIE Versionsnummer`. Es muss also einfach auf das Vorhandensein der Zeichenkette `MSIE` geprüft werden bzw. nach der Umwandlung in Kleinbuchstaben auf die Existenz der Zeichenkette `msie`:

```
var ua = navigator.userAgent.toLowerCase(),
var netscapebrowser = (ua.indexOf("mozilla")>-1 &&
    ua.indexOf("compatible")==-1 && ua.indexOf("opera")==-1),
var iebrowser = (ua.indexOf("msie") > -1),
```

3.3.3 Versionsnummer erkennen

Die Eigenschaft `navigator.appVersion` enthält die Zeichenkette nach Mozilla/ aus `navigator.userAgent`, zumindest für Netscape-Browser. Es gibt hier zwei Möglichkeiten, die Versionsnummer herauszubekommen. Die erste besteht darin, die Versionsnummer aus `navigator.appVersion` herauszuschneiden (die Versionsnummer endet vor dem ersten Leerzeichen) und dann mit `eval()`, `parseInt()` oder durch eine Multiplikation mit 1 in eine numerische Variable umzuwandeln:

```
var ua = navigator.userAgent.toLowerCase(),
    netscapebrowser = (ua.indexOf("mozilla")!=-1 &&
        ua.indexOf("compatible")==-1 && ua.indexOf("opera")==-1);
var iebrowser = (ua.indexOf("msie") > -1);
var av = navigator.appVersion;
var version = parseInt(av.substring(0, av.indexOf(" ")));
var version2 = parseInt(av.substring(0, av.indexOf(".")));
```

In der Variablen `version2` wird die Versionsnummer vor dem Punkt gespeichert, also im obigen Beispiel jeweils 4.



Der Netscape 6 meldet sich mit der Versionsnummer 5.0. Da es keine Version Netscape 5 mehr geben wird, ist diese Unterscheidung eindeutig.

Die Versionsnummer eines Netscape-Browsers wurde nun direkt ermittelt. Beim Internet Explorer müssen Sie etwas trickreicher vorgehen. Es wird an dieser Stelle darauf verzichtet, mühsam anhand einer umfangreichen Tabelle die Versionsunterschiede zwischen den Internet Explorer-Versionen herauszuarbeiten; nachfolgend kurz die Regeln, woran man die Versionen 3, 4 und 5.x erkennt:

- ▶ IE3: `version2` muss kleiner als 4 sein (ist 2 oder 3).
- ▶ IE4: `version2` muss 4 sein.
- ▶ IE5: `version2` ist ebenfalls 4, und `navigator.userAgent` enthält die Zeichenkette »MSIE 5.«).

In der folgenden, endgültigen Variante des Codes steht in der Variablen `versionsnummer` die Versionsnummer des Browsers, also eine ganze Zahl zwischen zwei und sechs.

```
var ua = navigator.userAgent.toLowerCase();
var netscapebrowser = (ua.indexOf("mozilla")!=-1 &&
    ua.indexOf("compatible")==-1 && ua.indexOf("opera")==-1);
var iebrowser = (ua.indexOf("msie") > -1);
var av = navigator.appVersion;
```

```

var version = parseInt(av.substring(0, av.indexOf(" ")));
var version2 = parseInt(av.substring(0, av.indexOf(".")));
if (netscapebrowser)
    var versionsnummer = version2;
else if (iebrowser){
    if (version2 < 4)
        versionsnummer = 3;
    if (version2 == 4 && ua.indexOf("msie 5") == -1)
        versionsnummer = 4;
    if (version2 == 4 && ua.indexOf("msie 5") > -1)
        versionsnummer = 5;
    if (version2 == 4 && ua.indexOf("msie 6") > -1)
        versionsnummer = 6;
}

```

Sie sehen hier, dass Sie den Code für den Internet Explorer beim Erscheinen einer neuen Version stets anpassen müssen. Zwar könnten Sie versuchen, eine Gesetzmäßigkeit zu erkennen, und beispielsweise davon ausgehen, dass der Internet Explorer 7 in `navigator.userAgent` die Zeichenkette `MSIE 7` enthält, aber zum einen hatte Microsoft immer einmal wieder eine Überraschung bei `navigator.userAgent` parat, und was spricht zum anderen gegen eine Version 6.5?

Eine kleine Anekdote am Rande: Auf manchen Seiten durchsuchte man `navigator.userAgent` nach der Zeichenkette `Mozilla/4.0`; damit sollten der Internet Explorer 4 und der Netscape Navigator 4 erkannt werden. Das ging ganz gut, auch bei den Unterversionen Netscape Navigator 4.01 bis 4.08. Die nächste Version hatte jedoch die Versionsnummer 4.5, und so kam es, dass eine gewisse Funktionalität ausgerechnet beim neuesten Browser auf dem Markt nicht funktionierte. Aus diesem Grund sollten Sie nie explizit auf eine Versionsnummer prüfen, sondern auf eine Mindestversionsnummer. Schreiben Sie also nicht

```

if (version2 == 4)
    // ...

```

sondern

```

if (version2 >= 4)
    // ...

```

3.4 Weiterleitung

Alle Angaben zur URL der aktuellen Seite sind im `location`-Objekt gespeichert, das ein Unterobjekt des `window`-Objekts ist. Das `window`-Objekt kennen Sie ja bereits ein wenig, beispielsweise von `window.alert()`. Man greift also auf das `location`-Objekt mit `window.location` zu.



Das `window`-Objekt ist jedoch das Standardobjekt unter JavaScript, der Name kann also weggelassen werden. Ein einfaches `location` (bzw. `alert()`) reicht zum Aufruf aus.



Das `window`-Objekt wird in Kapitel 4, Fenster, ausführlich behandelt.

Das `location`-Objekt hat mehrere Eigenschaften und Methoden. Zunächst ist für Sie die Eigenschaft `href` interessant, die die gesamte URL enthält. Ab dem Netscape Navigator 2 bzw. dem Internet Explorer 4 erhält man diese auch, wenn man direkt auf `location` zugreift, aber der Internet Explorer 3 fällt wieder einmal aus der Reihe. Aus diesem Grund sollten Sie immer `location.href` verwenden.

Das war auch schon alles, was wissenswert ist. Der folgende Code zeigt noch einmal exemplarisch auf, wie verschiedene Browser auf verschiedene Seiten umgeleitet werden. Achten Sie darauf, wie verschachtelte `if`-Anweisungen vermieden werden: Wenn `location.href` mit einem neuen Wert besetzt wird, wird die Ausführung des Skripts sofort beendet.

```
var ua = navigator.userAgent.toLowerCase();
var netscapebrowser = (ua.indexOf("mozilla")!=-1 &&
    ua.indexOf("compatible")==-1 && ua.indexOf("opera")==-1);
var iebrowser = (ua.indexOf("msie") > -1);
var av = navigator.appVersion;
var version = parseInt(av.substring(0, av.indexOf(" ")));
var version2 = parseInt(av.substring(0, av.indexOf(".")));
if (netscapebrowser)
    var versionsnummer = version2;
else if (iebrowser){
    if (version2 < 4)
        versionsnummer = 3;
    if (version2 == 4 && ua.indexOf("msie 5") == -1)
        versionsnummer = 4;
    if (version2 == 4 && ua.indexOf("msie 5") > -1)
        versionsnummer = 5;
    if (version2 == 4 && ua.indexOf("msie 6") > -1)
```

```

        versionsnummer = 6;
    }
    // Weiterleitung
    if (netscapebrowser && versionsnummer == 2)
        location.href="nn2.html";
    if (netscapebrowser && versionsnummer == 3)
        location.href = "nn3.html";
    if (netscapebrowser && versionsnummer == 4)
        location.href = "nn4.html";
    if (netscapebrowser && versionsnummer == 5)
        location.href = "n6.html";

    if (iebrowser && versionsnummer == 3)
        location.href = "ie3.html";
    if (iebrowser && versionsnummer == 4)
        location.href = "ie4.html";
    if (iebrowser && versionsnummer == 5)
        location.href = "ie5.html";
    if (iebrowser && versionsnummer == 6)
        location.href = "ie6.html";

```

Listing 3.4 Browserunterscheidung

Sie finden ein komplettes Beispiel auf der CD-ROM unter dem Dateinamen **browserunterscheidung.html**.



3.5 Event-Handler

In Kapitel 2, Spracheinführung, haben wir sie schon angesprochen, hier stellen wir noch kurz einige Event-Handler vor. Dies dient nur dazu, Ihnen einen groben Überblick zu geben.

- ▶ **onblur**: wenn ein Fenster oder Formularfeld den Fokus verliert (nicht mehr im Vordergrund ist bzw. den Cursor enthält),
- ▶ **onchange**: wenn der Wert eines Formularfelds geändert wird,
- ▶ **onclick**: beim Klicken auf einen Link oder auf ein Formularelement,
- ▶ **onfocus**: wenn ein Fenster oder Formularfeld den Fokus erhält (Mausklick),
- ▶ **onload**: beim Laden eines Dokuments,
- ▶ **onmouseover**: wenn mit der Maus über einen Link gefahren wird,

- ▶ `onmouseout`: wenn der Link mit der Maus wieder verlassen wird (nach `onmouseover`),
- ▶ `onunload`: beim Verlassen eines Dokuments.

Wie Event-Handler eingesetzt werden, ist ja auch bereits bekannt. In geeigneten HTML-Tags dienen die Event-Handler als Attributnamen. Der Wert des Attributs ist dann das JavaScript-Kommando, das beim Eintreten des Ereignisses ausgeführt werden soll.

4 Fenster

Wenn Sie einen Schweizer Bankier aus dem Fenster springen sehen, springen Sie hinterher. Es gibt bestimmt was zu verdienen.

– François Marie Voltaire

Das `window`-Objekt ist das wohl am häufigsten verwendete JavaScript-Objekt – und dasjenige, welches am seltensten ausgeschrieben wird. Die Methoden des Fenster-Objekts können nämlich aufgerufen werden, ohne dass man das Objekt voranstellt. Sie kennen ja bereits den Befehl `window.alert("ActionScript mit JavaScript")`; alternativ dazu reicht auch `alert("ActionScript mit JavaScript")`. Die Methode `document.write()` ist streng genommen auch eine Abkürzung. Das `document`-Objekt ist ein Unterobjekt des `window`-Objekts, ganz korrekt wäre also `window.document.write()`. Das `window`-Objekt bezeichnet ein Browser-Fenster oder einen Frame, denn jeder Frame wird intern als eigenes Fenster behandelt. Auf den aktuellen Frame beziehungsweise das aktuelle Browser-Fenster greift man mit `window` oder `self` oder `this` zu. Wie gesagt – den Sermon könnte man sich sparen, aber man macht seine Skripte so übersichtlicher und programmiert auch sauberer. Als Entwickler muss man immer fürchten, dass es eine neue Browser-Version oder einen ganz neuen Browser gibt, der die Syntax etwas strenger auslegt.

4.1 Modale Fenster

Unter einem modalen Fenster einer Applikation versteht man ein Fenster, das in den Vordergrund tritt, den Fokus bekommt und den Zugriff auf das Hauptapplikationsfenster sperrt: Solange das (modale) Fenster offen ist, kann auf das Hauptfenster nicht mehr zugegriffen werden. Ein Beispiel ist die Meldung einer Textverarbeitung, die den Benutzer vor dem Verlassen des Programms fragt, ob das aktuelle Dokument gespeichert werden soll (sofern es noch nicht gespeichert wurde). Sie müssen entweder auf **Ja**, auf **Nein** oder auf **Abbrechen** klicken, um wieder auf die Textverarbeitung zugreifen zu können.

4.1.1 Warnfenster

Eine Möglichkeit, ein modales Fenster mit JavaScript zu erstellen, haben Sie bereits kennen gelernt: `window.alert()` oder kurz `alert()`. Damit

wird ein modales Fenster mit einem Text angezeigt; als Icon wird ein Warndreieck verwendet (deswegen auch der Name: »alert«, dt. »Warnung«). Aus diesem Grund bieten wir hierzu nicht erneut ein Beispiel an, Sie haben dazu ja schon einiges gesehen.

4.1.2 Bestätigungsfenster

Bot das letzte modale Fenster noch sehr wenige Eingriffsmöglichkeiten – nur eine OK-Schaltfläche –, so können Sie mit `window.confirm()` ein modales Fenster mit zwei Schaltflächen anbieten: je nach Betriebssystem und Sprache **OK** und **Cancel** oder **OK** und **Abbrechen**. Klickt der Benutzer auf **OK**, so liefert die Methode `true` zurück, ansonsten `false`.

Auch hier macht Netscape wieder vor, wie man diese Methode sinnvoll einsetzen kann. Unter <http://www.mozilla.org/> können Sie den aktuellen Entwicklungsstand des Generation-6-Browsers von Netscape herunterladen. Zwar gibt es in unterschiedlichen Abständen so genannte **Milestones** (Meilensteine), die schon relativ stabil laufen, und auch erste Release Candidates für die Version 1.0 (die zu dem Zeitpunkt, an dem Sie dieses lesen, vermutlich schon erschienen sein wird), aber jede Nacht wird der aktuelle Stand der Dinge kompiliert und via FTP zur Verfügung gestellt. Es ist nicht sichergestellt, dass diese Version überhaupt startet. Aus diesem Grund wird mit JavaScript sicherheitshalber noch einmal nachgefragt, ob der Benutzer diese Version wirklich herunterladen will. Der zugehörige Code sieht dort ungefähr wie folgt aus:

```
<html>
<head>
<title>window.confirm()</title>
</head>
<body>
<h1>Laden Sie die Alpha-Version des neuen Netscape herunter!</h1>
<a href="datei.exe" onclick="return confirm('Are you sure you
want to do this crazy thing?');">Hier klicken</a>
</body>
</html>
```

Listing 4.1 Der Download muss bestätigt werden.



Der Dateiname dieses Skripts auf der CD-ROM zum Buch lautet **confirm.html**.

Hier sind ein paar zusätzliche Erklärungen notwendig:

- Der Befehl `confirm('Are you sure you want to do this crazy thing?')` öffnet ein modales Fenster. Klickt der Benutzer darin **OK** an, wird `true` zurückgeliefert, ansonsten `false`.
- Stimmt der Benutzer also dem Download zu, so lautet das `onClick`-Attribut des `<a>`-Elements `return true`, ansonsten `return false`.

Die Bedeutung ist relativ einfach: Im obigen Fall kollidieren der HTML-Link und der `onclick`-Event-Handler. Ein Anklicken des Links führt dazu, dass die Datei **datei.exe** vom Webserver angefordert wird; der `onclick`-Event-Handler führt den entsprechenden JavaScript-Code aus. Wer macht das Rennen?

Der Trick hierbei ist folgender: Der JavaScript-Code wird auf jeden Fall ausgeführt. Endet der Code mit `return true`, so wird daraufhin das Ziel des HTML-Links aufgerufen, bei `return false` wird der HTML-Link nicht aufgerufen. So funktioniert auch die Abfrage von oben: Verneint der Benutzer die Frage im modalen Fenster, so lautet der `onclick`-Event-Handler `return false`, und der Link wird nicht ausgeführt. Bejaht der Benutzer die Abfrage, so lautet der `onclick`-Event-Handler `return true` und **datei.exe** wird wie geplant aufgerufen.

4.1.3 Fenster mit Rückgabewert

Es gibt noch eine weitere Stufe, und diese ist das luxuriöseste Fenster, das Sie mit einem einzelnen JavaScript-Befehl erzeugen können: `window.prompt()`. Außer den Schaltflächen **OK** und **Cancel** (oder **Abbrechen**) gibt es noch ein einzeliges Texteingabefeld. Bei der Überprüfung auf Vollständigkeit von Formulareingaben kann das recht nützlich werden.

Der Rückgabewert der Methode ist der eingegebene Text, sofern der Benutzer **OK** anklickt. Bei der Betätigung der Schaltfläche **Cancel** oder **Abbrechen** wird der spezielle Wert `null` zurückgegeben, egal, was in das Textfeld eingegeben worden ist. Der folgende Code (Sie finden ihn auf der CD-ROM unter **confirm.html**) fragt den Namen des Benutzers ab und gibt ihn in einem weiteren modalen Fenster aus – sofern die Schaltfläche **OK** angeklickt und ein Name eingegeben worden ist.

```
<html>
<head>
<title>window.prompt()</title>
</head>
<body>
```

```

<h1>Wie heißen Sie?</h1>
<script type="text/javascript"><!--
var benutzername = prompt("Wie heißen Sie?", "");
if (benutzername != "")
    alert("Guten Tag, Herr/Frau " + benutzername);
//--></script>
</body>
</html>

```

Listing 4.2 Der Benutzername wird ein- und ausgegeben.

4.2 Fenster öffnen

Die Methode zum Öffnen eines neuen Fensters heißt `window.open()`. Dieser Methode werden bis zu drei Parameter übergeben, mindestens jedoch einer.

```
var fenster = window.open(URL, Fenstername, Optionen);
```

Der erste Parameter, `URL`, enthält die URL der Seite, die in das neue Fenster geladen werden soll. Ist `URL` eine leere Zeichenkette, so wird das Fenster zwar geöffnet, aber es wird keine Seite hineingeladen.

4.2.1 Fenster öffnen und schreiben

Die Position des neuen Fensters lässt sich nur ab dem Netscape Navigator 4 bestimmen; ansonsten hängt sie vom Browser ab. Die Größe hängt auch vom Browser ab, kann jedoch auch eingestellt werden (siehe weiter unten).

Mit dem folgenden Code wird die Homepage von Galileo Press in ein neues Fenster geladen:

```
var fenster = window.open("http://www.galileo-press.de/")
```



Verwenden Sie stets `window.open()`, nicht bloß `open()`.

Die Variable `fenster` ist eine Referenz auf das neue Fenster. Somit kann auf Eigenschaften des Fensters zugegriffen werden, beispielsweise auf das Objekt `location` oder auf das Objekt `document`. Letzteres ist besonders interessant, kann man doch so (mittels `write()`) den Inhalt des Fensters komplett mit JavaScript erzeugen. Die folgende Funktion öffnet ein neues Fenster und erzeugt in diesem eine HTML-Seite, die den übergebenen Parameter ausgibt.

```
function neuesfenster(text){
    var fenster = window.open("");
    if (fenster != null) {
        // Hat das Öffnen des Fensters funktioniert?
        fenster.document.open(); // Öffnen für Schreibzugriff
        fenster.document.write("<html><body>");
        fenster.document.write("<h1> + text + "</h1>");
        fenster.document.write("</body></html>");
        fenster.document.close(); // Schreibzugriff beenden
    }
}
```

Wie Sie sehen, können Sie mit `document.open()`, `write()` und `close()` rein theoretisch in jedem Fenster jeden beliebigen HTML-Code erzeugen; Anhängen geht nicht, aber Überschreiben ist möglich. Einige Leute haben das recht trickreich ausgenutzt und beispielsweise die Login-Masken mancher Webmail-Dienste mit dieser Methode überschrieben, wodurch sie die Passwörter von ahnungslosen Benutzern ausspionieren konnten. Aus diesem Grund erlauben die neueren Versionen aller Browser nur noch das Überschreiben von Fenstern, deren aktuelle URL von derselben Domain stammt wie das Fenster mit dem Skript.

4.2.2 Fenster öffnen und verlinken

Der zweite Parameter von `window.open()` gibt den Namen des neuen Fensters an. Man kann diesen Namen – genauso wie bei Frames – als Wert des `target`-Attributs eines HTML-Links angeben; das Ziel des Links wird dann in dem neuen Fenster geöffnet und nicht im aktuellen Fenster. Wenn Sie `window.open()` zweimal verwenden und dabei einen identischen zweiten Parameter angeben, wird nur ein Fenster geöffnet.

Die folgende Seite öffnet beim Laden ein neues Fenster mit der Homepage von Galileo Press. Die anderen Links auf der HTML-Seite öffnen die Homepages einiger Browser-Hersteller – in dem neuen Fenster.

```
<html>
<head>
<title>Neue Fenster</title>
<script type="text/javascript"><!--
function neuesfenster(){
    var fenster = window.open("http://www.galileo-press.de",
                              "neu");
}
```

```
//--></script>
</head>
<body onload="neuesfenster();">
<h1>Hier einige Links...</h1>
<ul>
  <li><a href="http://www.microsoft.com/"
        target="neu">Microsoft</a></li>
  <li><a href="http://www.netscape.com/"
        target="neu">Netscape</a></li>
  <li><a href="http://www.opera.com/"
        target="neu">Opera</a></li>
</ul>
</body>
</html>
```

Listing 4.3 Alle Links werden im neuen Fenster geöffnet.



Dieses Skript befindet sich selbstredend auch auf der CD-ROM zum Buch, und zwar unter dem Dateinamen **links.html**.



Allgemein gilt, dass Sie anstelle von `<body onload="funktionsname();">` auch Folgendes im JavaScript-Code verwenden können:

```
window.onload = funktionsname;
```

Ganz wichtig: Sie müssen die Klammern hinter dem Funktionsnamen weglassen!

4.2.3 Fenster öffnen und anpassen

Der letzte Parameter ist gleichzeitig der interessanteste. Hier können Sie das Aussehen des neuen Browser-Fensters etwas genauer anpassen. Neben der Höhe und Breite des Fensters können Sie auch die meisten Leisten ein- und ausschalten. Es gibt hier zwei Arten von Parametern:

- ▶ Numerische Werte: beispielsweise Höhe oder Breite; diese haben das Format `height=400`,
- ▶ Ja/Nein-Werte: beispielsweise Statuszeile einblenden ja/nein. Die Werte `yes` und `1` oder die bloße Erwähnung der Option bedeuten ja, `no` und `0` bedeuten nein. Beispiel: `status=yes,toolbar=no,menubar`.

Sie können natürlich mehrere Parameter zusammen verwenden. Sie müssen diese dazu durch Kommata voneinander trennen und Sie dürfen

keine Leerzeichen verwenden! Der folgende Befehl öffnet ein Fenster, das 400 Pixel breit ist und eine Statuszeile hat:

```
var fenster = window.open("", "neu", "width=400,status=1")
```

In Tabelle 4.1 finden Sie die wichtigsten möglichen Parameter.

Option	Typ	Beschreibung
dependent	Ja/Nein	Das neue Fenster wird vom aufrufenden Fenster »abhängig«, d. h. beim Schließen des aufrufenden Fensters wird das neue Fenster auch geschlossen.
directories	Ja/Nein	Nur Netscape: Die Schaltflächen What's new , What's cool werden ein- oder ausgeblendet.
height	numerisch	Höhe des Fensters in Pixeln
innerHeight	numerisch	NN4: Höhe des Fensterinhalts in Pixeln (aktiver Bereich, vgl. height)
innerWidth	numerisch	NN4: Breite des Fensterinhalts in Pixeln (aktiver Bereich, vgl. width)
left	numerisch	IE4: Abstand vom linken Bildschirmrand in Pixeln
menubar	Ja/Nein	Menüleiste wird ein- bzw. ausgeblendet.
outerHeight	numerisch	NN4: Gesamthöhe des Fensters (inkl. Titelleiste, Scrollbalken, Rahmen etc.) in Pixeln
outerWidth	numerisch	NN4: Gesamtbreite des Fensters (inkl. Titelleiste, Scrollbalken, Rahmen etc.) in Pixeln
screenX	numerisch	NN4: Abstand vom linken Bildschirmrand in Pixeln
screenY	numerisch	NN4: Abstand vom oberen Bildschirmrand in Pixeln
scrollbars	Ja/Nein	Scrollbalken am rechten und/oder unteren Rand des Browser-Fensters werden angezeigt/unterbunden.
status	Ja/Nein	Die Statuszeile wird ein- oder ausgeblendet.
toolbar	Ja/Nein	Die Navigationsleiste (mit den Schaltflächen für Vor/Zurück/Neu laden etc.) wird ein- oder ausgeblendet.
top	numerisch	IE4: Abstand vom oberen Bildschirmrand in Pixeln
width	numerisch	Breite des Fensters in Pixeln

Tabelle 4.1 Eine Auswahl der Fensteroptionen

Wenn einer dieser Parameter nicht angegeben wird, nimmt der JavaScript-Interpreter an, er sei auf `no` bzw. `0` gesetzt worden. Der folgende Aufruf erzeugt also ein ganz mageres Fenster:

```
var fenster = window.open("", "neu", "");
```

Wie Sie bei den vorhergehenden Beispielen gesehen haben, fehlt der dritte Parameter komplett, so sieht das neue Fenster gemäß den Standardwerten des Browsers aus. Beim Netscape Navigator beispielsweise entsprechen die Ausmaße und angezeigten Leisten denen des zuletzt aktiven Browser-Fensters.

Kommen wir zu einer der eingangs erwähnten Anwendungsmöglichkeiten zurück: zur Anzeige eines nervenden Werbebanners. Wenn Ihr Werbebanner eine Breite von $468 * 60$ Pixeln hat, könnte der Aufruf (in einer Datei **banner-aufruf.html**) folgendermaßen aussehen (ein wenig mehr Breite und Höhe für Titelleiste etc. muss natürlich sein):

```
var fenster = window.open("banner.html", "neu",  
                           "width=500,height=200");  
}
```



Unter UNIX kann die Größe eines Fensters über X-Ressourcen bestimmt werden, die immer Priorität haben.



Aus Sicherheitsgründen muss ein mit JavaScript geöffnetes Fenster mindestens 100 Pixel hoch und 100 Pixel breit sein. Wenn Sie also für Höhe oder Breite einen Wert kleiner hundert angeben, wird das Fenster trotzdem 100 Pixel breit oder hoch dargestellt. Mit dem Netscape Navigator 4 lässt sich das umgehen, indem man signierte Skripte nutzt. Ebenso können Sie (mit den `window`-Eigenschaften `screenX/screenY`) ein Fenster nur dann aus dem sichtbaren Bereich hinaus verschieben, wenn Sie ein signiertes Skript verwenden.

Mit der folgenden HTML-Seite (**fenstertest.html**) können Sie einige Parameter für das Aussehen des neuen Fensters anpassen:

```
<html>  
<head>  
<title>Fenster-Test</title>  
<script type="text/javascript"><!--  
function neuesfenster(formular){  
    var optionen = "width=400,height=300";
```

```

    for (var i=0; i<formular.elements.length; i++){
        if (formular.elements[i].checked)
            optionen += "," + formular.elements[i].name;
    }
    var fenster = window.open("", "neu", optionen);
    if (fenster != null){
        with (fenster.document){
            open();
            write("<html><body>")
            write("Neues Fenster mit folgenden Optionen: " +
                optionen);
            write("<hr />");
            write("<a href='javascript:window.close();'>");
            write("Fenster schließen</a>");
            write("<body></html>");
            close();
        }
    }
}
//--></script>
</head>
<body>
<h1>Fenster-Optionen auswählen</h1>
<form>
<input type="checkbox" name="dependent"> dependent<br />
<input type="checkbox" name="directories"> directories<br />
<input type="checkbox" name="hotkeys"> hotkeys<br />
<input type="checkbox" name="location"> location<br />
<input type="checkbox" name="menubar"> menubar<br />
<input type="checkbox" name="scrollbars"> scrollbars<br />
<input type="checkbox" name="status"> status<br />
<input type="checkbox" name="toolbar"> toolbar<br />
</form>
<a href="javascript:neuesfenster(document.forms[0])">Fenster
erzeugen</a>
</body>
</html>

```

Listing 4.4 Die wichtigsten Fensteroptionen können ausprobiert werden.



Einige Elemente in diesem Listing, beispielsweise der Formularzugriff, werden Ihnen vermutlich erst bei der Lektüre des nächsten Kapitels klar. Der Hauptzweck dieses Beispiels ist, Ihnen eine Möglichkeit zu geben, die Fensteroptionen mit wenig Aufwand ausprobieren zu können.

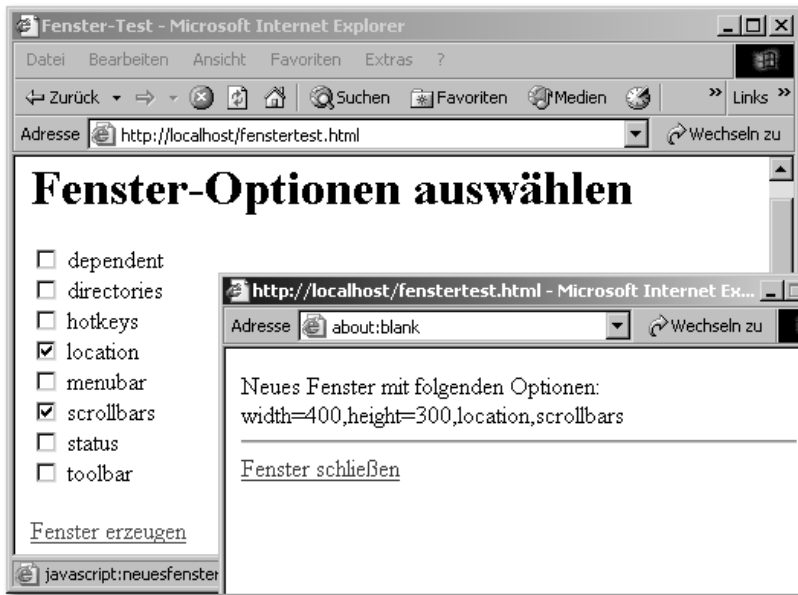


Abbildung 4.1 Die Fensteroptionen im Test

Ein kurzes Wort noch zu der Zeile `if (fenster != null)`: Es kann sein, dass das Öffnen des neuen Fensters nicht funktioniert hat; insbesondere der Internet Explorer gibt bei wenig Speicher gern eine Fehlermeldung aus und öffnet das Fenster dann nicht.

Anwendungsmöglichkeiten für die neuen JavaScript-Fenster gibt es viele: Beispielsweise könnten Sie auf Ihren Webseiten eine Hilfefunktion anbieten. Der Hilfetext kann dann in einem neuen Fenster (ohne platzraubende Symbolleisten) dargestellt werden; dies ist insofern praktisch, als der Benutzer nicht auf die **Zurück**-Schaltfläche seines Browsers klicken muss, um wieder zu der Seite zurückzukehren, auf der er die Hilfe in Anspruch nehmen wollte.

4.3 Zeitversetzte Codeausführung

Ganz kurz wollen wir uns an dieser Stelle den Möglichkeiten einer zeitversetzten Codeausführung zuwenden; in den Kapiteln, die sich mit ActionScript beschäftigen, werden wir öfter davon Gebrauch machen.

JavaScript bietet hierzu zwei Möglichkeiten:

- ▶ `setTimeout(Code, Verzögerung)` führt ein Codestück (oder eine Funktion) nach der spezifizierten Zeit (in Millisekunden!) aus.
- ▶ `setInterval(Code, Intervalllänge)` führt ein Codestück regelmäßig aus. Der Abstand zwischen den Ausführungen wird im zweiten Parameter angegeben, Maßeinheit sind auch hier Millisekunden.

Während `setTimeout()` bereits seit JavaScript 1.0 zur Verfügung steht, gibt es `setInterval()` erst seit JavaScript 1.2.



Was aber hat dies mit Fenstern zu tun? Nun, eigentlich wenig, außer dass `setTimeout()` und `setInterval()` jeweils Methoden des `window`-Objekts sind. Wie aber bereits erwähnt wurde, muss `window` nicht extra davor geschrieben werden.

Nachfolgender Aufruf führt eine Funktion `tue_irgendetwas()` – die natürlich noch geschrieben werden muss – alle fünf Sekunden aus:

```
setInterval("tue_irgendetwas();", 5000);
```

Sie sehen, dass der Code in Anführungszeichen stehen muss. Wenn Sie nur eine (selbst definierte) Funktion aufrufen, können Sie auch eine Referenz auf diese Funktion übergeben. Dazu entfernen Sie die Anführungszeichen und die runden Klammern:

```
setInterval(tue_irgendetwas, 5000);
```

Aber auch mit `setTimeout()` können Sie erreichen, dass eine Funktion regelmäßig aufgerufen wird. Sie müssen nur am Ende der Funktion erneut einen Timeout setzen:

```
function tue_irgendetwas() {  
    // ...  
    setTimeout("tue_irgendetwas();", 5000);  
}  
  
setTimeout("tue_irgendetwas();", 5000);
```

Der Rückgabewert von `setTimeout()` und `setInterval()` ist ein Identifikator, mit dem Sie den Timeout oder das Intervall später wieder löschen und somit die Ausführung des Codes unterbinden können. Dazu müssen Sie diese ID als Parameter an `clearTimeout()` bzw. `clearInterval()` übergeben.

Folgender Timeout wird also nie ausgeführt:

```
var ID = setTimeout("tue_irgendetwas();", 5000);  
clearTimeout(ID);
```



In Kapitel 7, DHTML, finden Sie Beispiele zum Einsatz von Timeouts bzw. Intervallen.

4.4 Auf Frames zugreifen

Nun werfen wir noch einen Blick auf Frames. Per JavaScript ist es möglich, auf jeden einzelnen Frame zuzugreifen. Damit sind oft verblüffende Effekte möglich.

4.4.1 Zugriff

Jeder Frame wird durch ein `window`-Objekt repräsentiert (um ganz genau zu sein: Es gibt Unterschiede, diese sind jedoch marginal). Jedes Fenster – und damit auch jeder Frame – hat als Eigenschaft das Array `frames[]`, welches alle Unterframes des aktuellen Fensters enthält.

Angenommen, Sie haben folgende Frame-Definition:

```
<frameset cols="20%,30%,40%">  
  <frame src="1.html" name="eins" />  
  <frame src="2.html" name="zwei" />  
  <frame src="3.html" name="drei" />  
</frameset>
```

Sie können nun vom Hauptdokument aus (also dem mit dem `<frameset>`) mit JavaScript auf die drei Frames zugreifen. Dazu haben Sie folgende Möglichkeiten:

- ▶ `window.Framename`, also `window.eins`, `window.zwei`, `window.drei`.
- ▶ `window.frames["Framename"]`, also `window.frames["eins"]`, `window.frames["zwei"]`, `window.frames["drei"]`.
- ▶ `window.frames[Index]`, also `window.frames[0]`, `window.frames[1]`, `window.frames[2]`.

Außerdem gibt es noch folgende Schlüsselwörter:

- ▶ Mit `parent` kommen Sie in der Framehierarchie eine Ebene nach oben, erreichen also den übergeordneten Frame (oder das aktuelle Fenster, falls es keinen übergeordneten Frame gibt).
- ▶ Mit `top` erreichen Sie das Hauptfenster, also das Dokument auf der obersten Ebene der Framehierarchie. Auch hier hat `top` als Wert eine Referenz auf das aktuelle Fenster, sollte das aktuelle Fenster gleichzeitig das Hauptdokument sein.

4.4.2 Immer Frames

Eine sehr häufige Anwendung von Frame-Zugriff mit JavaScript ist die folgende. Ihre Website besteht aus zwei Frames: Der linke enthält die Navigation, der rechte den Inhalt. Ihre **index.html**-Datei sieht wie folgt aus:

```
<html>
<head>
<title>Homepage mit Frames</title>
</head>
<frameset cols="150,*">
  <frame src="navigation.html" name="Navi" />
  <frame src="inhalt.html" name="Inhalt" />
</frameset>
</html>
```

Listing 4.5 Das Hauptdokument

In allen Unterseiten soll nun überprüft werden, ob ein Frameset existiert; wenn nicht, soll das Frameset aufgebaut werden. Dies geschieht mit ein wenig JavaScript-Code: Die URL der aktuellen Seite wird über die Kommandozeile an eine Datei **frameset.html** übergeben, die die Frame-Struktur erzeugt.

Der folgende Code gehört in alle Unterseiten:

```
<script type="text/javascript"><!--
if (self == top) location.href = "frameset.html?" +
    location.href;
//--></script>
```

Die Datei **frameset.html** wertet die Kommandozeile aus und baut – mittels `document.write()` – die Frame-Struktur auf. Beachten Sie, dass Sie das gesamte Frameset mit JavaScript erzeugen müssen, eine Mischung ist

nicht möglich (sie wäre zwar syntaktisch prinzipiell korrekt, die Browser kommen damit aber nicht zurecht).

```
<html>
<head>
<title>Homepage mit Frame</title>
</head>
<script type="text/javascript">
document.write("<frameset cols=\"150,*\">");
document.write("<frame src=\"navigation.html\" name=\"Navi\">");
seite = location.search.substring(1,
        location.search.length); // beginnt mit "?" !
document.write("<frame src=\"" + seite + "\" name=\"Inhalt\">");
document.write("</frameset>");
</script>
</html>
```

Listing 4.6 Die Seite erstellt dynamisch das Frameset.

Wenn also ein Benutzer nun direkt eine Inhaltsseite in einem neuen Fenster aufruft, wird er zur Seite **frameset.html** weitergeleitet, wo dann der linke Frame automatisch dazu geladen wird. Sie können dieses Beispiel direkt von der CD-ROM zum Buch aus testen, wenn Sie einmal die Datei **index.html** öffnen (mit Frame-Definition) und einmal die Datei **inhalt.html** (eine Inhaltsdatei, die dann automatisch zu **frameset.html** weiterleitet).



Abbildung 4.2 Die URL zeigt es an – die Frames kommen automatisch.

Die Eigenschaft `location.search` liefert alles zurück, was an die URL mit einem Fragezeichen angehängt worden ist. Allerdings wird auch dieses Fragezeichen mit zurückgeliefert; Sie dürfen daher die Zeichenkette erst ab dem zweiten Zeichen (das den Index 1 hat) auswerten.



4.4.3 Mehrere Frames gleichzeitig ändern

Eine der Fragen, die in Newsgroups am häufigsten gestellt werden, lautet: »Wie kann ich den Inhalt mehrerer Frames gleichzeitig ändern?« Dies ist eine Problemstellung, die tatsächlich häufig vorkommt. Denken Sie an eine der Standardanwendungen für Frames: die Navigation. Wenn Sie einen Punkt in der Navigation auswählen, wird (in einem anderen Frame) die entsprechende Inhaltsseite geladen. Jedoch kann es auch vorkommen, dass sich das Aussehen der Navigation selbst ändern soll (beispielsweise soll der gerade ausgewählte Navigationspunkt hervorgehoben werden, damit der Benutzer weiß, wo er sich befindet).

Dieser Effekt ist sehr einfach umzusetzen. Auf die oben aufgezeigte Art und Weise wird auf die entsprechenden Frames zugegriffen und die Eigenschaft `location.href` entsprechend verändert. Hier lohnt es sich, eine eigene Funktion zu schreiben, die Sie wiederverwenden können. Als Parameter werden Referenzen auf die Frames sowie die zu ladenden URLs übergeben:

```
<script type="text/javascript"><!--  
function ZweiFrames(frame1, url1, frame2, url2){  
    frame1.location.href = url1;  
    frame2.location.href = url2;  
}  
//--></script>
```

Stellen Sie sich vor, im vorhergehenden Beispiel befinden Sie sich im Frame A und wollen via Link in Frame C die Seite **cc.html** sowie in Frame D die Seite **dd.html** laden. Sie können das folgendermaßen tun:

```
<a href="javascript:ZweiFrames(parent.B.C, 'cc.html',  
    parent.B.D, 'dd.html');">hier klicken</a>
```

Sie sollten jedoch immer eine Alternative bereitstellen für den Fall, dass der Browser des Benutzers kein JavaScript unterstützt. In diesem Fall ist das relativ einfach: Wenn Sie sich die Beispielseite noch einmal anschauen, sehen Sie, dass die Frames C und D in der Datei **b.html** defi-

nirt werden. Sie könnten also alternativ eine Datei **bb.html** erstellen, die wie folgt aussieht:

```
<html>
<head>
<title>Noch mehr Frames</title>
</head>
<frameset rows="100,*">
  <frame name="C" src="cc.html" />
  <frame name="D" src="dd.html" />
</frameset>
</html>
```

Der Link vereinfacht sich dann zu folgendem puren HTML:

```
<a href="bb.html" target="B">hier klicken</a>
```

Überlegen Sie also immer, ob Sie die Aufgabenstellung auch ohne JavaScript lösen können. Sie machen Ihre Website dadurch einem größeren Benutzerkreis zugänglich.

5 Grafiken

Kunst ist schön. Macht aber viel Arbeit.

– Karl Valentin

Ohne Grafiken ist das Web nur noch die Hälfte wert. Die Zahl derjenigen Puristen, die mit Textbrowsern wie etwa Lynx unterwegs sind, ist mittlerweile verschwindend gering (inzwischen gibt es auch für Unix/Linux grafikfähige Browser, die **nicht** von Microsoft oder Netscape stammen).

In der Anfangszeit des World Wide Web und der ersten Browser (Netscape Navigator 1/2, Internet Explorer 2) standen insbesondere für den Privatverbraucher nur geringe Übertragungskapazitäten zur Verfügung, so dass Grafiken als notwendiges Übel angesehen wurden. Und auch heute noch bieten die Browser die Möglichkeit, die Anzeige (und insbesondere das Laden) der Grafiken zu unterbinden.¹

In JavaScript werden Grafiken mit dem `Image`-Objekt dargestellt. Der Netscape Navigator 3 führte dieses Objekt erstmals ein. Der Internet Explorer 3 erschien ungefähr zur selben Zeit, unterstützte dieses Objekt zunächst aber nicht, was zu großem Frust bei den Benutzern dieses Browsers führte – damals hatte der Netscape Navigator im »Browserkrieg« noch deutlich die Nase vorn, viele Webmaster testeten nicht einmal mit dem Internet Explorer (mittlerweile ist es oft andersherum: Eine Surftour mit einem Netscape-Browser kann bei schlampig programmierten Seiten einer Geisterbahnfahrt gleichkommen). Die etwas später erschienene Macintosh-Version des Internet Explorer 3 unterstützte schließlich das `Image`-Objekt.

Image-Objekt

Mittlerweile ist das aber ohnehin kein Thema mehr, da inzwischen alle Browser das `Image`-Objekt unterstützen. Dennoch ist in der JavaScript-Programmierung der Einsatz von Code zur Überprüfung, ob das der Fall ist, immer noch obligatorisch. Mit der zunehmenden Verbreitung von Handhelds mit kleinen Displays (und dem damit verbundenen Wunsch, nicht allzu viele Grafiken zu übertragen) besteht auch heute noch ein Interesse daran, JavaScript-Applikationen, die das `Image`-Objekt verwenden, möglichst browserunabhängig zu halten.

Handhelds

¹ Der Netscape Navigator bot in früheren Versionen sogar eine eigene Schaltfläche in der Symbolleiste, mit der die fehlenden Grafiken auf Knopfdruck nachgeladen werden konnten.

5.1 Image-Objekt einbinden

Eine neue Instanz des Image-Objekts wird in der objektorientierten Programmierung wie folgt erstellt:

```
var img = new Image();
```

Danach enthält die Variable `img` ein `Image`-Objekt – wird aber im Speicher gehalten und hat noch nichts mit Grafiken zu tun. Interessant wird es erst, wenn die Eigenschaft `src` dieser Variablen gesetzt wird – der Name der Grafik:

```
var img = new Image();  
img.src = "grafik.png";
```

Nach der Ausführung dieser beiden Codezeilen ist Folgendes passiert:

- ▶ Eine Variable `img` wurde erzeugt.
- ▶ Die `src`-Eigenschaft dieser Variablen wurde auf `"grafik.png"` gesetzt.
- ▶ Der Browser fordert die Datei **grafik.png** an, sollte sie sich noch nicht im Cache des Browsers befinden.



Der letzte Punkt ist für spätere Anwendungen noch entscheidend – die Grafik wird angefordert und geladen.

Größe der Grafik

Die Größe der Grafik kann als Parameter an `new Image()` übergeben werden, zunächst die Breite, dann die Höhe der Grafik:

```
var img = new Image(150, 200);  
img.src = "grafik.png";
```

Diese Vorgehensweise ist aber nicht sehr empfehlenswert.

- ▶ Wenn die Größe nicht angegeben wird, übernimmt JavaScript die tatsächliche Größe der Grafik, wie das der Browser auch tut.
- ▶ Weiter hinten in diesem Kapitel wollen wir Funktionen anbieten, die universell `Image`-Objekte erzeugen sollen. Die Pflicht, eine Grafikgröße angeben zu müssen, würde die Universalität der Funktion aufheben.

Noch ein paar Sätze zur Größe von Grafiken. Im HTML-Code sollten die Attribute `width` und `height` des `<img>`-Tags stets angegeben werden, um einen korrekten Aufbau der Seite schon während des Ladens zu ermöglichen. Beim JavaScript-Aufruf ist das natürlich nicht der Fall. Und wenn Sie auf die Idee kommen sollten, Grafiken durch die Angabe der Parameter für `new Image()` zu skalieren, beachten Sie bitte Folgendes:

- ▶ Wenn Sie die Grafik vergrößern, müssen Sie Qualitätsverluste bei der Grafikanzeige in Kauf nehmen. Browser unterstützen in der Regel nur Bitmapgrafiken. Das sind Grafiken, die aus einzelnen Bildpunkten (Pixeln) bestehen. Beim Vergrößern dieser Grafiken werden die einzelnen Pixel verdoppelt und neben-/untereinander gestellt. Die Grafikqualität sinkt dadurch, das Aussehen der Grafik wird blockartig.
- ▶ Wenn Sie die Grafik verkleinern, ist der Qualitätsverlust freilich nicht so groß. Dennoch, der Benutzer muss die größere Original-Datei übertragen. Eine Datei mit geringerer Breite und Größe wäre kleiner und schneller zu übertragen gewesen. Für diesen Zweck wären Sie also mit einer kleineren Datei besser gefahren.

Beim Internet Explorer und ab Netscape 6 können Sie über die Eigenschaften `width` und `height` die Größe einer Grafik im HTML-Dokument verändern. Der Netscape Navigator 4 unterstützt dies nicht, da dort aufgrund des internen Ablaufs Größenänderungen an Elementen nicht zu einer Neupositionierung des restlichen Inhaltes führen. Aus diesem Grund wird vom Einsatz dieses Effekts an dieser Stelle Abstand genommen.

5.1.1 Grafiken austauschen

Wie können Sie das `Image`-Objekt nun einsetzen? Per se eigentlich gar nicht. Wenn Sie eine Grafik innerhalb einer HTML-Seite austauschen möchten, müssen Sie einfach die `src`-Eigenschaft der Grafik setzen:

```
<html>
<head>
<title>Grafik</title>
<script type="text/javascript"><!--
function rot() {
    document.images["grafik"].src = "rot.png";
}
function gruen() {
    document.images["grafik"].src = "gruen.png";
}
//--></script>
</head>
<body>

<form>
    <input type="button" value="Rot" onclick="rot();">
```

```

        <input type="button" value="Grün" onclick="gruen();">
    </form>
</body>
</html>

```

Listing 5.1 Auf Mausklick ändert sich die Grafik.



Dieses Beispiel trägt auf der CD-ROM den Dateinamen **grafik-1.html**.

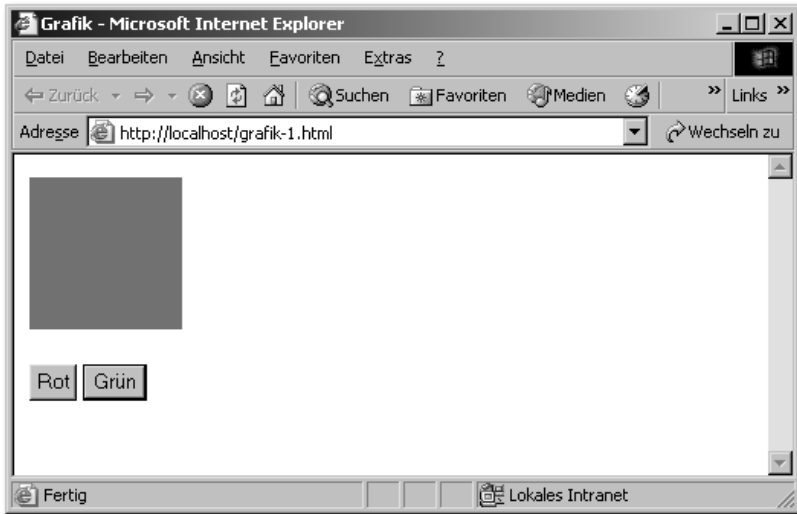


Abbildung 5.1 Schwarzweiß schwer zu sehen, aber das Quadrat ist grün.

5.1.2 Vorladen

Wozu dient nun das `Image`-Objekt? Erinnern Sie sich noch an die Auflistung von oben mit den Auswirkungen des folgenden Codes?

```

var img = new Image();
img.src = "grafik.png";

```

src-Attribut Die im `src`-Attribut angegebene Grafik wird geladen. Damit liegt sie im Cache des Browsers. Das ist im Vergleich zu oben ein großer Vorteil:

- ▶ Bei obigem Code wird die Grafik **rot.png** bzw. **gruen.png** erst geladen, wenn der Benutzer die Schaltfläche anklickt. Es kann also zu Verzögerungen kommen.
- ▶ Wenn die Grafik zuvor mit dem `Image`-Objekt geladen worden ist, erscheint sie nach dem Anklicken der Schaltfläche sofort – sie liegt ja bereits im Zwischenspeicher des Browsers vor und muss nicht mehr vom Webserver abgerufen werden.

Obiges Beispiel kann also verbessert werden:

```
<html>
<head>
<title>Grafik</title>
<script type="text/javascript"><!--
var img_rot = new Image();
img_rot.src = "rot.png";
var img_gruen = new Image();
img_gruen.src = "gruen.png";
function rot() {
    document.images["grafik"].src = "rot.png";
}
function gruen() {
    document.images["grafik"].src = "gruen.png";
}
//--></script>
</head>
<body>

<form>
    <input type="button" value="Rot" onclick="rot();">
    <input type="button" value="Grün" onclick="gruen();">
</form>
</body>
</html>
```

Listing 5.2 Die Grafik wird nun direkt in den Browsercache geladen.

5.1.3 Rollover

Das häufigste Einsatzgebiet des `Image`-Objekts sind die so genannten **Rollover-Effekte**. Sobald die Maus über eine Grafik fährt, ändert diese ihr Erscheinungsbild. Verlässt der Mauszeiger die Grafik wieder, nimmt die Grafik ihre ursprüngliche Gestalt wieder an.

Beim Internet Explorer (ab Version 4) und dem Netscape 6 funktioniert dieser Effekt, indem die Attribute `onmouseover` und `onmouseout` des `<img>`-Tags entsprechend mit JavaScript-Code versehen werden. Beim Netscape Navigator 4 geht das leider schief, da dort die beiden Attribute im `<img>`-Tag ignoriert werden. Sie müssen sich also damit behelfen, einen (funktionslosen) Link um die Grafiken herum zu legen:

```

<html>
<head>
<title>Grafik</title>
<script type="text/javascript"><!--
function rot() {
    document.images["grafik"].src = "rot.png";
}
function gruen() {
    document.images["grafik"].src = "gruen.png";
}
//--></script>
</head>
<body>
<a href="#" onclick="return false;"
    onmouseover="gruen();" onmouseout="rot();">
    </a>
</body>
</html>

```

Listing 5.3 Die Grafik wird beim Überfahren mit der Maus geändert.



Dieses Beispiel finden Sie auf der CD-ROM zum Buch unter dem Dateinamen **grafik-3.html**.

Beachten Sie die weiteren Attribute des Links:

```

<a href="#" onclick="return false;"
    onmouseover="gruen();" onmouseout="rot();">
</a>

```

Link deaktivieren

Der Link wird dadurch deaktiviert, dass er auf die Pseudo-URL # verweist. Die meisten Browser haben jedoch die Angewohnheit, beim Anklicken eines solchen Links an den Beginn des Dokuments zu scrollen. Deswegen sollten zumindest JavaScript-fähige Browser dazu gebracht werden, dies zu verhindern – mit `onclick="return false;"`, was den Link völlig deaktiviert.

Zuletzt noch ein obligatorischer Punkt: die Überprüfung, ob der Browser das Image-Objekt unterstützt oder nicht.

Früher verwendeten viele den Ansatz, einfach den Browsertyp und die Browserversion zu ermitteln. Viele dieser Abfragen scheiterten aus oben genannten Gründen zumindest an der Macintosh-Version des Internet Explorer 3. Eine im Web recht verbreitete Abfrage ging sogar noch weiter und prüfte explizit auf die Browserversion, so unter anderem auf Netscape 3 und Netscape 4.0x. Als der Netscape Navigator 4.5 erschien, schlug diese Abfrage fehl, und Nutzer des damals aktuellsten Netscape-Browsers surfen nur mit halbem Vergnügen. Wobei anheim gestellt werden soll, ob die damit verbundene geringere Datenmenge, die übertragen werden muss, wirklich als Nachteil einzustufen ist.

Die Optimallösung ist die direkte Abfrage des entsprechenden JavaScript-Objekts. Alle Grafiken einer Webseite werden im Array `document.images[]` abgelegt. Die Überprüfung muss also mit `if (document.images)` durchgeführt werden:

```
if (document.images) {
    var img = new Image();
    img.src = "grafik.png";
}
```

Es gibt jedoch einen Browser, der das Image-Objekt unterstützt, aber bei dieser Abfrage »aussteigt«: den Opera 3. Dessen Verbreitung ist aber kaum noch messbar.



5.1.4 Komplettes Beispiel

Ein Listing mit allen vorgestellten Techniken finden Sie im Folgenden (und auf der CD-ROM in der Datei **grafik-4.html**). Diese Elemente sind darin enthalten:

- ▶ Verwendung des Image-Objekts,
- ▶ Rollover,
- ▶ Vorladen (**Preloading**) der Grafiken in den Browser-Cache,
- ▶ Abfrage, ob der Browser das Image-Objekt unterstützt.

```
<html>
<head>
<title>Grafik</title>
<script type="text/javascript"><!--
if (document.images) {
    var img_rot = new Image();
    img_rot.src = "rot.png";
```



```

    var img_gruen = new Image();
    img_gruen.src = "gruen.png";
}

function rot() {
    if (document.images)
        document.images["grafik"].src = "rot.png";
}

function gruen() {
    if (document.images)
        document.images["grafik"].src = "gruen.png";
}
//--></script>
</head>
<body>
<a href="#" onclick="return false;"
    onmouseover="gruen();" onmouseout="rot();">
    </a>
</body>
</html>

```

Listing 5.4 Preloading, Rollover und Browsercheck in einem

5.2 Vorladen per Funktion

Wie Sie zuvor schon gesehen haben, ist das Vorladen Pflicht, insbesondere bei Rollover-Effekten – denn sonst müsste der Benutzer beim Austausch einer Grafik unter Umständen längere Zeit warten, bis sich ein Effekt einstellt. Besser ist es, wenn die Grafiken bereits im Cache des Browsers vorliegen.



Das Ganze funktioniert natürlich auch ohne JavaScript. Sie müssen dazu lediglich die Grafiken in die HTML-Seite einbauen, am besten so, dass der Benutzer sie nicht sieht:

```

```

Sie müssen als Höhe und Breite mindestens einen Pixel angeben, die Angabe `width="0"` oder `height="0"` führt in der Regel dazu, dass die Grafik in ihrer Original-Breite bzw. -Höhe angezeigt wird.

Da das Vorladen (Preloading) einer Grafik häufig benötigt wird, bietet es sich geradezu an, dies in eine Funktion auszulagern.

Vom Funktionsdesign her gibt es zwei Möglichkeiten. Entweder Sie übergeben der Funktion als Parameter eine Liste der Grafiken oder ein Array mit den Grafiknamen.

Beginnen wir mit der Liste der Grafiken: Über das `arguments`-Array der Funktion werden die Parameter ausgelesen und die Grafik-Objekte erzeugt.

**Funktions-
name.arguments**

Die Grafiken werden dabei in unterschiedlichen Variablen gespeichert. Dies hat den Grund, dass Sie die Funktion dann leicht für eigene Zwecke anpassen können, beispielsweise wenn Sie die Variablen noch für andere Zwecke benötigen.

Um das sicherzustellen, wird eine einfache Namenskonvention für die Variablen verwendet: `imgX`, wobei `X` für die laufende Nummer der Grafik steht.

**Namens-
konvention**

```
function vorladen() {
    var arg = vorladen.arguments;
    if (document.images)
        for (var i=0; i<arg.length; i++) {
            eval("var img" + i + " = new Image();");
            eval("img" + i + ".src = \"" + arg[i] + "\";");
        }
}
```

Wenn nicht eine Liste der Grafiken als Parameter übergeben wird, sondern lediglich ein Array, ist die Funktion ein wenig einfacher, da nur das übergebene Array durchlaufen werden muss und nicht die `arguments`-Eigenschaft der Funktion.

```
function vorladen_array(a) {
    if (document.images)
        for (var i=0; i<a.length; i++) {
            eval("var img" + i + " = new Image();");
            eval("img" + i + ".src = \"" + a[i] + "\";");
        }
}
```

Zu guter Letzt ist es sinnvoll, auch den Austausch einer Grafik in einer Funktion zu kapseln. Dazu übergeben wir zwei Parameter: das `name`-Attribut der Grafik sowie den Namen der Datei, die an deren Stelle treten soll:

```
function austauschen(name, grafik) {
    if (document.images && document.images[name]) {
        document.images[name].old = document.images[name].src;
        document.images[name].src = grafik;
    }
}
```

Beachten Sie die folgende Anweisung:

```
document.images[name].old = document.images[name].src;
```

Hiermit legen wir im Image-Objekt eine neue Eigenschaft namens `old` an und speichern darin die URL der »alten« Grafik. Der Grund ist einfach: Das Zurücksetzen der Grafik (beispielsweise mittels `onmouseout`) ist dann sehr einfach:

```
function zuruecksetzen(name) {
    if (document.images && document.images[name]) {
        document.images[name].src = document.images[name].old;
    }
}
```

Im folgenden Beispiel (**grafik-5.html** auf der CD-ROM) werden diese Funktionen eingesetzt (wir binden dazu die externe Datei **images.js**, die den vorherigen Code enthält, ein).

```
<html>
<head>
<title>Grafik</title>
<script type="text/javascript" src="images.js"></script>
</head>
<body onload="vorladen('gruen.png');">
<a href="#" onclick="return false;"
    onmouseover="austauschen('grafik', 'gruen.png');"
    onmouseout="zuruecksetzen('grafik');">
    </a>
</body>
</html>
```

Listing 5.5 Rollover-Effekt mit externen Funktionen



Durch das Auslagern der Funktionen in eine externe Datei schaffen Sie sich schnell eine mächtige Skriptbibliothek und müssen nicht auf jeder Seite das Rad sprichwörtlich neu erfinden.

5.3 Animierte JPEGs

Sie kennen sicherlich die Basis aller nervenden Werbebanner im World Wide Web – animierte GIF-Grafiken. Als das Grafikformat 1987 bzw. 1989 im Auftrag der Firma CompuServe entwickelt wurde, dachte noch niemand daran, dass dieses geheime Feature – nämlich die Integration mehrerer Einzelgrafiken in eine GIF-Datei – später einmal Furore machen würde. Leider ist dies nur mit GIF-Grafiken möglich, die ja auf lediglich 256 Farben beschränkt sind. Fotorealistische Animationen sind damit nicht oder nur in sehr schlechter Qualität möglich.

Mit JavaScript kann dieser Missstand behoben werden, und zwar mit dem `Image`-Objekt. Nach Ablauf einer bestimmten Zeitdauer wird einfach das nächste Bild der Animation eingeblendet. Nachfolgend finden Sie den Code, der die Animation mit einer Verzögerung von einer Sekunde zwischen den Einzelbildern auf den Bildschirm bringt.

```
<html>
<head>
<title>JPEG-Animation</title>
</head>
<script type="text/javascript"><!--
var bilder = new Array("anil.jpg", "ani2.jpg", "ani3.jpg",
"ani4.jpg");

var naechstesBild = 0 //Array-Index des nächsten Bildes
var verzoegerung = 1000 //Verzögerung in Millisekunden

function animation(){
    document.ani.src = bilder[naechstesBild];
    naechstesBild++
    if (naechstesBild==bilder.length)
        naechstesBild = 0
    setTimeout("animation()", verzoegerung)
}
//--></script>
</head>
<body onload="animation();">

</body>
</html>
```

Listing 5.6 Eine JPEG-Animation dank JavaScript

5.3.1 Eine Animation mit JavaScript

Hier folgt die Erklärung der vorhergehenden Codezeilen. Zunächst werden für das Skript ein paar globale Variablen benötigt:

```
var bilder = new Array("anil.jpg", "ani2.jpg", "ani3.jpg",  
"ani4.jpg");
```

```
var naechstesBild = 0 //Array-Index des nächsten Bildes  
var verzoegerung = 1000 //Verzögerung in Millisekunden
```

In dem Array `bilder` stehen die Dateinamen der einzelnen Grafiken der JPEG-Animation. In der Variablen `naechstesBild` wird der Array-Index des nächsten Bildes der Animation gespeichert. Angefangen wird beim ersten Bild. Es besitzt den Index 0, also enthält die Variable den Startwert 0. Die letzte Variable, die die Verzögerung in Millisekunden zwischen den einzelnen Animationsschritten enthält, müsste eigentlich nicht als global deklariert werden. Behalten Sie aber immer im Hinterkopf, dass Ihre Skripte von Leuten bearbeitet werden könnten, die sich nicht so gut in JavaScript auskennen. Für diesen Fall ist es gut, wenn für das Skript wichtige Parameter an markanter Stelle stehen, so dass dort auch unbedarf-tiere Kollegen Änderungen vornehmen können, ohne gleich den gesamten Programmcode zu ruinieren.

Fehlt nur noch die Funktion `animation()`, die das nächste Bild der Animation anzeigt (der Index wird aus der Variablen `naechstesBild` gewonnen) und sich selbst nach einiger Zeit (diese steht in `verzoegerung`) wieder aufruft. Davor muss natürlich noch die Nummer des nächsten Bildes um eins erhöht werden. Ist die Funktion beim letzten Bild angekommen, so soll die Animation wieder von vorn beginnen, `naechstesBild` wird also auf 0 gesetzt.

```
function animation(){  
    document.ani.src = bilder[naechstesBild];  
    naechstesBild++;  
    if (naechstesBild==bilder.length)  
        naechstesBild = 0  
    setTimeout("animation()", verzoegerung)  
}
```

Sie fragen sich, wo die Abfrage bleibt, ob der Browser das `Image`-Objekt überhaupt unterstützt? Der Trick besteht hier darin, dass die Funktion

`animation()` ja erst einmal aufgerufen werden muss – und zwar mit dem `onload`-Event-Handler des `<body>`-Tags:

```
<body onload=" animation();">
```

In Abbildung 5.1 sehen Sie eine Momentaufnahme aus der Animation. Das komplette Skript finden Sie auf der CD-ROM zum Buch unter dem Dateinamen **grafik-6.html**.

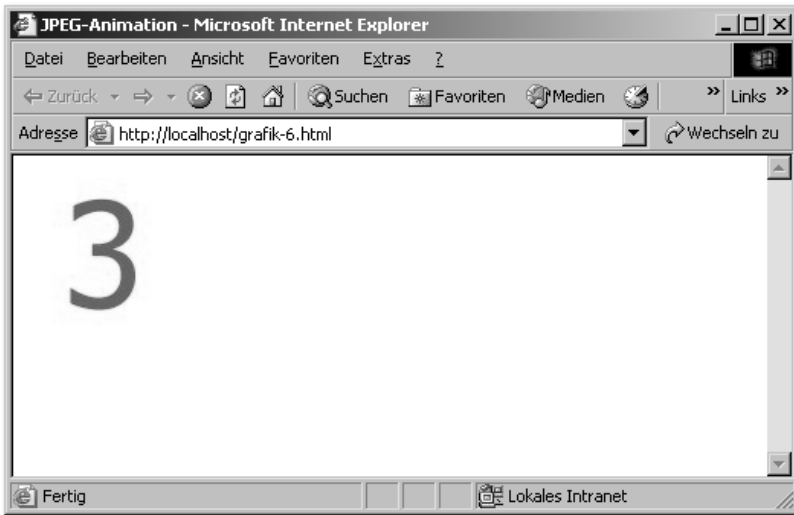


Abbildung 5.2 Eine Animation aus JPEG-Grafiken

5.3.2 Vorladen

Auch der obige Code kann und sollte durch die Funktionalität des Vorladens erweitert werden. Am einfachsten ist es, wenn Sie dazu unsere Bibliothek **images.js** verwenden:

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

Diese wird beispielsweise dazu benutzt, die angegebenen Grafiken in den Browsercache zu laden, was nun mit einer einzigen Anweisung möglich ist:

```
vorladen_array(bilder);
```

Der Vollständigkeit halber können Sie auch das Austauschen der Grafik mit einer der Funktionen aus **images.js** erledigen:

```
austauschen("ani", bilder[naechstesBild]);
```

Nachfolgend das um die Preloading-Funktionalität erweiterte Listing:

```
<html>
<head>
<title>JPEG-Animation</title>
</head>
<script type="text/javascript" src="images.js"></script>
<script type="text/javascript"><!--
var bilder = new Array("anil.jpg", "ani2.jpg", "ani3.jpg",
"ani4.jpg");

var naechstesBild = 0 //Array-Index des nächsten Bildes
var verzoegerung = 1000 //Verzögerung in Millisekunden

vorladen_array(bilder);

function animation(){
    austauschen("ani", bilder[naechstesBild]);
    naechstesBild++
    if (naechstesBild==bilder.length)
        naechstesBild = 0
    setTimeout("animation()", verzoegerung)
}
//--></script>
</head>
<body onload="animation();">

</body>
</html>
```

Listing 5.7 JPEG-Animation samt Vorladen



Dieses Listing finden Sie auf der CD-ROM zum Buch unter dem Namen **grafik-7.html**.

5.4 Fazit

Mouseover-Effekte sind gang und gäbe im World Wide Web. In diesem Kapitel haben Sie gesehen, dass sie sehr einfach zu realisieren sind. Mit der Bibliothek **images.js** müssen Sie dazu nicht einmal mehr großartig selbst programmieren. Rufen Sie einfach die entsprechenden Funktionen auf, den Rest erledigt das Skript.

6 Formulare

Form heißt Verzicht.

– Reinhold Schneider

Es gibt prinzipiell drei Einsatzgebiete für JavaScript: Grafische Effekte (Beispiele gab es in Kapitel 5, Grafiken), Navigationshilfen (zum Beispiel in Kapitel 4, Fenster) und echte Interaktion mit dem Benutzer. Interaktion bedeutet hier, dass der Benutzer etwas tut und die Webseite sich dementsprechend verändert. Sehr oft werden hier HTML-Formulare verwendet. Das hat mehrere Gründe. HTML-Schaltflächen eignen sich vorzüglich, um JavaScript-Funktionen auszuführen, und in HTML-Eingabefeldern können Sie Daten vom Benutzer abfragen. In diesem Kapitel soll es primär darum gehen, einen HTML-Fragebogen mit JavaScript-Funktionalität zu versehen.

6.1 Überprüfung auf Vollständigkeit

Sie haben das garantiert schon einmal im World Wide Web gesehen: Wenn Sie sich bei einem Freemail-Dienst, einer Online-Community oder bei MyGalileo registrieren, müssen Sie einige persönliche Angaben machen. Die Betreiber der Website haben natürlich ein großes Interesse daran, dass das Formular vollständig ausgefüllt wird. Deshalb sollen die Benutzereingaben auf Vollständigkeit überprüft werden. Das kann serverseitig gelöst werden, was den Vorteil hat, dass die Überprüfung auch mit wirklich jedem Browser funktioniert. Die Nachteile liegen aber gleichzeitig auf der Hand. Nicht immer steht dem Programmierer eine serverseitige Programmiersprachen-Unterstützung zur Verfügung. Der Programmieraufwand ist hierbei recht hoch, denn es soll ja nicht nur eine Fehlermeldung ausgegeben werden, sondern nach Möglichkeit das Formular auch noch mit den zuvor eingegebenen Daten dargestellt werden. Es wäre schon sehr ärgerlich für den Benutzer, wenn er ein Formular fast vollständig ausgefüllt hätte und aufgrund einer einzigen Auslassung das komplette Formular noch einmal ausfüllen müsste.

Mit JavaScript kann auch auf der Clientseite die Überprüfung auf Vollständigkeit durchgeführt werden. Damit wird keine neue Seite in den Browser geladen, die Formulardaten bleiben unangetastet, und der Benutzer kann den Fehler sofort korrigieren.

Im Folgenden finden Sie ein HTML-Formular, das ausschließlich aus Pflichtfeldern bestehen soll. Nach dem Anklicken der Absende-Schaltfläche soll die Funktion `pruefen()` aufgerufen werden, die überprüft, ob das Formular vollständig ausgefüllt wurde, und gegebenenfalls eine Fehlermeldung ausgibt. Diese Funktion werden wir in diesem Kapitel schrittweise erarbeiten. Vorab jedoch noch ein paar Hinweise, wie diese Funktion in das HTML-Dokument eingebunden wird.

Das `<form>`-Tag hat den Event-Handler `onsubmit`, der ausgelöst wird, sobald das Formular abgeschickt wird. Wie schon bei anderen Beispielen in diesem Buch können Sie hier einen altbekannten Trick anwenden. Beim Versenden des Formulars kollidieren wieder die HTML-Konsequenzen des Versendens (nämlich das Aufrufen einer anderen Seite) und die mit `onsubmit` zugeordneten JavaScript-Aktionen. Endet der `onsubmit`-Event-Handler jedoch auf `return false`, so wird die HTML-Aktion (das Aufrufen der anderen Seite) nicht ausgeführt; bei `return true` wird sie ausgeführt. Also hat der `<form>`-Tag in diesem Beispiel folgenden Aufbau:

```
<form action="/cgi-bin/skript" name="Fragebogen"
      onsubmit="return pruefen()">
```

Die Funktion `pruefen()` gibt nur dann `true` zurück, wenn das Formular vollständig ausgefüllt worden ist, ansonsten `false`. Somit ist sichergestellt, dass nur vollständige Formulare verschickt werden – sofern der Browser des Benutzers JavaScript unterstützt und es auch aktiviert ist. Ältere Browser oder solche, bei denen JavaScript ausgeschaltet wurde, ignorieren den Skriptteil komplett und versenden das Formular direkt.

```
<html>
<head>
<title>Formular-Überprüfung</title>
<script><!--
function pruefen(){
    // ... kommt noch
    return true
}
//--></script>
</head>
<body>
<h1>Anmeldeformular</h1>
<form action="/cgi-bin/skript" name="Fragebogen"
      onsubmit="return pruefen();">
Vorname: <input type="text" name="Vorname"><br />
```

```

Nachname: <input type="text" name="Nachname"><br />
E-Mail-Adresse: <input type="text" name="Email"><br />
<input type="radio" name="Geschlecht" value="m"> männlich |
<input type="radio" name="Geschlecht" value="w"> weiblich
<br />
Welche Bücher von Galileo Press besitzen Sie?<br />
<input type="checkbox" name="Flash"> Flash |
<input type="checkbox" name="JavaScript"> JavaScript |
<input type="checkbox" name="Rezepte"> JavaScript-Rezepte
<br />
Welches der Bücher hat das schönste Cover?
<select name="Cover">
  <option>Bitte wählen</option>
  <option value="Flash">Flash</option>
  <option value="JavaScript">JavaScript</option>
  <option value="Rezepte">JavaScript-Rezepte</option>
</select><br />
<input type="submit" value="Absenden">
<input type="reset" value="Formular löschen">
</form>
</body>
</html>

```

Listing 6.1 Das (noch funktionslose) HTML-Formular

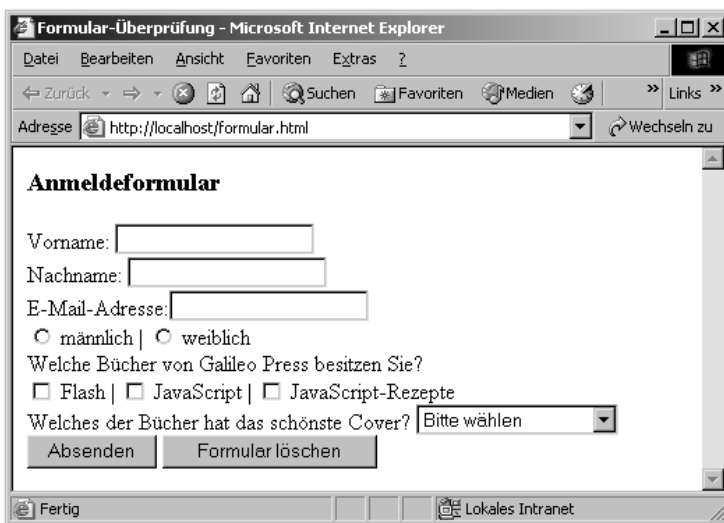


Abbildung 6.1 Das HTML-Formular im Webbrowser

6.1.1 Allgemeiner Aufbau

Alle Formulare werden im Array `document.forms[]` gespeichert. Auf dieses Formular kann über den Array-Index zugegriffen werden, wobei die Formulare in der Reihenfolge im Array stehen, in der sie im HTML-Dokument vorkommen. Die Nummerierung beginnt beim Index 0. In unserem Beispiel benutzen wir also `document.forms[0]`.

Alternativ dazu können Sie auch über das `name`-Attribut des `<form>`-Tags auf das Formular zugreifen, im Beispiel also mit `document.Fragebogen`. Sollte das `name`-Attribut Spezialzeichen wie etwa das Leerzeichen enthalten (beispielsweise "Mein Fragebogen"), so kann noch mit `document.forms["Mein Fragebogen"]` darauf zugegriffen werden, zumindest seit JavaScript Version 1.1. Im Allgemeinen aber sollten Sie den Array-Index verwenden. Da die meisten HTML-Seiten maximal ein Formular enthalten, reicht ein `document.forms[0]`, und es ist auch nicht sonderlich unleserlich. Bei anderen Arrays, die viele Elemente besitzen (in Kapitel 9, Images, finden Sie Beispiele), ist diese Vorgehensweise sicherlich unpraktikabel.

Das Formular speichert all seine Elemente (also alle HTML-Formularelemente in diesem Formular) im Array `elements[]`. Auch hier können Sie auf die einzelnen Elemente über den Array-Index zugreifen; im obigen Beispiel greifen Sie auf das erste Texteingabefeld mit `document.forms[0].elements[0]` zu. Alternativ können Sie auch wieder das `name`-Attribut verwenden, in diesem Fall `document.forms[0].Vorname`. Die meisten Formularelemente haben eine Eigenschaft `value`, die in der Regel die Eingabe des Benutzers angibt. Details hierzu finden Sie in den entsprechenden Abschnitten.

Zur Vereinfachung wird folgende Zeile am Anfang der Funktion `pruefen()` eingeführt:

```
var f = document.Fragebogen;
```

Hiermit sparen Sie sich ein wenig Tipparbeit; statt `document.Fragebogen.elements[0]` reicht jetzt `f.elements[0]`.

6.1.2 Texteingabefelder

Unter Texteingabefeldern werden in diesem Zusammenhang Formularfelder verstanden, die folgendermaßen in HTML abgebildet werden:

- ▶ `<input type="text">` – einzeiliges Texteingabefeld
- ▶ `<input type="password">` – einzeiliges Passwortfeld

- ▶ `<input type="hidden">` – unsichtbares Formularfeld
- ▶ `<textarea>` – mehrzeiliges Texteingabefeld

In diesem Fall gestaltet sich die Vollständigkeitsabfrage sehr einfach. Wie oben schon ausgeführt, heißt die Eigenschaft, die den Eingabewert im Texteingabefeld enthält, `value`. Der folgende JavaScript-Code kann also in die Funktion `pruefen()` eingefügt werden. Er funktioniert so: Ist ein Texteingabefeld nicht ausgefüllt, wird die Funktion mit `return false` verlassen, was dann zur Folge hat, dass das Formular nicht versandt wird.

```
if (f.Vorname.value == "")
    return false;
if (f.Nachname.value == "")
    return false;
if (f.Email.value == "")
    return false;
```

Auf eine Besonderheit bei Texteingabefeldern muss noch hingewiesen werden: Wenn Sie mehrere Texteingabefelder mit identischen `name`-Attributen haben, werden die Werte in einem Array gespeichert. Enthielte das Beispielformular von oben also noch ein Texteingabefeld namens `Vorname`, so könnte man auf das erste dieser Felder mit `f.Vorname[0]` zugreifen und auf das zweite Feld mit `f.Vorname[1]`. Beim Internet Explorer 3 funktioniert das leider nicht, hier müssen Sie über das `elements[]`-Array auf die Texteingabefelder zugreifen. Aus diesem Grund ein einfacher Tipp: Verwenden Sie stets eindeutige Bezeichnungen für Ihre Formularelemente.

6.1.3 Radiobuttons

Aus einer Gruppe von Radiobuttons kann immer nur einer ausgewählt werden. Eine Gruppe von Radiobuttons ist durch ein identisches `name`-Attribut gekennzeichnet. Leider greift man im obigen Beispiel mit `f.Geschlecht` auf die Gruppe von Radiobuttons mit dem `name`-Attribut `Geschlecht` zu. Die Eigenschaft `value` liefert hier nicht das richtige Ergebnis: Es muss auf einen individuellen Radiobutton zugegriffen werden. Das wäre natürlich über das `elements[]`-Array möglich, aber es gibt einen eleganten Weg. Analog zu Texteingabefeldern werden gleichnamige Radiobuttons auch in einem Array abgespeichert. Mit der folgenden Schleife werden alle Radiobuttons (in diesem Fall zwei) durchlaufen und es wird überprüft, ob der Button ausgewählt ist (boolesche Eigenschaft `checked`).

Wenn keiner der beiden ausgewählt ist, wird `pruefen()` mit `return false` verlassen.

```
var geschlecht_ok = false
for (var i=0; i<f.Geschlecht.length; i++)
    if (f.Geschlecht[i].checked)
        geschlecht_ok = true;
if (!geschlecht_ok)
    return false;
```

Alternativ dazu können Sie eine Variante ohne Schleife benutzen, die etwas kürzer ist:

```
if (!(f.Geschlecht[0].checked || f.Geschlecht[1].checked))
    return false;
```

6.1.4 Checkboxes

Checkboxen können wie Radiobuttons ausgewählt werden, wobei sich die grafische Darstellung leicht unterscheidet: Checkboxen werden angekreuzt. Da Checkboxen nicht gruppiert sind, also jede Checkbox für sich ausgewählt werden kann (oder auch nicht), ist die Überprüfung auf Ausfüllung nicht weiter schwer. Wieder gibt es eine boolesche Eigenschaft `checked`, die angibt, ob eine Checkbox angekreuzt ist oder nicht. Das Formular ist dann unvollständig ausgefüllt, wenn keine der drei Checkboxen angekreuzt wurde.

```
if (!f.Flash.checked &&
    !f.JavaScript.checked &&
    !f.Rezepte.checked)
    return false;
```

6.1.5 Auswahllisten

Eine HTML-Auswahlliste wird mit dem Tag `<select>` eingeleitet; es ist sinnvoll, diesem Tag ein aussagekräftiges `name`-Attribut zu geben. Die einzelnen Elemente werden durch das HTML-Tag `<option>` dargestellt. Dieses Tag hat standardmäßig kein `name`-Attribut; wenn man also dort Informationen unterbringen will, sollte man das `value`-Attribut besetzen.

In JavaScript können Sie über das `name`-Attribut des `<select>`-Tags auf die Auswahlliste zugreifen. Die einzelnen Optionen sind im Array `options[]` abgespeichert. Jede Option hat die boolesche Eigenschaft `selected`, die angibt, ob die entsprechende Option gerade ausgewählt ist. Mit der

Eigenschaft `value` können Sie auf das `value`-Attribut der Option zugreifen.

Im Fragebogen-Beispiel kommt es darauf an, dass nicht die erste Option ausgewählt wird, denn die heißt "Bitte wählen". Ein erster Versuch der Abfrage lautet also wie folgt:

```
if (f.Cover.options[0].selected)
    return false
```

Wenn festgestellt werden soll, welche der Optionen gerade ausgewählt ist, könnte man auf die Idee kommen, das `options[]`-Array mit einer Schleife zu durchlaufen, um so den Index der ausgewählten Option herauszubekommen:

```
for (var i=0; i<f.Cover.options.length; i++)
    if (f.Cover.options[i].selected)
        var Auswahl = i;
if (Auswahl == 0)
    return false;
```

Diese Arbeit können Sie sich jedoch sparen: Auswahllisten haben die Eigenschaft `selectedIndex`, die den Array-Index der ausgewählten Option angibt. Die Abfrage kann also folgendermaßen vereinfacht werden:

```
if (f.Cover.selectedIndex == 0)
    return false;
```

Es gibt auch Auswahllisten, bei denen mehr als eine Option ausgewählt werden kann. Diese werden in HTML mit `<select multiple>` abgebildet. In diesem Fall enthält `selectedIndex` den Index der obersten ausgewählten Option.

Wenn Sie auf alle ausgewählten Elemente in der Auswahlliste zugreifen möchten, durchlaufen Sie in einer Schleife alle Elemente und überprüfen Sie deren `selected`-Eigenschaft. Oder prüfen Sie, ob `selectedIndex` den Wert `-1` hat (dann nämlich ist kein Element der Liste ausgewählt).



Nachfolgend finden Sie das HTML-Formular, mit JavaScript-Code angereichert. Wenn ein Fehler aufgetreten ist, wird eine Fehlermeldung ausgegeben.

```
<html>
<head>
```

```

<title>Formular-Überprüfung</title>
<script><!--
function pruefen(){
    var f = document.forms[0];
    var fehler = false; // Fehler?
    // *** Überprüfung auf vollständige Ausfüllung
    if (f.Vorname.value == "")
        fehler = true;
    if (f.Nachname.value == "")
        fehler = true;
    if (f.Email.value == "")
        fehler = true;
    if (!(f.Geschlecht[0].checked ||
        f.Geschlecht[1].checked))
        fehler = true;
    if (!f.Flash.checked &&
        !f.JavaScript.checked &&
        !f.Rezepte.checked)
        fehler = true;
    if (f.Cover.selectedIndex == 0)
        fehler = true;
    // *** Gegebenenfalls Fehlermeldung
    if (fehler) {
        alert("Bitte füllen Sie alle Felder aus!");
        return false;
    } else
        return true;
}
//--></script>
</head>
<body>
<h1>Anmeldeformular</h1>
<form action="/cgi-bin/skript" name="Fragebogen"
    onsubmit="return pruefen();">
Vorname: <input type="text" name="Vorname"><br />
Nachname: <input type="text" name="Nachname"><br />
E-Mail-Adresse: <input type="text" name="Email"><br />
<input type="radio" name="Geschlecht" value="m"> männlich |
<input type="radio" name="Geschlecht" value="w"> weiblich
<br />
Welche Bücher von Galileo Press besitzen Sie?<br />

```

```

<input type="checkbox" name="Flash"> Flash |
<input type="checkbox" name="JavaScript"> JavaScript |
<input type="checkbox" name="Rezepte"> JavaScript-Rezepte
<br />
Welches der Bücher hat das schönste Cover?
<select name="Cover">
    <option>Bitte wählen</option>
    <option value="Flash">Flash</option>
    <option value="JavaScript">JavaScript</option>
    <option value="Rezepte">JavaScript-Rezepte</option>
</select><br />
<input type="submit" value="Absenden">
<input type="reset" value="Formular löschen">
</form>
</body>
</html>

```

Listing 6.2 Das Formular wird auf vollständige Ausfüllung überprüft.

Dieses Listing befindet sich (wie alle anderen) auf der CD-ROM zum Buch; der Dateiname lautet **formularcheck.html**.



Abbildung 6.2 Einige Felder wurden wohl übersehen.

6.1.6 Fehlermeldung ausgeben

Die obige Fehlermeldung ist jedoch wenig konstruktiv. Insbesondere, wenn Sie (wie wir im Beispiel oben) vergessen haben, Pflichtfelder besonders zu kennzeichnen (etwa durch Fettdruck der Beschriftung oder ein Sternchen samt zugehöriger Erklärung), fragt sich der Benutzer oft zu Recht, wo er denn nun einen Fehler gemacht hat.

Deswegen geben wir hier eine neue Version der Vollständigkeitsüberprüfung des Formulars wieder. Es wurden Kommandos hinzugefügt, die eine Warnmeldung ausgeben, sofern das Formular unvollständig ausgefüllt wurde. Anstelle einer booleschen Variablen wird diesmal eine Stringvariable verwendet. In dieser landen die Bezeichnungen aller nicht ausgefüllten Felder. Wenn die Variable leer ist, wurde das Formular vollständig ausgefüllt, ansonsten nicht. Im letzteren Fall kann eine detaillierte Meldung ausgegeben werden, welche Felder nicht ausgefüllt wurden. Auf der CD-ROM zum Buch hat dieses Beispiel den Dateinamen **formular-check-meldung.html**.

```
<html>
<head>
<title>Formular-Überprüfung</title>
<script><!--
function pruefen(){
    var f = document.forms[0];
    var fehler = ""; // enthält die Bezeichnungen
                    // der nicht ausgefüllten Felder
    // *** Überprüfung auf vollständige Ausfüllung
    if (f.Vorname.value == "")
        fehler += "Vorname ";
    if (f.Nachname.value == "")
        fehler += "Nachname ";
    if (f.Email.value == "")
        fehler += "E-Mail ";
    if (!(f.Geschlecht[0].checked ||
        f.Geschlecht[1].checked))
        fehler += "Geschlecht ";
    if (!(f.Flash.checked &&
        !f.JavaScript.checked &&
        !f.Rezepte.checked)
        fehler += "Bücher ";
    if (f.Cover.selectedIndex == 0)
```

```

        fehler += "Cover ";
    // *** Gegebenenfalls Fehlermeldung
    if (fehler != "") {
        var fehlertext = "Die folgenden Felder wurden nicht
                           vollständig ausgefüllt:\n";
        fehlertext += fehler;
        alert(fehlertext);
        return false;
    }
    return true;
}
//--></script>
</head>
<body>
<h1>Anmeldeformular</h1>
<form action="/cgi-bin/skript" name="Fragebogen"
      onsubmit="return pruefen();">
Vorname: <input type="text" name="Vorname"><br />
Nachname: <input type="text" name="Nachname"><br />
E-Mail-Adresse: <input type="text" name="Email"><br />
<input type="radio" name="Geschlecht" value="m"> männlich |
<input type="radio" name="Geschlecht" value="w"> weiblich
<br />
Welche Bücher von Galileo Press besitzen Sie?<br />
<input type="checkbox" name="Flash"> Flash |
<input type="checkbox" name="JavaScript"> JavaScript |
<input type="checkbox" name="Rezepte"> JavaScript-Rezepte
<br />
Welches der Bücher hat das schönste Cover?
<select name="Cover">
    <option>Bitte wählen</option>
    <option value="Flash">Flash</option>
    <option value="JavaScript">JavaScript</option>
    <option value="Rezepte">JavaScript-Rezepte</option>
</select><br />
<input type="submit" value="Absenden">
<input type="reset" value="Formular löschen">
</form>
</body>
</html>

```

Listing 6.3 Der Code zur Vollständigkeitsüberprüfung



Abbildung 6.3 Alle nicht ausgefüllten Felder werden gemeldet.

6.1.7 Konstruktive Vorschläge

Bei Texteingabefeldern gibt es noch eine Alternative zur »passiven« Fehlermeldung: Sie können auch die Methode `window.prompt()` benutzen, um den Benutzer direkt zu einer Texteingabe zu zwingen. Diese Methode wurde bereits erläutert. Sie müssen in diesem Beispiel noch zusätzlich beachten, dass die `value`-Eigenschaft der Texteingabefelder auch auf den eingegebenen Wert gesetzt werden muss, damit diese Daten auch an das serverseitige Skript übergeben werden können.

Dazu müssen Sie in der Funktion `pruefen()` den Code für die drei Textfelder – Vorname, Nachname, Email – durch den folgenden Code ersetzen:

```
while (f.Vorname.value == "") {
    var vorname = prompt(
        "Bitte geben Sie Ihren Vornamen ein!", "");
    if (vorname)
        f.Vorname.value = vorname;
}
while (f.Nachname.value == "") {
    var nachname = prompt(
        "Bitte geben Sie Ihren Nachnamen ein!", "");
```

```

    if (nachname)
        f.Nachname.value = nachname;
}
while (f.Email.value == "") {
    var email = prompt(
        "Bitte geben Sie Ihre E-Mail-Adresse ein!", "");
    if (email)
        f.Email.value = email;
}

```

Gehen wir eine solche Anweisung einmal schrittweise durch. Zunächst wird überprüft, ob im entsprechenden Feld kein Wert steht:

```
while (f.Vorname.value == "") {
```

Falls das Feld leer ist, wird der Benutzer per `window.prompt()` aufgefordert, den Wert für das Feld anzugeben:

```

    var vorname = prompt(
        "Bitte geben Sie Ihren Vornamen ein!", "");

```

Geben Sie als zweiten Parameter immer eine leere Zeichenkette an, sonst ist das Feld mit "undefined" vorausgefüllt!



Falls der Benutzer in das Feld tatsächlich einen Wert eingegeben hat, wird dieser in das Formular eingetragen:

```

    if (vorname)
        f.Vorname.value = vorname;
}

```

Der Benutzer steckt also in einer Endlosschleife fest und wird gezwungen, die Angaben zu machen, ansonsten geht es nicht weiter. So ein Abfragefenster wirkt allerdings relativ aufdringlich, und wenn ein Benutzer partout nichts angeben will, dann macht er eben Falschangaben. Sie werden mit diesen Mitteln niemanden dazu bringen, die Wahrheit zu sagen, wenn er nichts über sich preisgeben will. Vergisst aber jemand, ein Mussfeld auszufüllen, bieten Sie ihm eine bequeme Möglichkeit an, dies nachzuholen.

Hier das komplette Listing – auf der CD-ROM übrigens unter dem Dateinamen **formularcheck-prompt.html**:

```

<html>
<head>
<title>Formular-Überprüfung</title>
<script><!--
function pruefen(){
    var f = document.forms[0];
    var fehler = ""; //enthält die Bezeichnungen
                        //der nichtausgefüllten Felder
    // *** Überprüfung auf vollständige Ausfüllung
    while (f.Vorname.value == "") {
        var vorname = prompt(
            "Bitte geben Sie Ihren Vornamen ein!", "");
        if (vorname)
            f.Vorname.value = vorname;
    }
    while (f.Nachname.value == "") {
        var nachname = prompt(
            "Bitte geben Sie Ihren Nachnamen ein!", "");
        if (nachname)
            f.Nachname.value = nachname;
    }
    while (f.Email.value == "") {
        var email = prompt(
            "Bitte geben Sie Ihre E-Mail-Adresse ein!", "");
        if (email)
            f.Email.value = email;
    }
    if (!(f.Geschlecht[0].checked||f.Geschlecht[1].checked))
        fehler += "Geschlecht ";
    if (!f.Flash.checked &&
        !f.JavaScript.checked &&
        !f.Rezepte.checked)
        fehler += "Lieblingsbuch ";
    if (f.Cover.selectedIndex==0)
        fehler += "Cover ";
    // *** Gegebenenfalls Fehlermeldung
    if (fehler != ""){
        var fehlertext = "Die folgenden Felder wurden nicht voll-
ständig ausgefüllt:\n";
        fehlertext += fehler;
        alert(fehlertext);
    }
}

```

```

        return false;
    }
    return true;
}
//--></script>
</head>
<body>
<h1>Anmeldeformular</h1>
<form action="/cgi-bin/skript" name="Fragebogen"
    onsubmit="return pruefen();">
Vorname: <input type="text" name="Vorname"><br />
Nachname: <input type="text" name="Nachname"><br />
E-Mail-Adresse: <input type="text" name="Email"><br />
<input type="radio" name="Geschlecht" value="m"> männlich |
<input type="radio" name="Geschlecht" value="w"> weiblich
<br />
Welche Bücher von Galileo Press besitzen Sie?<br />
<input type="checkbox" name="Flash"> Flash |
<input type="checkbox" name="JavaScript"> JavaScript |
<input type="checkbox" name="Rezepte"> JavaScript-Rezepte
<br />
Welches der Bücher hat das schönste Cover?
<select name="Cover">
    <option>Bitte wählen</option>
    <option value="Flash">Flash</option>
    <option value="JavaScript">JavaScript</option>
    <option value="Rezepte">JavaScript-Rezepte</option>
</select><br />
<input type="submit" value="Absenden">
<input type="reset" value="Formular löschen">
</form>
</body>
</html>

```

Listing 6.4 Fehlermeldungen mit Korrekturmöglichkeit

Abbildung 6.4 Dem Benutzer wird das Ausfüllen direkt aufgedrängt.



Testen Sie Ihren Code sehr gründlich. Folgender Tippfehler führt dazu, dass der Benutzer in einer Endlosschleife festsitzt, die er nicht mehr verlassen kann:

```
while (f.Email.value == "") {
    var email = prompt(
        "Bitte geben Sie Ihre E-Mail-Adresse ein!", "");
    if (email)
        f.Nachname.value = email;
}
```

Dieser Fehler schleicht sich bei der Verwendung von Copy-and-Paste sehr schnell ein, und er ruiniert das Skript. Nicht das Email-Feld wird aktualisiert, sondern das Nachname-Feld. Außerdem bleibt das Email-Feld immer leer, egal, was der Benutzer in das Abfragefenster eingibt. Es geht also an dieser Stelle nicht weiter, und der Benutzer kommt so schnell nicht wieder auf Ihre Seiten.

6.2 Automatische Überprüfung



Das Skript zur Vollständigkeitsüberprüfung muss für jedes Formular neu geschrieben werden. Zwar können Sie Elemente wiederverwenden, aber etwas zusätzliche Arbeit ist immer nötig. Seit JavaScript-Version 1.1 ist es

aber möglich, die Überprüfung auch zu automatisieren. Jedes Formularelement besitzt nämlich die Eigenschaft `type`, die den Typ des Elements – beispielsweise Texteingabefeld, Radiobutton oder Checkbox – angibt. Man kann also den Ansatz verfolgen, in einer Schleife alle Formularelemente durchzugehen und zu überprüfen, ob sie ausgefüllt sind. Die Fehlermeldungen sind dann vielleicht nicht mehr so individuell wie im vorherigen Beispiel, aber Sie ersparen sich eine Menge Tipp- und Denkarbeit.

Im Folgenden sehen Sie das Grundgerüst für die JavaScript 1.1-fähige Version der Funktion `pruefen()`. Bei gleichlautenden Funktionsnamen wird immer die zuletzt definierte Funktion ausgeführt. Ältere Browser (sofern sie überhaupt noch im Einsatz sind) führen eine eingeschränkte, andere Funktion `pruefen()` aus, die lediglich durch `return true` für das Versenden des Formulars sorgt.

```
<script language="JavaScript"><!--
function pruefen(){
    return true
}
//--></script>
<script language="JavaScript1.1"><!--
function pruefen(f) { // Formular als Parameter
    var fehler = ""; // Variable für die Fehlermeldung
    var i; // Enthält das jeweilige Formularelement
    for (var j=0; j<f.elements.length; j++){
        i = f.elements[j];
        // kommt noch...
    }
    return true
}
//--></script>
```

6.2.1 Texteingabefelder

Der Wert der `type`-Eigenschaft eines Texteingabefelds ist bei einem einzeiligen Texteingabefeld `"text"`, bei einem Passwortfeld `"password"` und bei einem mehrzeiligen Eingabefeld `"textarea"`. In diesem Fall wird die `value`-Eigenschaft des Formularelements (zur Erinnerung: Es steht in der Variablen `i`) überprüft. Falls keine Eingabe erfolgt ist, wird das `name`-Attribut des Eingabefelds (es steht in der Eigenschaft `name`) dem Fehlerstring hinzugefügt, um wenigstens eine einigermaßen aussagekräftige Fehlermeldung ausgeben zu können.


```

if (i.type == "text" ||
    i.type == "password" ||
    i.type == "textarea")
    if (i.value == "")
        fehler += i.name + "\n";

```

6.2.2 Radiobuttons

Für Radiobuttons (erkennbar an der `type`-Eigenschaft "radio") müssen Sie sich eine besondere Strategie überlegen. Jeder Radiobutton ist ein eigenes Formularelement und muss nicht unbedingt ausgewählt sein. Auf folgende Art und Weise kommen Sie dennoch zum Ziel:

1. Wenn Sie feststellen, dass das aktuell betrachtete Formularelement ein Radiobutton ist, lesen Sie das `name`-Attribut aus (Eigenschaft `name`). Dann durchsuchen Sie das zugehörige Array von Radiobuttons, ob zumindest einer davon ausgewählt ist. Falls nicht, wird der Fehlerstring um das `name`-Attribut der Gruppe Radiobuttons erweitert.
2. Jeder Radiobutton wird wie ein einzelnes Formularelement behandelt. Ist also von einer Gruppe aus fünf Radiobuttons keiner ausgewählt, werden bei der Überprüfung der fünf Radiobuttons insgesamt fünfmal Mängel gefunden – es würde also fünf Fehlermeldungen geben. Aus diesem Grund wird eine weitere Variable, `radiocheck`, eingeführt (und zwar außerhalb der `for`-Schleife), in der die `name`-Attribute bereits geprüfter Radio-Buttons, durch Leerzeichen getrennt, eingetragen werden. Bevor ein Radiobutton überprüft wird, muss zuerst nachgeschaut werden, ob schon einmal ein anderer Radiobutton aus dieser Gruppe betrachtet worden ist. Daraus folgt auch eine Einschränkung für das Skript: Die `name`-Attribute der Radiobuttons dürfen keine Leerzeichen enthalten – aber das sollte sich von selbst verstehen, wir verweisen auf die Einleitung.

```

if (i.type == "radio") {
    if (radiocheck.indexOf(i.name + " ") < 0) {
        radiocheck += i.name + " ";
        var radiogroup = f[i.name];
        var ok = false;
        for (var k=0; k<radiogroup.length; k++)
            if (radiogroup[k].checked)
                ok = true;
        if (!ok)

```

```

        fehler += i.name + "\n";
    }
}

```

6.2.3 Checkboxes

Bei Checkboxes ist eine allgemeine Überprüfung schlecht möglich. Es ist dem Benutzer ja freigestellt, ob er eine Checkbox ankreuzt oder nicht. Jede Checkbox ist im Grunde genommen eine Ja/Nein-Frage, und somit bedeutet kein Kreuz auch eine Antwort, nämlich Nein. Aus diesem Grund werden Checkboxes (die `type`-Eigenschaft heißt übrigens "checkbox") nicht überprüft.

6.2.4 Auswahllisten

Auswahllisten erkennt man daran, dass die `type`-Eigenschaft des Formularelements entweder den Wert "select-one" oder den Wert "select-multiple" hat, je nachdem, ob der HTML-Parameter `Multiple` gesetzt worden ist oder nicht. Die Eigenschaft `selectedIndex` muss überprüft werden. Wir unterscheiden zwischen zwei Fällen:

- **Einfache Auswahlliste.** Dann muss `selectedIndex` ungleich 0 sein. Wir nehmen einfach einmal an, dass die oberste Option einer Auswahlliste immer »Bitte wählen Sie« oder ähnlich lautet.
- **Mehrfach-Auswahlliste.** Dann bedeutet -1 als Wert für `selectedIndex`, dass gar keine Option ausgewählt worden ist.

```

if (i.type == "select-one")
    if (i.selectedIndex == 0) // ggf. Ersetzen durch -1
        fehler += i.name + "\n";

if (i.type == "select-multiple")
    if (i.selectedIndex == -1)
        fehler += i.name + "\n";

```

6.2.5 Zusammenfassung

Im Folgenden finden Sie noch einmal den gesamten Code für dieses Beispiel. Mit diesem Code sind Sie in der Lage, mit JavaScript Version 1.1 jedes Formular auf vollständige Ausfüllung hin zu überprüfen. In der Praxis kommt es aber oft vor, dass nicht alle Felder Pflichtfelder sind. Viele Benutzer sind auch sehr zurückhaltend, wenn sie allzu persönliche Daten wie etwa Geburtsdatum, E-Mail-Adresse, Telefonnummer oder gar ihr

Einkommen angeben müssen. Überlegen Sie sich also ganz genau, ob Sie wirklich alle Daten benötigen.

```
<html>
<head>
<title>Formularüberprüfung mit JS 1.1</title>
<script language="JavaScript"><!--
function pruefen(){
    return true
}
//--></script>
<script language="JavaScript1.1"><!--
function pruefen(f){
    var fehler = ""; // Variable für die Fehlermeldung
    var radiocheck = ""; // Variable für überprüfte
                        // Radiobutton-Gruppen
    var i; // Enthält das jeweilige Formularelement
    for (var j=0; j<f.elements.length; j++){
        var i = f.elements[j];
        //Texteingabefelder
        if (i.type == "text" ||
            i.type == "password" ||
            i.type == "textarea")
            if (i.value == "")
                fehler += i.name + "\n";
        //Radiobuttons
        if (i.type == "radio") {
            if (radiocheck.indexOf(i.name + " ") < 0) {
                radiocheck += i.name + " ";
                var radiogroup = f[i.name];
                var ok = false;
                for (var k=0; k<radiogroup.length; k++)
                    if (radiogroup[k].checked)
                        ok = true;
                if (!ok)
                    fehler += i.name + "\n";
            }
        }
        //Auswahllisten
        if (i.type == "select-one")
            if (i.selectedIndex == 0)
                fehler += i.name + "\n";
    }
}
```

```

        if (i.type == "select-multiple")
            if (i.selectedIndex == -1)
                fehler += i.name + "\n";
    }
    //Fehlermeldung
    if (fehler != ""){
        alert("Bitte füllen Sie die folgenden Felder aus:\n" +
            fehler)
        return false;
    }
    return true;
}
//--></script>
</head>
<body>
<form action="/cgi-bin/skript"
    onsubmit="return pruefen(this);">
Vorname: <input type="text" name="Vorname"><br />
Nachname: <input type="text" name="Nachname"><br />
E-Mail-Adresse: <input type="text" name="Email"><br />
<input type="radio" name="Geschlecht" value="m"> männlich |
<input type="radio" name="Geschlecht" value="w"> weiblich
<br />
Welche Bücher von Galileo Press besitzen Sie?<br />
<input type="checkbox" name="Flash"> Flash |
<input type="checkbox" name="JavaScript"> JavaScript |
<input type="checkbox" name="Rezepte"> JavaScript-Rezepte
<br />
Welches der Bücher hat das schönste Cover?
<select name="Cover">
    <option>Bitte wählen</option>
    <option value="Flash">Flash</option>
    <option value="JavaScript">JavaScript</option>
    <option value="Rezepte">JavaScript-Rezepte</option>
</select><br />
<input type="submit" value="Absenden">
<input type="reset" value="Formular löschen">
</form>
</body>
</html>

```

Listing 6.5 Formularüberprüfung ganz automatisch



Sie finden die Datei auf der CD-ROM zum Buch unter **formularcheck-automatisch.html**.



Abbildung 6.5 Alle Felder bis auf Checkboxes werden überprüft.

Wenn Sie dieses Skript in Ihre eigenen Formulare einbauen möchten, befolgen Sie einfach folgende Schritte:

- ▶ Fügen Sie die Funktion `pruefen()` wie oben gezeigt in Ihre HTML-Seite ein.
- ▶ Setzen Sie im `<form>`-Tag `onsubmit="return pruefen();"`
- ▶ Vergeben Sie einprägsame und »sprechende« `name`-Attribute für Ihre Formularfelder!



Wie in Abbildung 6.5 zu sehen und bereits zuvor im Text erwähnt, überprüfen wir Checkboxes aus nahe liegenden Gründen nicht. Sollten Sie diese Funktionalität dennoch benötigen, hier das zugehörige Code-schnipsel:

```
//Texteingabefelder
if (i.type == "checkbox")
    if (i.checked == false)
        fehler += i.name + "\n";
```

7 DHTML

Brüllt ein Mann, ist er dynamisch. Brüllt eine Frau, ist sie hysterisch.

– Hildegard Knef

Stellenanzeigen aus dem IT-Bereich bieten oft Anlass zum Schmunzeln (in Zeiten der Dotcom-Pleiten leider weniger, da es weniger Anzeigen gibt). Oft fragt man sich, ob die Personalabteilung mit der Technikabteilung überhaupt Rücksprache gehalten hat, bevor die getextete und gelayoutete Anzeige übermittelt worden ist.

Das Studium von Stellenanzeigen ist aber auch sehr lehrreich, denn dort finden Sie nicht nur eine Ansammlung von Marketing-Schlagwörtern, sondern Sie erfahren oft auch, was momentan im Web »in« ist und was nicht.

Der Begriff »DHTML« fällt in beide Kategorien: Das Thema ist »in«, aber nichts anderes als ein Marketing-Schlagwort. Viele Leute denken, bei DHTML handle es sich:

**Begriffs-
bestimmung**

- ▶ um eine eigene Programmiersprache,
- ▶ um eine Erweiterung von HTML,
- ▶ um einen W3C-Standard.

Das ist alles falsch.

Bei vielen Bewerbungen findet man in der Rubrik »Programmiersprachen, die ich beherrsche«, auch den Eintrag DHTML. Eigentlich ist es falsch, das so zu schreiben – denn DHTML ist alles Mögliche, nur keine Programmiersprache. Auf der anderen Seite ist es aber genau richtig, auf seine DHTML-Kenntnisse zu verweisen, denn DHTML ist »in« und gefragt.

Doch was genau ist dieses sagenumwobene DHTML?

Wofür das Akronym steht, ist klar: Dynamic HTML, also dynamisches HTML. Eine genaue Definition dessen, was sich dahinter eigentlich verbirgt, fällt aber schwer und – je nach Quelle – anders aus.

Die wohl zutreffendste Definition von DHTML ist ungefähr folgende: DHTML ist ein Kunstwort, das die Möglichkeiten des Zusammenspiels von HTML, CSS und JavaScript bei Browsern ab Versionsnummer 4 unter einem Oberbegriff zusammenfassen will.

**Definitions-
versuch**

Mit »DHTML« (und das ist das letzte Mal, dass dieser Begriff in Anführungszeichen gesetzt wird, wie es sich eigentlich gehört) können Sie mit den einzelnen Elementen auf einer Webseite viele Dinge anstellen, beispielsweise deren Position verändern und sie ein- und ausblenden.

In weiten Teilen dieses Buchs wurde davon geredet, wie die eine oder andere Aufgabenstellung für Browser **ab** Versionsnummer 4 umgesetzt werden kann. In Sachen DHTML gibt es jedoch ein großes Problem: Die Browser **mit** der Versionsnummer 4.

Browser der Version 4

Im Jahre 1997 erschienen kurz hintereinander der Netscape Navigator 4 und der Internet Explorer 4. Beide Browser beherrschten DHTML – zumindest **ihre Interpretation** von DHTML. Und die beiden DHTML-Versionen sind zueinander hochgradig inkompatibel. Was also auf dem Internet Explorer funktioniert, schlägt beim Netscape Navigator 4 fehl und vice versa.

W3C-Standard

Das W3C (World Wide Web Consortium) hat daraufhin einen eigenen Standard entwickelt, um dem Wust an Implementierungen Einhalt zu gebieten. Der Internet Explorer 5 von Microsoft war der erste Browser, der das **W3C DOM** zumindest teilweise unterstützt hat. In den nachfolgenden Versionen (5.01 nebst zwei Service Packs, 5.5 nebst zwei Service Packs sowie 6) wurde die Unterstützung immer weiter verbessert. Der Opera-Browser ab Version 4 unterstützt das W3C DOM auch, wobei Version 5 wohl eine weitere Verbreitung findet (da werbefinanziert und kostenlos), und der Netscape 6 ist Spitzenreiter in Sachen W3C-DOM-Unterstützung.

Wenn Sie also die Browser mit der Versionsnummer 4 ignorieren, haben Sie relativ leichtes Spiel. Doch leider ist das nicht möglich, und der Hauptgrund dafür ist der Netscape Navigator 4.x, der immer noch weit verbreitet ist. Für diesen Browser müssen Sie sich also etwas Besonderes einfallen lassen.

Skripte für Netscape 6 anpassen

Auch bei der Anpassung von Skripten, die Sie vor Urzeiten einmal für den Netscape Navigator 4 und den Internet Explorer 4 »optimiert« haben, müssen Sie aufpassen. Diese Skripte funktionieren in der Regel auch im Internet Explorer 5 und 6, weil diese zu den Vorgängerversionen abwärtskompatibel sind. Diese Skripte funktionieren aber nicht im Netscape 6, denn dieser ist dem Internet Explorer 5 nur in Bezug auf dessen neue Möglichkeiten ähnlich, nicht aber in Bezug auf die Relikte des Internet Explorer 4.

Dieses Kapitel ist viergeteilt:

- ▶ Zunächst werden kurz die drei verschiedenen Ansätze vorgestellt.
- ▶ Dann sehen Sie, wie Sie die Browser voneinander unterscheiden und Ihre DHTML-Seiten weitestgehend browserunabhängig gestalten können.
- ▶ Im Anschluss erstellen wir einige Hilfsfunktionen, mit denen Sie browserunabhängig auf Elemente der HTML-Seite zugreifen können.
- ▶ Zuletzt zeigen wir Ihnen noch einige Beispiele.

7.1 Browserabhängiges DHTML

Wie zuvor schon erläutert wurde, gibt es prinzipiell drei verschiedene Ansätze: Den des Netscape 4, den des Internet Explorer 4 sowie den des W3C. Alle drei sollen an dieser Stelle noch einmal kurz rekapituliert werden.

7.1.1 Der Ansatz von Netscape

Im Netscape Navigator 4 wurde ein neues Tag eingeführt: `<layer>`. Die Halbwertszeit dieses Tags war relativ kurz: Weder hat Microsoft es in seinen Internet Explorer übernommen, noch wird es vom Netscape-eigenen Nachfolger, dem Netscape 6, unterstützt.

Das JavaScript-Gegenstück dazu ist das Objekt `document.layers`, welches einen Zugriff auf das `<layer>`-Objekt ermöglicht. Sie haben folgende Zugriffsmöglichkeiten:

`document.layers`

- ▶ `document.layers[Layer-Index]`
- ▶ `document.layers["Layer-ID"]`
- ▶ `document.Layer-ID`

Um eines gleich vorwegzunehmen: Anstelle des `<layer>`-Tags sollten Sie das `<div>`-Tag verwenden. Von der Funktionalität und dem JavaScript-Zugriff her ändert sich nichts, aber dafür gehen Sie einen ersten, wichtigen Schritt in Richtung Kompatibilität mit dem Internet Explorer und in Richtung W3C.

Betrachten Sie als Beispiel den folgenden Layer:

```
<div id="Test" style="position: absolute;">Test</div>
```


Sie können auf diesen Layer wie folgt zugreifen:

- ▶ `document.layers[0]` (unter der Prämisse, dass dies das erste `<div>`-Element auf der Seite ist),
- ▶ `document.layers["Test"]` (Sie verwenden also das `id`-Attribut),
- ▶ `document.Test` (die Kurzvariante zur obigen Art des Zugriffs).

In den Layer schreiben

Das `Layer`-Objekt verhält sich ähnlich wie ein `window`-Objekt. So hat es beispielsweise die Eigenschaft `document`, mit der auf den Inhalt des Layers zugegriffen werden kann. Um also den Inhalt des oben definierten Layers zu ändern, können Sie folgenden Code verwenden:

```
with (document.layers["Test"].document) {  
    open();  
    write("Neuer Text!");  
    close();  
}
```

Die Verwendung der `open()`-Methode ist dabei nicht notwendig, dafür aber das `close()` – vorher wird nichts angezeigt.

Beachten Sie auch, dass der Zugriff auf Elemente innerhalb eines `<div>`-Containers nur über das `Layer`-Objekt möglich ist. Ein Formular innerhalb des Layers kann nur wie folgt angesprochen werden:

```
document.layers["Test"].document.forms["xxx"]
```

Der folgende Aufruf würde fehlschlagen:

```
document.forms["xxx"]
```

Warum? Nun, das Hauptdokument enthält kein Formular, der Layer schon.

Sie können beim Netscape Navigator 4 nur auf Layer zugreifen und diese in ihrer Gesamtheit verschieben sowie sichtbar und unsichtbar machen. Es ist jedoch nicht möglich, auf die Eigenschaften anderer Elemente zuzugreifen. So ist es beispielsweise nicht möglich, die Farbe eines HTML-Elements zu ändern. Dazu müsste die `class`-Eigenschaft des Elements geändert werden, doch das ist mit dem Netscape Navigator 4 nicht möglich.

Farbe ändern

Um die Farbe eines Textes zu ändern, gibt es nur eine Möglichkeit:

- ▶ Setzen Sie den Text in einen Layer.
- ▶ Erstellen Sie parallel dazu einen weiteren Layer, der denselben Text in einer anderen Farbe enthält, und machen Sie diesen Layer unsichtbar.

- Wenn sich die Farbe des Textes ändern soll, müssen Sie den alten Layer unsichtbar und den neuen Layer sichtbar machen. Damit erzielen Sie den gewünschten Effekt: Der Text erscheint (auf wundersame Art und Weise) in einer anderen Farbe.

Sie sehen: Die Netscape-Möglichkeiten sind beschränkt. Wenn das nur das einzige Problem wäre ... Das Hauptproblem ist jedoch: Die Ansätze von Microsoft und dem W3C sind hiervon grundsätzlich verschieden.

Und noch eine Einschränkung: Nur absolut positionierte Elemente (`style="position: absolute;"`) können von JavaScript angesprochen werden. Bei relativer Positionierung finden Sie die angegebenen Layer nicht in `document.layers`, oder einige wichtige Funktionen und Eigenschaften fehlen.

Absolute Positionierung

7.1.2 Der Ansatz von Microsoft

Ab dem Internet Explorer 4 bestand die Möglichkeit, auf fast alle Elemente der Website zuzugreifen. Es war also eine nahe liegende Idee, den Zugriff auf alle Elemente über ein Objekt namens `document.all` zu ermöglichen (zumindest wird das als Grund für die Namensgebung angegeben).

Sie benötigen zum Zugriff das `id`-Attribut des Elements. Wir verwenden erneut das `<div>`-Element aus dem vorhergehenden Abschnitt:

id-Attribut

```
<div id="Test" style="position: absolute;">Test</div>
```

Der Zugriff ist auf mehrere Arten möglich:

- `document.all[Index]` (Zählung beginnt wie immer bei 0),
- `document.all["id-Attribut"]` (bevorzugte Möglichkeit),
- `document.all.id-Attribut`,
- `id-Attribut` (Sie haben richtig gesehen – als Eigenschaft des `window`-Objekts; gleichbedeutend mit dem `window.id-Attribut`).

Um eine gewisse Konsistenz beizubehalten, sollten Sie `document.all["id-Attribut"]` verwenden, dann kommen Sie bei Namenskonflikten nicht durcheinander. Stellen Sie sich beispielsweise vor, Sie haben ein `id-Attribut` namens »Galileo-Press«. Der Zugriff über `document.all["Galileo-Press"]` funktioniert ohne Probleme, `document.all.Galileo-Press` dagegen schlägt (natürlich) fehl.

Um den Text eines Elements auf der HTML-Seite zu verändern, müssen Sie nicht wie bei Netscape umständlich mit `document.open()/write()`

Text schreiben

`close()` operieren. Verwenden Sie einfach die Eigenschaft `innerHTML`, mit der Sie auf den HTML-Code innerhalb des Elements zugreifen können (mit `innerText` können Sie übrigens auf den reinen Text zugreifen):

```
document.all["Test"].innerHTML = "Neuer Text!";
```

Farbe ändern

Kommen wir nun zu einem der offensichtlichsten Vorteile des Microsoft-Modells: Um beispielsweise die Farbe eines Textes zu ändern, müssen Sie nicht (wie bei Netscape) mit verschiedenen Layern operieren. Über die `style`-Eigenschaft des HTML-Dokuments können Sie direkt auf die wichtigsten Attribute zugreifen, unter anderem auch auf die Farbe:

```
document.all["Test"].style.color = rgb(0, 0, 255);
```

Obiger Code färbt den Text im Element `<div id="Test">` blau ein. Die Schreibweise der Farbe entspricht der, die Sie in einem Stylesheet verwenden würden. Das deutet bereits der Name der Eigenschaft `style` an: Damit können Sie Stil-Eigenschaften des entsprechenden Elements auslesen und setzen.



Ein Hinweis zum Auslesen der `style`-Eigenschaften: Mit `<style>` gesetzte Werte können nicht ausgelesen werden. Um `style`-Eigenschaften auslesen zu können, müssen sie auf eine der beiden Arten gesetzt worden sein:

- ▶ ebenfalls mit JavaScript,
- ▶ über die `style`-Eigenschaft des Elements.

Der Internet Explorer 4 war der später verabschiedeten W3C-Empfehlung sehr nahe, wenngleich einige wichtige Funktionalitäten fehlten. Mit dem Internet Explorer 5 und dem Netscape 6 änderte sich das, die Browser bieten nun einen einigermaßen einheitlichen Zugriff auf die Elemente einer HTML-Seite.

7.1.3 Der Ansatz des W3C

Ab Browserversion 5, das heißt Internet Explorer 5, Opera 5 und Netscape 6 (denn vom Netscape-Browser gab es keine Version 5 und wird es mit ziemlicher Sicherheit auch nie eine geben), ändert sich der Zugriff auf das DOM (Document Object Model) des Browsers. Dazu gibt es eine Methode `getElementById()` des `document`-Objekts, mit dem auf alle Elemente einer HTML-Seite zugegriffen werden kann.

Nehmen wir als Beispiel ein letztes Mal erneut den `<div>`-Container:

```
<div id="Test" style="position: absolute;">Test</div>
```

Sie erhalten über `document.getElementById("Test")` eine Referenz auf diesen Container und können so dessen Eigenschaften verändern.

Um den Inhalt zu verändern, können Sie erneut die Eigenschaft `innerHTML` verwenden, die Sie schon aus dem vorhergehenden Abschnitt kennen:

Text schreiben

```
document.getElementById("Test").innerHTML = "Neuer Text!";
```

Der DOM-Entwurf des W3C sieht eine Baumstruktur für die Elemente einer Website vor. Es ist mit JavaScript möglich, innerhalb dieses Baumes die einzelnen Verzweigungen zu verfolgen. Näheres dazu finden Sie in der JavaScript-Referenz von Galileo Press; für browserunabhängiges Programmieren ist diese Vorgehensweise nicht weiter interessant.



Neben dem HTML-Inhalt können Sie auch Stil-Eigenschaften der diversen Elemente verändern, beispielsweise die Farbe:

Farbe ändern

```
document.getElementById("Test").style.color = rgb(0,0,255);
```

Wenn Sie exzessiv mit Stylesheets arbeiten, können Sie auch das `class`-Attribut des entsprechenden HTML-Elements mit JavaScript ändern:

```
document.getElementById("Test").className = "Klassenname";
```

Doch nun genug der browserspezifischen Theorie, wenden wir uns dem Hauptproblem zu: Wie kriegen Sie es hin, dass sich alle Browser (weitgehend) gleich verhalten?

7.2 Browserunabhängiges DHTML

Fassen wir zusammen:

- ▶ Beim Netscape Navigator funktioniert der Zugriff auf HTML-Elemente über `document.layers`.
- ▶ Beim Internet Explorer ab Version 4 verwenden Sie `document.all`.
- ▶ Beim Internet Explorer ab Version 5 verwenden Sie `document.getElementById()`.
- ▶ Beim Netscape 6 verwenden Sie `document.getElementById()`.

Das »Hauptproblem« beim Festlegen einer Strategie für browserunabhängiges DHTML liegt nun im Internet Explorer ab Version 5. Dieser Browser beherrscht zwar (größtenteils) das DOM des W3C, aber auch noch die Zugriffsart à la Internet Explorer 4.

Sie müssen sich also überlegen, wie Sie die neueren Internet-Explorer-Versionen behandeln. Trauen Sie dem Microsoft-Produkt eine weitestgehende Kompatibilität zum Netscape 6 zu – oder nicht.

In der Praxis wird meist der Weg gewählt, dass alle Internet-Explorer-Versionen gleich behandelt werden. Da die »IEs« zueinander weitestgehend kompatibel sind, können Sie so mit demselben Code alle Versionen des Internet Explorers zufrieden stellen. Für den Netscape 6 (und den Opera 5) verwenden Sie dann den DOM-Code und können so auf Netscape-only-Bugs gezielter eingehen.

Um dies zu realisieren, müssen Sie eine Abfrage nach folgendem Muster einsetzen:

```
if (document.layers) { // Netscape Navigator 4
    // Code für Netscape Navigator 4.x
} else if (document.all) { // Internet Explorer ab Version 4
    // Code für den Internet Explorer ab Version 4
} else if (document.getElementById) { // Netscape 6
    // Code für den Netscape 6 und andere, DOM-kompatible Browser
}
```

Auf den Internet Explorer ab Version 5 passt zwar auch die Abfrage `if (document.getElementById)`, aber eben auch `if (document.all)`, weswegen der Code für den Internet Explorer 4 ausgeführt wird.

Wenn Sie den Internet Explorer Version 5 und höher den DOM-kompatiblen Browsern zuordnen möchten, müssen Sie umstellen:

```
if (document.layers) { // Netscape Navigator 4
    // Code für Netscape Navigator 4.x
} else if (document.getElementById) { // Netscape 6, IE 5
    // Code für Netscape 6 und Internet Explorer ab Version 5
} else if (document.all) { // Internet Explorer 4
    // Code für Internet Explorer 4
}
```

Wie bereits erwähnt wurde, verwenden wir die erstere Methode der Unterscheidung, alle Internet-Explorer-Versionen werden also gleich behandelt.

Sie haben nun mehrere Möglichkeiten. Die einfachste besteht wohl darin, für jeden Browsertyp eine spezielle Version der Webseite herzustellen und die Benutzer direkt dorthin weiterzuleiten.

Bei DHTML-fähigen Browsern können Sie im Übrigen die Funktion `location.replace()` zur Weiterleitung verwenden. Im Gegensatz zum direkten Setzen von `location.href` bietet diese Funktion den Vorteil, dass die vorherige URL in der History (Verlaufsliste) des Browsers überschrieben wird und deswegen nicht mit der Schaltfläche **Zurück** (oder **Back**) erreicht werden kann.

Ebenso sollten Sie Browser ohne oder mit deaktiviertem JavaScript berücksichtigen. Folgender HTML-Code versucht, es jedem Browser recht zu machen:

```
<html>
<head>
<title>Unsere Homepage</title>
<script language="JavaScript"><!--
if (document.layers) // Netscape Navigator 4
    location.replace("nn4.html");
else if (document.all) // Internet Explorer ab 4
    location.replace("ie.html");
else if (document.getElementById) // Netscape 6
    location.replace("n6.html");
else // Browser, die zumindest ein wenig JavaScript beherrschen
    if (location.replace)
        location.replace("js.html");
else
    location.href = "js.html";
//--></script>
</head>
<body>
<h1>Willkommen auf unserer Website!</h1>
Sie betrachten gerade die Version
ohne jegliche Skriptunterstützung
</body>
</html>
```

Listing 71 Umleitung je nach DHTML-Fähigkeiten des Browsers

Beim Aufrufen dieser Seite – auch auf der CD-ROM, unter dem Dateinamen **homepage-1.html** – passiert Folgendes:

- ▶ Benutzer des Netscape Navigator 4.x werden auf die Seite **n4.html** weitergeleitet.
- ▶ Benutzer des Internet Explorers ab Version 4 werden auf die Seite **ie.html** weitergeleitet.
- ▶ Benutzer des Netscape 6 (und des Opera 5) werden auf die Seite **n6.html** weitergeleitet.
- ▶ Alle anderen JavaScript-fähigen Browser werden auf die Seite **js.html** weitergeleitet.
- ▶ Alle restlichen Browser verweilen auf der aktuellen Seite.

In der Praxis ist dieses Vorgehen jedoch ein wenig unrentabel, da der Verwaltungsaufwand für mehrere einzelne Seiten natürlich größer ist, als wenn Sie eine einzige (aber dafür browserunabhängige) Seite haben.

Den JavaScript-Code müssen Sie freilich für alle Browsertypen anpassen, das Problem sind eher die restlichen Inhalte der Seite. Ändert sich beispielsweise ein einzelner Absatz, so müssen Sie diesen bei der Verwendung mehrerer unterschiedlicher Seiten mehrfach ändern, bei einer einzigen, universellen Seite natürlich nur einmal.

Windows-Plattform

Sie sollten aber unter Umständen in Erwägung ziehen, DHTML-Effekte auf die Windows-Plattform zu beschränken. Insbesondere die Macintosh-Versionen der verschiedenen Browser überbieten sich gegenseitig in der Einführung neuer Bugs (wobei die Internet-Explorer-Versionen unter Macintosh mehr Bugs zu haben scheinen als die Netscape-Versionen unter Macintosh, warum auch immer). Unter Linux scheint die Situation ein wenig besser zu sein. Es ist also eine gute Idee, speziell Macintosh-Benutzer ebenfalls auf eine Schmalspurvariante der Seite umzuleiten:

```
<html>
<head>
<title>Unsere Homepage</title>
<script language="JavaScript"><!--
var mac = navigator.platform.toLowerCase().indexOf("mac");
if (document.layers && mac == -1) // Netscape Navigator 4
    location.replace("nn4.html");
else if (document.all && mac == -1) //Internet Explorer ab 4
    location.replace("ie.html");
else if (document.getElementById && mac == -1) // Netscape 6
    location.replace("n6.html");
else // Browser, die zumindest ein wenig JavaScript beherrschen
    if (location.replace)
```

```

        location.replace("js.html");
    else
        location.href = "js.html";
    //--></script>
</head>
<body>
<h1>Willkommen auf unserer Website!</h1>
Sie betrachten gerade die Version
ohne jegliche Skriptunterstützung
</body>
</html>

```

Listing 7.2 Browsererkennung inklusive Ausschluss von Macs

Dieses Listing finden Sie auf der CD-ROM zum Buch unter dem Dateinamen **homepage-2.html**.



Wenn Sie nun allen Umständen zum Trotz für verschiedene Browser verschiedene Versionen Ihrer Website erstellen möchten, können Sie zum nächsten Kapitel weiterblättern. Wollen Sie allerdings jede einzelne Seite Ihres Angebots browserunabhängig gestalten, lesen Sie hier weiter.

Wir wollen im Folgenden zwei Beispiele browserunabhängig umsetzen.

7.3 Beispiel: Fliegendes Firmenlogo

Vielleicht haben Sie diesen Effekt schon einmal gesehen: Sie gehen auf eine Website, und ein Firmenlogo (oder irgendeine andere Grafik) kommt seitlich hereingeschwebt und nimmt dann schließlich ihre finale Position auf der Seite ein.

Der Code hierzu wird in zwei Schritten erstellt:

- ▶ Zunächst wird das Firmenlogo in einen `<div>`-Container gelegt, damit es auch browserunabhängig verschoben werden kann.
- ▶ Dann wird der Code eingefügt, der die Position des Logos kontinuierlich verändert. Dieser wird für alle drei »großen« Browser – Internet Explorer, Netscape Navigator 4, Netscape 6 – umgesetzt.

Hier zunächst der `<div>`-Container; achten Sie auf die absolute Positionierung:

```

<div id="logo" style="position: absolute">
    
</div>

```


Nun benötigen wir noch einige Hilfsfunktionen. Es ist mit JavaScript möglich, die Position des `<div>`-Containers zu verändern. Genauer gesagt, die x- und die y-Koordinate der linken oberen Ecke können gesetzt werden. Je nach Browsertyp geht das anders vonstatten:

- beim Internet Explorer über die Eigenschaften `style.posLeft` (x-Koordinate) und `style.posTop` (y-Koordinate),
- beim Netscape Navigator 4.x über die Eigenschaften `left` (x-Koordinate) und `top` (y-Koordinate),
- beim Netscape 6 über die Eigenschaften `style.left` (x-Koordinate) und `style.top` (y-Koordinate).



Die linke obere Ecke hat dabei die Koordinate (0, 0).

Außerdem gilt es zu beachten, dass der Netscape 6 keinen Pixelwert zurückliefert, sondern den Wert und die dazugehörige Einheit, beispielsweise »10px« für 10 Pixel. Hier müssen Sie also mit `parseInt()` den wahren Wert ermitteln.

Die folgende Hilfsfunktion ermittelt für jeden der drei Browsertypen die x-Koordinate; als Parameter wird der `id`-Wert des Layers übergeben:

```
function lese_x(id) {
    if (document.all)
        return document.all(id).style.posLeft;
    else if (document.layers)
        return document.layers[id].left;
    else if (document.getElementById)
        return parseInt(document.getElementById(id).style.left);
}
```

Analog lautet der Code für die y-Koordinate:

```
function lese_y(id) {
    if (document.all)
        return document.all(id).style.posTop;
    else if (document.layers)
        return document.layers[id].top;
    else if (document.getElementById)
        return parseInt(document.getElementById(id).style.top);
}
```

Auf dieser Basis ist es ein Leichtes, Funktionen zu schreiben, welche die x- und die y-Koordinate eines Layers setzen können. Als Parameter werden die ID des Layers sowie der gewünschte Wert übergeben:

```
function setze_x(id, wert) {
    if (document.all)
        document.all(id).style.posLeft = wert;
    else if (document.layers)
        document.layers[id].left = wert;
    else if (document.getElementById)
        document.getElementById(id).style.left = wert + "px";
}
```

```
function setze_y(id, wert) {
    if (document.all)
        document.all(id).style.posTop = wert;
    else if (document.layers)
        document.layers[id].top = wert;
    else if (document.getElementById)
        document.getElementById(id).style.top = wert + "px";
}
```

Jetzt ist es nicht mehr sehr aufwändig, das animierte Firmenlogo zu realisieren:

- ▶ Zunächst wird das Logo außerhalb des sichtbaren Bereichs gesetzt (beispielsweise negative x-Koordinate).
- ▶ Dann wird per Timeouts das Logo immer mehr in Richtung des sichtbaren Bereichs bewegt.
- ▶ Sobald das Logo vollkommen sichtbar ist (positive x-Koordinate oder noch größer, je nachdem, welcher Abstand zum linken Rand gewünscht wird), wird die Animation gestoppt.

Zunächst müssen Sie also beim Laden der Seite das Logo verschwinden lassen:

```
window.onload = init;

function init() {
    // Logo "verstecken"
    setze_x("logo", -200);
}
```

```

    // Animation starten
    animate();
}

```

In der Funktion `animate()` wird dann die Logo-Position kontinuierlich verändert/angepasst:

```

function animate() {
    if (leselinks("logo") < 0) {
        setzelinks("logo", leselinks("logo") + 5);
        setTimeout("animate()", 50);
    }
}

```

Das war dann aber auch schon alles. Wenn Sie die Seite in den Browser laden, schwebt das Logo – aktivierte JavaScript-Unterstützung vorausgesetzt – von links in den sichtbaren Bereich hinein. Ist dagegen JavaScript deaktiviert, so passiert gar nichts; da wir allerdings das Logo erst per JavaScript aus dem sichtbaren Bereich verschwinden lassen, befindet sich das Logo bei Benutzern ohne JavaScript-Unterstützung trotzdem an der richtigen Stelle.

Hier der komplette Code – auf der CD-ROM zum Buch unter dem Dateinamen **logo-1.html**:

```

<html>
<head>
<title>Animiertes Logo</title>
<script type="text/javascript"><!--
function lese_x(id) {
    if (document.all)
        return document.all(id).style.posLeft;
    else if (document.layers)
        return document.layers[id].left;
    else if (document.getElementById)
        return parseInt(document.getElementById(id).style.left);
}

function setze_x(id, wert) {
    if (document.all)
        document.all(id).style.posLeft = wert;
    else if (document.layers)
        document.layers[id].left = wert;
    else if (document.getElementById)

```

```

        document.getElementById(id).style.left = wert + "px";
    }

    window.onload = init;

    function init() {
        // Logo "verstecken"
        setze_x("logo", -200);
        // Animation starten
        animate();
    }

    function animate() {
        if (lese_x("logo") < 0) {
            setze_x("logo", lese_x("logo") + 5);
            setTimeout("animate()", 50);
        }
    }
}
//--></script>
<body>
<div id="logo" style="position: absolute">
    
</div>
</body>
</html>

```

Listing 7.3 Die x-Koordinate des Layers wird angepasst.

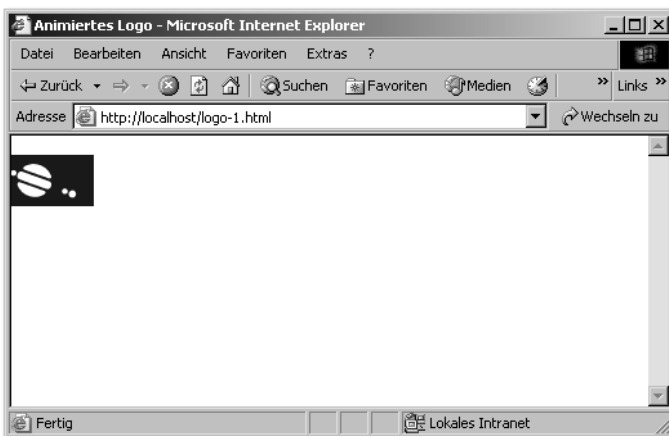


Abbildung 7.1 Das Logo schwebt von links herein.

Wir haben im vorhergehenden Listing lediglich die Hilfsfunktionen `lese_x()` und `setze_x()` verwendet; wenn Sie `lese_y()` und `setze_y()` einsetzen möchten, kann das Logo beispielsweise von oben einschweben. Dazu müssen Sie lediglich die Funktionen `init()` und `animate()` leicht anpassen:

```
function init() {
    // Logo "verstecken"
    setze_y("logo", -150);
    // Animation starten
    animate();
}

function animate() {
    if (lese_y("logo") < 0) {
        setze_y("logo", lese_y("logo") + 5);
        setTimeout("animate()", 50);
    }
}
```

Auf der CD-ROM zum Buch finden Sie das komplette Listing unter dem Dateinamen **logo-2.html**. Um das Ganze aber auch einmal am Stück zu sehen, finden Sie es zusätzlich hier abgedruckt:

```
<html>
<head>
<title>Animiertes Logo</title>
<script type="text/javascript"><!--
function lese_y(id) {
    if (document.all)
        return document.all(id).style.posTop;
    else if (document.layers)
        return document.layers[id].top;
    else if (document.getElementById)
        return parseInt(document.getElementById(id).style.top);
}

function setze_y(id, wert) {
    if (document.all)
        document.all(id).style.posTop = wert;
    else if (document.layers)
        document.layers[id].top = wert;
```

```

        else if (document.getElementById)
            document.getElementById(id).style.top = wert + "px";
    }

window.onload = init;

function init() {
    // Logo "verstecken"
    setze_y("logo", -150);
    // Animation starten
    animate();
}

function animate() {
    if (lese_y("logo") < 0) {
        setze_y("logo", lese_y("logo") + 5);
        setTimeout("animate()", 50);
    }
}
//--></script>
<body>
<div id="logo" style="position: absolute">
    
</div>
</body>
</html>

```

Listing 7.4 Die y-Koordinate des Layers wird angepasst.

Wie Sie Abbildung 7.2 entnehmen können, funktioniert das Beispiel tatsächlich (auch) im Netscape 6.



Abbildung 7.2 Das Logo schwebt von oben herein.

Sie können diese Techniken beliebig kombinieren, beispielsweise das Logo auch von links oben einschweben lassen.

7.4 Beispiel: Permanentes Werbebanner

Das nächste Beispiel beinhaltet ein Werbebanner. Damit soll mitnichten zum Ausdruck gebracht werden, DHTML sei nur etwas für die werbetreibende Zunft. Es ist nur nun einmal so, dass DHTML insbesondere bei Werbebannern häufig zum Einsatz kommt.

Betrachten Sie dazu Abbildung 7.3. Es handelt sich hierbei um eine willkürlich gewählte Seite auf Geocities, einem Gratis-Hoster. Dieser versucht sich unter anderem dadurch zu refinanzieren, dass bei Mitgliederseiten ein Werbebanner eingeblendet wird. Dieses platziert sich in der rechten oberen Ecke des Browserbildschirms. Bei einigen Versionen des Werbebanners geht das sogar so weit, dass sich das Banner **immer** in der rechten oberen Ecke befindet, auch wenn der Benutzer weiter nach unten scrollt. Das Banner bleibt immer sichtbar.

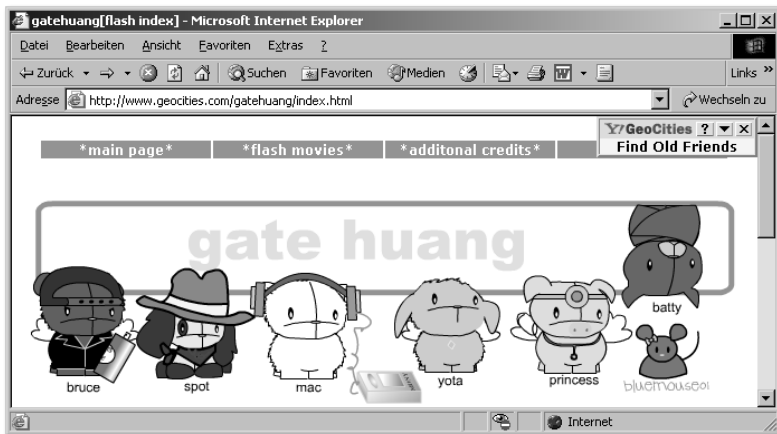


Abbildung 7.3 Rechts oben ist ein Werbebanner zu sehen.

Wir wollen diesen Effekt mit JavaScript nachbilden und dafür sorgen, dass das Galileo-Banner immer sichtbar bleibt.

Sie ahnen bereits – das ist wieder ein Fall für DHTML und absolut positionierte Layer. Wir beginnen wieder mit dem HTML-Code für den Layer:

```
<div id="banner" style="position:absolute">
  <a href="http://www.galileo-press.de/"
    target="_blank"
```

```

        onclick="unsichtbar('banner');"
    ></a>
</div>

```

Ebenso benötigen wir wieder Funktionen zum Setzen der Position des Banners, und zwar sowohl für die x- als auch für die y-Koordinate:

```

function setze_x(id, wert) {
    if (document.all)
        document.all(id).style.posLeft = wert;
    else if (document.layers)
        document.layers[id].left = wert;
    else if (document.getElementById)
        document.getElementById(id).style.left = wert + "px";
}

function setze_y(id, wert) {
    if (document.all)
        document.all(id).style.posTop = wert;
    else if (document.layers)
        document.layers[id].top = wert;
    else if (document.getElementById)
        document.getElementById(id).style.top = wert + "px";
}

```

Hauptproblem bei diesem Beispiel ist es zu ermitteln, an welche Position das Banner überhaupt bewegt werden soll.

Dazu müssen Sie zunächst die Breite des Browserfensters bestimmen, was sich je nach Browserhersteller (leider) etwas anders gestaltet:

- ▶ Beim Internet Explorer greifen Sie auf `document.body.clientWidth` zu.
- ▶ Beim Netscape Navigator 4.x verwenden Sie `window.innerWidth`.
- ▶ Im Netscape 6 schließlich benötigen Sie ebenfalls die Eigenschaft `window.innerWidth`.

Sie ermitteln damit die Breite des nutzbaren Bereichs des Browsers, also abzüglich Scrollbalken et cetera.



Damit ist es möglich, die Position des Banners zu ermitteln:

- ▶ Die x-Koordinate ist die Breite des Browsers abzüglich der Bannerbreite.
- ▶ Die y-Koordinate ist 0.

Allerdings haben wir hier einen Punkt übersehen – auf umfangreichen Seiten kann der Benutzer horizontal und vertikal scrollen. Damit sich das Banner auch dann noch rechts oben befindet, müssen diese Scrollweiten mit einberechnet werden. Auch dies ist wieder von Browser zu Browser unterschiedlich:

- Im Internet Explorer ermitteln Sie den nach links gescrollten Bereich mit `document.body.scrollLeft`, den nach unten gescrollten Bereich mit `document.body.scrollTop`.
- Der Netscape Navigator 4.x und der Netscape 6 verwenden beide `window.pageXOffset` und `window.pageYOffset`.

Beim Internet Explorer ist des Weiteren zu beachten, dass es hier im Vergleich zu den Netscape-Browsern leichte Messunterschiede gibt; Sie müssen absolut gesehen das Banner in Netscape-Browsern weiter links positionieren als im Internet Explorer.

Die Funktion `animate()` setzt dies alles um: Die gewünschten Koordinaten werden berechnet und dann mit `setze_x()` und `setze_y()` in die Tat umgesetzt:

```
function animate() {  
    if (document.layers){  
        var x = window.innerWidth+window.pageXOffset-100;  
        var y = window.pageYOffset;  
    } else if (document.all){  
        var x = document.body.clientWidth+document.body.scrollLeft-89;  
        var y = document.body.scrollTop;  
    } else if (document.getElementById){  
        var x = window.innerWidth+window.pageXOffset-100;  
        var y = window.pageYOffset;  
    }  
    setze_x("banner", x);  
    setze_y("banner", y);  
}
```

Diese Funktion wird per Intervall regelmäßig aufgerufen, damit die Bannerposition permanent aktualisiert werden kann:

```
var ID;  
window.onload = init;
```

```
function init() {
    ID = setInterval("animate()", 100);
}
```

Es ist wichtig, `setInterval()` erst nach dem **vollständigen** Laden der Seite aufzurufen; vorher haben Sie nämlich keinen Zugriff auf das Banner!



Wie Sie zuvor im HTML-Code gesehen haben, wird beim Anklicken des Banners nicht nur die Homepage von Galileo Press in einem neuen Fenster (`target="_blank"`) geöffnet, sondern auch noch eine JavaScript-Funktion `unsichtbar()` aufgerufen. In dieser Funktion wird das Werbebanner unsichtbar gemacht – der Benutzer hat mittlerweile sicher lange genug darauf gestartet.

Auch dies ist mit DHTML möglich. Sie müssen die Stil-Eigenschaft `visibility` auf `"hidden"` setzen, um den Layer verschwinden zu lassen.

Um den anderen Weg zu gehen, also ein Layer sichtbar zu machen, setzen Sie diese Eigenschaft auf `"visible"`.



Die Stil-Eigenschaften verbergen sich je nach Browserhersteller und Browserversion an einer anderen Stelle. Im Microsoft Internet Explorer haben Sie in allen Versionen ab 4 Zugriff mittels `document.all(...).style`, analog im Netscape 6, wo Sie `document.getElementById(...).style` verwenden können. Der Netscape Navigator 4.x fällt ein wenig aus der Reihe, hier sind die Stil-Eigenschaften direkt unterhalb von `document.layers[...]` untergebracht.

Damit ist die Funktion `unsichtbar()` schnell erstellt:

```
function unsichtbar(id) {
    if (document.all)
        document.all(id).style.visibility = "hidden";
    else if (document.layers)
        document.layers[id].visibility = "hidden";
    else if (document.getElementById)
        document.getElementById(id).style.visibility = "hidden";
    clearInterval(ID);
}
```

Beachten Sie die letzte Zeile – dort wird die Intervall-ID gelöscht, denn sobald das Banner unsichtbar ist, muss seine Position nicht mehr permanent aktualisiert werden.

Nachfolgend noch einmal das komplette Listing im Zusammenhang; auf der CD-ROM zum Buch finden Sie es unter dem Dateinamen **banner.html**.

```
<html>
<head>
<title>Animiertes Logo</title>
<script type="text/javascript"><!--
function setze_x(id, wert) {
    if (document.all)
        document.all(id).style.posLeft = wert;
    else if (document.layers)
        document.layers[id].left = wert;
    else if (document.getElementById)
        document.getElementById(id).style.left = wert + "px";
}

function setze_y(id, wert) {
    if (document.all)
        document.all(id).style.posTop = wert;
    else if (document.layers)
        document.layers[id].top = wert;
    else if (document.getElementById)
        document.getElementById(id).style.top = wert + "px";
}

function unsichtbar(id) {
    if (document.all)
        document.all(id).style.visibility = "hidden";
    else if (document.layers)
        document.layers[id].visibility = "hidden";
    else if (document.getElementById)
        document.getElementById(id).style.visibility = "hidden";
    clearInterval(ID);
}

var ID;
window.onload = init;

function init() {
    ID = setInterval("animate();", 100);
```

```

}

function animate() {
  if (document.layers){
    var x = window.innerWidth+window.pageXOffset-100;
    var y = window.pageYOffset;
  } else if (document.all){
    var x = document.body.clientWidth+document.body.scrollLeft-89;
    var y = document.body.scrollTop;
  } else if (document.getElementById){
    var x = window.innerWidth+window.pageXOffset-100;
    var y = window.pageYOffset;
  }
  setze_x("banner", x);
  setze_y("banner", y);
}
//--></script>
<body>
<div id="banner" style="position:absolute">
  <a href="http://www.galileo-press.de/"
    target="_blank"
    onclick="unsichtbar('banner');">
    </a>
</div>

<table width="1500" height="1500">
  <tr><td>Platzhalter :-)</td></tr>
</table>
</body>
</html>

```

Listing 7.5 Das Werbebanner befindet sich permanent rechts oben.

Damit der Effekt auch beobachtet werden kann, haben wir in den HTML-Code eine 1500 * 1500 Pixel große Tabelle eingefügt, die das Scrolling auch bei höheren Auflösungen ermöglicht. In Abbildung 7.2 sehen Sie das Resultat (diesmal im Netscape Navigator 4.x): Das Banner befindet sich stets in der rechten oberen Ecke.



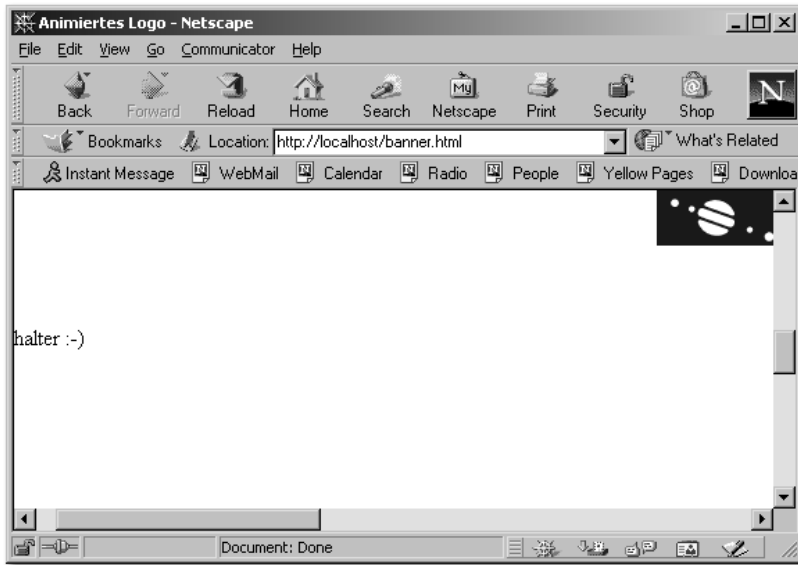


Abbildung 7.4 Das permanente Banner im Netscape Navigator 4

7.5 Fazit

Wir konnten natürlich in diesem Kapitel die wirklich umfassenden Möglichkeiten von DHTML nur anreißen. Es gibt noch viele weitere Einsatzbereiche, und wir hoffen, Ihr Interesse geweckt zu haben.

Es sei aber noch vermerkt, dass beim Einsatz von DHTML der Entwicklungsaufwand stark ansteigt – schließlich müssen Sie (zumindest zum Zeitpunkt der Entstehung dieses Buchs) parallel für drei Browser entwickeln. Hier kann es sich durchaus lohnen, auf eine browserunabhängige Lösung wie beispielsweise Flash zu setzen.

8 Basics

Aller Anfang ist leicht – oder wie es ist, Eulen nach Athen zu tragen
– frei nach Aristophanes

Sie haben sich dieses Buch gekauft (geliehen, schenken lassen, geklaut), um ActionScript zu lernen und herauszufinden, welch starkes Team ActionScript mit Flash bildet. Wir setzen deswegen Grundkenntnisse in Flash voraus. Dies ist das einzige Kapitel, in dem wir auch ein wenig über Flash reden. Wir wollen hier vor allem sicherstellen, dass Sie verstehen, was mit bestimmten Begriffen gemeint ist. Vielleicht sind Sie in Flash schon sehr fit, wissen aber mit den Begriffen Symbol und Instanz noch nicht so viel anzufangen.

8.1 Einsatzgebiete von ActionScript

Auch komplexe Animationen und Bewegungen lassen sich mit Flash sehr einfach realisieren. Dafür benötigt man erst einmal kein ActionScript. Interessanter wird es erst, wenn Sie dem Anwender Gelegenheit geben möchten, zu interagieren. Wenn Sie komplexere Anwendungen schaffen möchten, die in unterschiedlichen Situationen anders reagieren. Wenn Sie Informationen aus einer externen Quelle wie einem XML-Dokument einfügen möchten. In solchen Fällen benötigen Sie eine Programmiersprache – und das ist für Flash eben ActionScript.

Ein Programmierer sollte ActionScript lernen, weil er damit Flash beherrschen lernt. Warum aber sollte ein Designer ActionScript lernen? Hauptsächlich, weil es für anspruchsvollere und damit zeitgemäße Anwendungen nicht ausreicht, sich ein paar Aktionen aus dem **Aktionen**-Bedienfeld zusammenzuklicken. Die Grenzen für den Designer sind einfach viel zu eng, wenn er die Konzepte hinter der Sprache nicht versteht. Und – ActionScript ist einfach. Freuen Sie sich also auf schnelle Lernerfolge!

ActionScript gibt es in Flash schon länger, allerdings wurde erst in Flash 5 auf die Kompatibilität zum Standard ECMA-262 umgestellt. Dieser Standard ist derselbe Standard, von dem sich JavaScript ableitet. Die Vorteile liegen auf der Hand: Sie müssen nur einmal die Syntax lernen, um beide Sprachen zu beherrschen, und Sie werden sehen, dass auch in der Funktionalität und bei Problemlösungen beide Sprachen sehr ähnlich funktionieren. Dies macht sich dieses Buch zu Nutze, indem es Ihnen beide Sprachen in einem zeigt.

Neben diesen Gemeinsamkeiten gibt es allerdings auch noch viele Möglichkeiten, ActionScript und JavaScript gemeinsam einzusetzen. Dazu finden Sie ein dickes Praxiskapitel (Kapitel 17, ActionScript mit JavaScript) im Praxis-Teil.



Wir weisen in diesem Buch öfter auf die (oftmals fehlende) Kompatibilität zu Flash 4 hin. Neue Funktionen in Flash MX benennen wir ebenfalls explizit als solche. Sie können also mit Flash 4, 5 oder MX arbeiten, wobei wir Flash 4 nicht mehr empfehlen, da erst in Flash 5 die Modernisierung von ActionScript entsprechend ECMA-262 vorgenommen wurde, und die ist es, die ActionScript zu einer wirklich ernstzunehmenden Programmiersprache und Flash zu einer Anwendungssoftware gemacht hat.

8.2 Flash-Basics

Als Sie Flash zum ersten Mal aufgerufen haben, sind Ihnen sicherlich die Bühne und die Zeitleiste als Erstes ins Auge gesprungen. Die Arbeit mit den verschiedenen Elementen der Oberfläche (siehe Abbildung 8.1) ist nicht Teil dieses Buches, allerdings finden Sie in der Abbildung die Begrifflichkeiten, die wir verwenden werden.



Der Flash-Film besteht aus Bildern. Diese Bilder können entweder normale Bilder oder Schlüsselbilder sein. Nur für Schlüsselbilder ist der Einsatz von ActionScript möglich. Ein Bild heißt oft auch Frame. Vorsicht, dieses Frame hat nichts mit dem Begriff Frame für einen Teil des Browserfensters verwechseln.



Abbildung 8.1 Die Flash-Oberfläche

Ein kurzer Hinweis gilt dem Bereich außerhalb der Bühne. Wir haben diesen Bereich extra in die Beschriftung übernommen, weil er für einige Animationen mit ActionScript von Bedeutung ist. Dabei werden Elemente, meist Movieclip-Symbole, außerhalb der Bühne abgelegt und von dort dann in die Bühne gezogen oder dupliziert.



8.2.1 Windows und Mac

Die Screenshots dieses Buches entstanden unter Windows. Oh, werden eingefleischte Mac-Fans sagen, die Autoren sind Mac-Hasser. Ganz im Gegenteil, Grund war nur, dass sich Windows als Standard mit Laptops und verteilten Rechnern einfacher realisieren ließ.

Ein zweiter Grund, warum wir uns auf ein Betriebssystem beschränken, ist sehr einfach: Es gibt kaum Unterschiede. Die wenigen, die vorhanden sind, wollen wir kurz nennen.

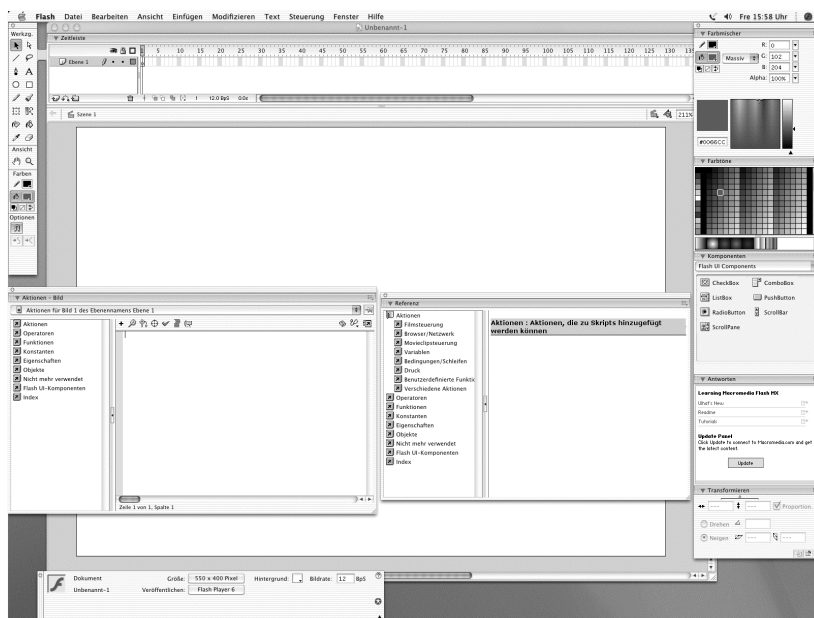


Abbildung 8.2 Flash unter MacOS X

Wenn Sie einen Blick auf Flash am Mac werfen (siehe Abbildung 8.2), sehen Sie genau dieselben Bedienfelder, dieselbe Bühne, den Eigenschafteninspektor, die Zeitleiste, kurz, viele alte Bekannte. Auch im Handling von ActionScript gibt es keine Unterschiede.

Eine Besonderheit in der Oberfläche gibt es allerdings doch. Unter MacOS X hat sich das Programmmenü als Standard eingebürgert. Diese Neuerung der Aqua-Oberfläche heißt bei Flash natürlich **Flash** (siehe Abbildung 8.3). Sie finden dort als wichtigste Einstellungen die **Voreinstellungen** und **Tastenkombinationen**, die vorher unter **Bearbeiten** verborgen waren. Wenn wir also im Verlauf dieses Kapitels und des Buches von **Voreinstellungen** im Menü **Bearbeiten** reden, wissen Aqua-Anhänger, dass sie im Programmmenü fündig werden.



Abbildung 8.3 Das Programmmenü Flash

Nun kommen wir noch zu einem etwas größeren Unterschied zwischen Mac und Windows, den Tastenkürzeln. Er fällt für ActionScript-Fans nicht so stark ins Gewicht. Für die gibt es, wie der Anhang zu Tastenkürzeln zeigt, sowieso nicht so viele. Dennoch seien hier die Unterschiede erwähnt:

- ▶ Die **[Strg]**-Taste heißt am Macintosh  (Apfel-Taste). Wenn Sie also **[Strg]** sehen, müssen Sie den netten Apfel/Propeller/Doppelkreuz drücken.
- ▶ Die **[Alt]**-Taste entspricht am Macintosh der , die bei optischen Vergleichen doch sehr an eine Badewanne erinnert.
- ▶ Das Kontextmenü erreichen Sie am Macintosh mit einer Ein-Tasten-Maus, indem Sie die **[Strg]**-Taste drücken und an die gewünschte Stelle mit der Maus klicken.
- ▶ Die **[Entf]**-Taste ist am Mac nur bei erweiterterem Tastaturlayout vorhanden. Hier arbeiten Sie stattdessen nur mit .

8.2.2 Koordinatensystem

Wenn Sie mit Elementen in ActionScript arbeiten, können Sie diese nicht mehr einfach irgendwohin ziehen. Stattdessen benötigen Sie exakte Koordinaten, wo sich die Elemente befinden.

Die wichtigsten Maße, die Größe der Bühne, legen Sie im Eigenschaftensinspektor fest, wenn Sie auf die Bühne geklickt haben. Alle Elemente werden dann mit Koordinaten relativ zum Ursprung der Bühne platziert.

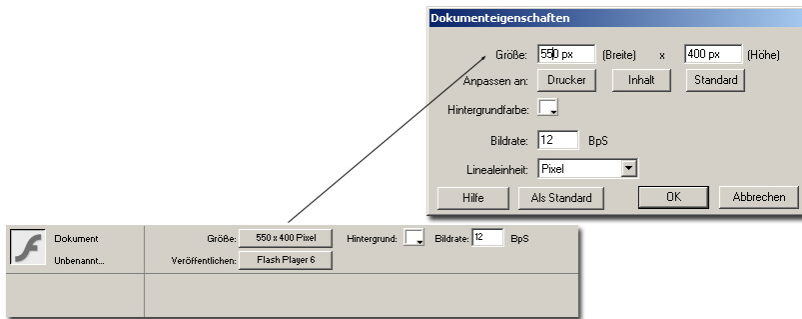


Abbildung 8.4 Die Größe ändern

Was sich am Anfang wie schwierige Zahlenschieberei anhört, ist bei näherer Betrachtung sehr einfach, wenn man ein paar Grundregeln befolgt:

- ▶ Das Koordinatensystem der Bühne hat seinen Ursprung in der linken oberen Ecke der Bühne (siehe Abbildung 8.5). Das Koordinatensystem sehen Sie, wenn Sie es mit dem Menübefehl **Ansicht • Lineale** einblenden. Aus den Linealen ziehen Sie übrigens auch die Hilfslinien auf.
- ▶ Die x-Achse verläuft horizontal vom Ursprung nach rechts (positive Werte). Elemente links außerhalb der Bühne erhalten eine negative x-Koordinate.
- ▶ Die y-Achse verläuft vertikal nach unten (positive Werte). Koordinaten für Elemente oberhalb der Bühne werden mit negativen Werten angegeben (siehe Abbildung 8.5).
- ▶ Jedes Element hat seinen Ursprungspunkt, der im Eigenschaftensinspektor als Koordinate angegeben wird, in der linken oberen Ecke eines (imaginären) Rechtecks um das Element.

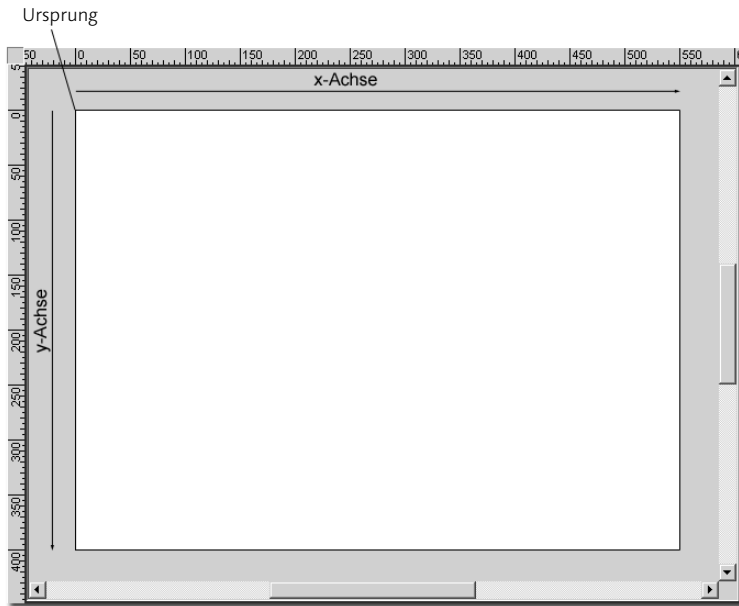


Abbildung 8.5 Das Koordinatensystem der Bühne



Die Kontur eines Elements – soweit an den Grenzen vorhanden – befindet sich zur Hälfte außer- und zur Hälfte innerhalb dieses imaginären Begrenzungsrechtecks.

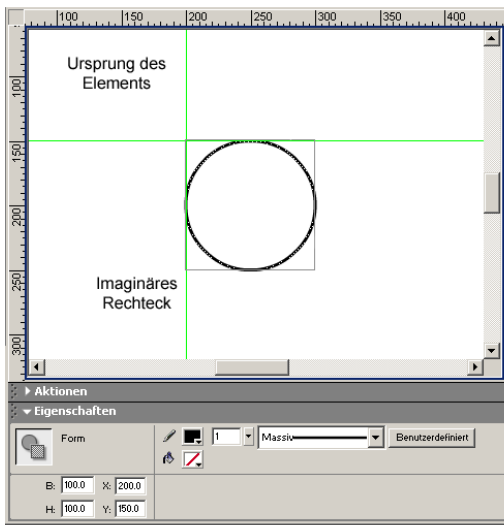


Abbildung 8.6 Ein Objekt mit seiner Position

- ▶ Ein Movieclip oder ein anderes Symbol hat ein eigenes Koordinatensystem, das nicht von der linken oberen Ecke, sondern von der Mitte ausgeht.
- ▶ Bei einem Symbol reichen positive Werte der x-Achse von der Mitte nach links, negative Werte von der Mitte nach rechts.
- ▶ Auf der y-Achse eines Symbols gehen positive Werte nach unten, negative nach oben.

Wenn Sie ein Element in ein Symbol konvertieren, hat es automatisch negative Koordinaten, da seine Mitte auch die Mitte des Movieclips bildet, die Koordinaten aber von der linken oberen Ecke berechnet werden. Um dies zu verhindern, können Sie allerdings einen anderen Registrierungspunkt wählen; wie das funktioniert, erfahren Sie im Abschnitt »Symbole und Instanzen«.

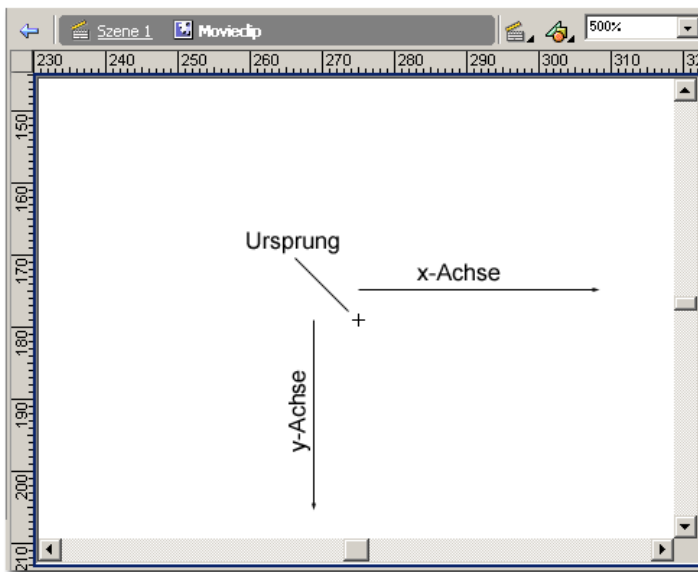


Abbildung 8.7 Das Koordinatensystem eines Movieclips

8.2.3 Ereignisse

ActionScript ist eine ereignisorientierte Programmiersprache. Was bedeutet das? ActionScript reagiert immer auf Ereignisse. Nur bei Ereignissen wird ActionScript-Code ausgeführt. Die Suche nach Ereignissen führt uns dazu, festzustellen, für welche Flash-Elemente wir ActionScript-Code einfügen können.

Man unterscheidet die Ereignisse in Flash wie folgt:

- ▶ ActionScript-Code in Schlüsselbildern. Das einzige Ereignis hierfür ist der Eintritt des Abspielkopfs in ein Schlüsselbild (auch Bildaktion). Es muss hier nicht explizit in den Code geschrieben werden. Stattdessen können Sie einfach das jeweilige Schlüsselbild in der Zeitleiste aktivieren und dann im **Aktionen**-Bedienfeld den Code einfügen.
- ▶ Ereignisse, die durch den Anwender veranlasst werden. Dies sind Ereignisse für Schaltflächen (man nennt sie auch Schaltflächenereignisse oder Objektaktionen). Sie werden mit dem Schlüsselwort `on` begonnen (`on` wird auch Event-Handler genannt).

```
on (Ereignis) {  
    Anweisungen;  
}
```

- ▶ Ereignisse bei Movieclips. Diese Ereignisse werden bei Instanzen eines Movieclips eingefügt und mit dem Schlüsselwort `onClipEvent` (auch Event-Handler) begonnen.

```
onClipEvent (Ereignis) {  
    Anweisungen;  
}
```



Die Ereignisse und Ereignisprozeduren für Movieclips finden Sie in Kapitel 12, Movieclips, in den Abschnitten »Ereignisse für Movieclips« und »Ereignisprozeduren«.

- ▶ Ereignisprozeduren reagieren auf Ereignisse, müssen aber nicht bei den Instanzen der Movieclips, Schaltflächen oder Textfelder eingefügt werden, sondern können von überall aus aufgerufen werden. Im Folgenden sehen Sie ein Beispiel für einen Movieclip

```
Movieclip.onload();
```

und eine Schaltfläche

```
Schaltfläche.onPress();
```



Ereignisprozeduren sind neu in Flash MX. Wie die Adressierung von Movieclips für Ereignisprozeduren funktioniert, erfahren Sie in Kapitel 12, Movieclips, im Abschnitt »Ereignisprozeduren«. Schaltflächen und Textfelder werden bei Ereignisprozeduren ebenso mit dem Movieclip adressiert, in dem sie enthalten sind. Folgende Zeile verweist beispielsweise

auf eine Schaltfläche, die sich in der Hauptzeitleiste des Flash-Films befindet (`_root`).

```
_root.Schaltfläche.onPress();
```

Schaltflächenereignisse

In der folgenden Tabelle finden Sie eine Sammlung aller wichtigen Schaltflächenereignisse.

Ereignis	Beschreibung
dragOut	Die linke Maustaste wird auf der Schaltfläche gedrückt, gedrückt gehalten und außerhalb der Schaltfläche gezogen.
dragOver	Anschlussereignis zu dragOut. Der Mauszeiger wird bei immer noch gedrückter linker Maustaste wieder zurück über die Schaltfläche gezogen.
keyPress "Taste"	Drücken einer Taste der Tastatur
press	Drücken der linken Maustaste auf der Schaltfläche
release	Loslassen der linken Maustaste Wird meist statt press eingesetzt, wenn für den Drücken -Zustand noch ein anderes Aussehen der Schaltfläche angezeigt werden soll
releaseOutside	Loslassen der linken Maustaste außerhalb der Schaltfläche. Wird oft eingesetzt, da bei schnellem Loslassen und gleichzeitigem Ziehen der Mauszeiger schon die Schaltfläche verlassen hat, obwohl sie noch angeklickt werden soll
rollOut	Der Mauszeiger verlässt die Schaltfläche.
rollOver	Der Mauszeiger überquert die Schaltfläche. Ruft den Zustand Darüber des Schaltflächen-Symbols auf.

Tabelle 8.1 Schaltflächenereignisse

Eine besondere Erwähnung ist das Ereignis `keyPress(Taste)` wert. Es wird aktiv, wenn die als Parameter übergebene Taste aktiv ist.

Die Taste kann als Buchstabe angegeben werden:

```
on (keyPress "c") {  
    trace("c");  
}
```

Alternativ gibt es Sonderbezeichnungen für besondere Tasten, beispielsweise `<Enter>` für .

```
on (keyPress "<Space>") {
    trace("Space");
}
```

Die möglichen Sonderbezeichnungen sehen Sie, wenn Sie im Aktionen-Bedienfeld die Codehinweise angeschaltet haben (siehe Abbildung 8.8).

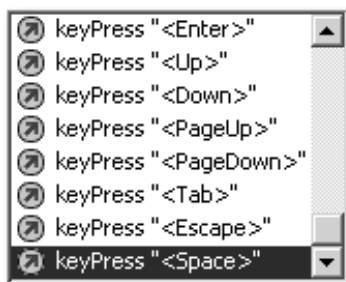


Abbildung 8.8 Die Sonderbezeichnungen



Eine Alternative zum Ereignis `keyPress` ist das Objekt `key`. Es hat den großen Vorteil, dass es nicht nur bei Schaltflächen erscheinen kann. Es wird mit dem Ereignis `keyDown` oder der Ereignisprozedur `onKeyDown` eingesetzt. Mit den Methoden `getCode` (liefert den Code der Taste) und `isDown` (prüft, ob die angegebene Taste gedrückt ist) erhalten Sie die aktuelle Taste. Die Eigenschaften des Objekts sind einige sehr wichtige Tasten als Konstanten. Alternativ werden die Tasten mit Tastencodewerten verwendet. Mehr zu Eigenschaften und Methoden finden Sie in Kapitel 11, Objekte, Eigenschaften und Methoden.

Ereignisprozeduren

Die Schaltflächen-Ereignisprozeduren setzen sich aus `on` und dem Ereignis zusammen. Für Schaltflächen ergeben sich daraus die Ereignisprozeduren `onDragOut`, `onDragOver`, `onPress` etc.

Ereignisprozeduren werden Funktionen übergeben, die Anweisungen erhalten, die beim Eintritt des Ereignisses ausgeführt werden.

```
Schaltfläche.onPress = function () {
    Anweisungen;
}
```

Die Möglichkeiten, die in Ereignisprozeduren liegen, sind außerordentlich. Auch in verschachtelten Filmen mit vielen Movieclips können Sie aus jedem Movieclip das Ereignis einer Schaltfläche oder eines Textfelds aufrufen.

Weitere Ereignisprozeduren werden Ihnen noch bei verschiedenen Themen begegnen. Eine gute Übersicht erhalten Sie im Index der Referenz unter [on](#).



8.2.4 Symbole und Instanzen

Eines der wichtigsten Konzepte von Flash sind die Symbole. Sie werden in der Bibliothek gesammelt. Die Bibliothek blenden Sie mit **F11** oder dem Menübefehl **Fenster • Bibliothek** ein.

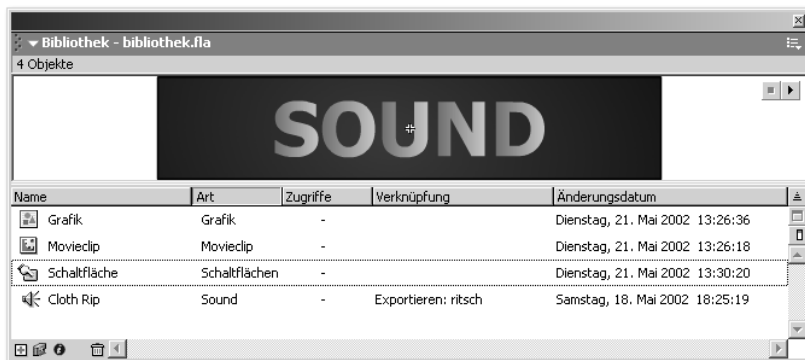


Abbildung 8.9 Die Bibliothek mit verschiedenen Symbolarten

Symbole haben einen Hauptvorteil: Sie werden einmal in der Bibliothek angelegt und können beliebig oft in den Film gezogen werden, um dort zum Einsatz zu kommen. Jede Version eines Symbols, die sich auf der Bühne befindet, wird als Instanz bezeichnet.

Eine Instanz kann einen eigenen Instanznamen erhalten, mit dem Sie per ActionScript auf das Symbol zugreifen können. Den Instanznamen vergeben Sie im Eigenschafteninspektor.

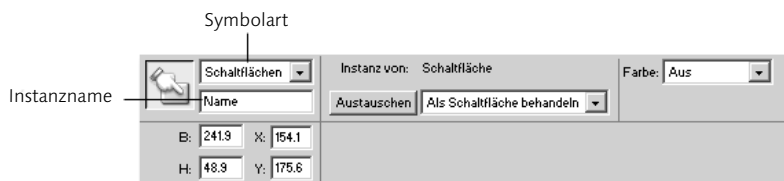


Abbildung 8.10 Den Instanznamen ändern

Eine Instanz kann unterschiedliche Eigenschaften haben. Sie können eine Instanz unabhängig vom Symbol rotieren lassen, umfärben und einiges

mehr. Dadurch spart Flash Speicherplatz. Es speichert einmal das Symbol, dann nur noch die geänderten Eigenschaften der einzelnen Instanzen.

In Flash gibt es natürlich nicht nur eine Art von Symbol. Wenn Sie ein neues Symbol erstellen, indem Sie Elemente in ein Symbol konvertieren (Kontextmenü: **In Symbol konvertieren**; Menü: **Einfügen • In Symbol konvertieren**; Taste: **[F8]**), werden Sie gefragt, welcher Art das Symbol sein soll. Dieselben Wahlmöglichkeiten erhalten Sie, wenn Sie ein neues Symbol erstellen (Menü: **Einfügen • Neues Symbol**; Taste: **[Strg] + [F8]**).

Die einzige Besonderheit beim Konvertieren eines bestehenden Elements in ein Symbol ist, dass Sie den Registrierungspunkt wählen können. Wie Sie schon im Abschnitt »Koordinatensystem« gelesen haben, wird der Mittelpunkt konvertierter Elemente standardmäßig zum Mittelpunkt des Symbols. Wenn Sie einen anderen Registrierungspunkt im (imaginären) Rechteck um die Elemente wählen, wird dieser zum Ursprung. Sie können also auch die linke obere Ecke der bestehenden Elemente als Ursprung für das Koordinatensystem einer Grafik oder eines Movieclips wählen.

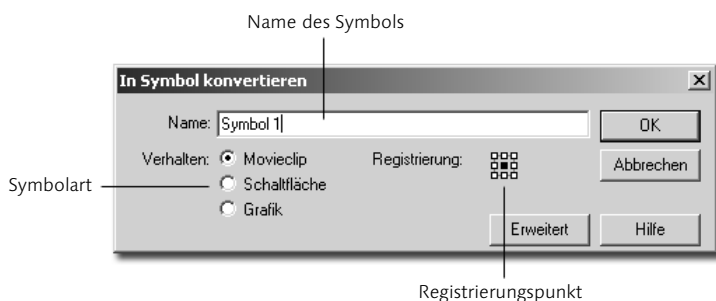


Abbildung 8.11 Art und Namen des Symbols festlegen

Was sich hinter der Schaltfläche **Erweitert** beim Erstellen oder Konvertieren eines Symbols verbirgt, werden wir Ihnen in Kapitel 12, Movieclips, näher erläutern. Dort können Sie ein Symbol exportieren, so dass es per ActionScript verwendet werden kann, ohne eine eigene Instanz im Film zu haben.



Sie können die Art und andere Eigenschaften eines Symbols auch nachträglich verändern. Klicken Sie dazu das Symbol in der Bibliothek mit der rechten Maustaste an (Ein-Tasten-Maus: **[Strg]** + Anklicken) und wählen Sie im Kontextmenü **Eigenschaften**.

Wir sind Ihnen bisher noch eine Erläuterung zu den drei Grundarten von Symbolen schuldig geblieben. Als gute Schuldner zahlen wir nun gerne zurück:

- ▶ **Grafiksymbole** sind die einfachste Form der Symbole. Sie enthalten grafische Elemente und laufen parallel zur Hauptzeitleiste ab. Das heißt ein Bild im Grafiksymbol wird in einem Bild der Hauptzeitleiste angezeigt. Wenn das Ende der Hauptzeitleiste erreicht ist, werden auch vom Grafiksymbol keine weiteren Bilder angezeigt. Die Grafiksymbole haben aus Sicht des ActionScript-Programmierers einen großen Nachteil: Sie erlauben keinen ActionScript-Code und keinen Zugriff per ActionScript. Von diesen Symbolen werden Sie also im Verlauf dieses Buches nichts mehr hören.
- ▶ **Schaltflächensymbole** verhalten sich ganz anders als Grafiksymbole. Sie können vier Zustände annehmen. Diese Zustände sind allerdings rein optisch. Das heißt, wenn der Anwender über eine Schaltfläche fährt, können Sie ein Schlüsselbild im Zustand **Darüber** angeben, das das Aussehen der Schaltfläche in diesem Zustand enthält. Zustände können kein ActionScript enthalten. Schaltflächen lassen sich sehr wohl mit ActionScript-Code versehen, der bei bestimmten Ereignissen (siehe »Schaltflächenereignisse«) eintritt.
- ▶ **Movieclip-Symbole** hießen nicht immer so. Vielmehr handelt es sich hier wohl um eine – drücken wir es vorsichtig aus – geänderte Übersetzungskonvention. Man hat einfach beschlossen, Filmsequenzen in Flash MX wie im englischen Original Movieclips zu nennen. Movieclips sind eigenständig. Sie haben eine eigene Zeitleiste und sind hierarchisch unter der Hauptzeitleiste, einer Art Haupt-Movieclip, angeordnet. Movieclips können direkt aus ActionScript angesprochen und verändert werden. Sie sind überhaupt eines der mächtigsten Werkzeuge, wenn nicht sogar das mächtigste, für ActionScript-Programmierer. Deswegen haben wir ihnen ein eigenes Kapitel (Kapitel 12, Movieclips) gewidmet. Movieclips können selbst in ihrer Zeitleiste ActionScript-Code enthalten. Genauso lässt sich Code auch für eine Movieclip-Instanz, hier verbunden mit dem Schlüsselwort `onClipEvent`, schreiben.

Auch eine Instanz kann verschiedene Arten (hier auch Symbolarten) annehmen. Die Instanz wechselt dann unabhängig vom Symbol in eine andere Art. Das ist aber nur in Ausnahmefällen sinnvoll.





Symbole können auch mehrere Elemente und Gruppen von einzelnen Elementen enthalten.

8.2.5 Szenen

Szenen sind für den ActionScript-Entwickler und ActionScript-Anwender eigentlich ziemlich langweilig. Man kann auf sie nicht einzeln zugreifen. Sie werden im Bedienfeld **Szene (Fenster • Szene)** verwaltet und vom Flash Player in der dort angegebenen Reihenfolge durchlaufen.

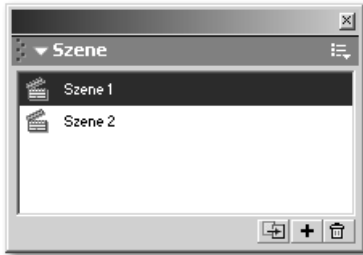


Abbildung 8.12 Das Bedienfeld **Szene**

Das einzig Bemerkenswerte, was Sie mit Szenen in ActionScript anfangen können, ist, von einer zur anderen zu springen. Dazu verwenden Sie in den Funktionen `gotoAndPlay` (springe zu einem Bild und spiele weiter ab) und `gotoAndStop` (springe zu einem Bild und halte an) einfach zuerst den Szenennamen in Anführungszeichen, dann die Nummer des Bildes in der Szene. Nehmen wir an, Sie hätten einen Flash-Film mit zwei Szenen (**Szene 1** und **Szene 2**). Mit folgendem Befehl springen Sie zum ersten Bild in Szene 2 und spielen weiter ab:

```
gotoAndPlay("Szene 1", 1);
```

Mehrere Szenen werden vom Flash-Film automatisch aneinander gehängt. Dies gilt auch für die Bildnummerierung. Hat also die erste Szene 25 Bilder, dann beginnt die zweite Szene mit Bild 26. Folgender Sprung würde auch auf das erste Bild der zweiten Szene wechseln:

```
gotoAndPlay(26);
```

8.2.6 Textfelder

Textfeldern haben wir in diesem Buch ein eigenes Kapitel gewidmet (Kapitel 13, Textfelder). Dort zeigen wir, wie man sie per ActionScript

erstellt. Aber auch schon vorher kommen Textfelder häufig zum Einsatz. Man unterscheidet folgende Arten von Textfeldern:

- ▶ Statische Textfelder, die nur einen Beschreibungstext beinhalten. Sie können per ActionScript nicht bearbeitet oder erstellt werden.
- ▶ Eingabefelder, also Felder, in die der Anwender Text eingeben kann. Für diese Textfelder können Sie im Eigenschafteninspektor eine Variable vergeben, mit der Sie dann per ActionScript auf die Eingaben des Anwenders zugreifen können.
- ▶ Dynamische Textfelder können ebenfalls eine Variable erhalten, mit der sie sich dynamisch füllen lassen.

Alle Einstellungen für Textfelder werden in Flash nach dem Zeichnen des Feldes mit dem Text-Werkzeug im Eigenschaftenspektor vorgenommen.



Abbildung 8.13 Eigenschafteninspektor für ein dynamisches Textfeld

8.2.7 Film testen

Als guter ActionScript-Programmierer sollte man seinen Film sehr oft testen. Um genauer zu sein: immer wieder, auch nach kleineren Arbeitsschritten. Damit sind nicht die Steuerungsbefehle oben im Menü **Steuerung** oder die Symbolleiste **Steuerung** (**Fenster • Symbolleisten • Steuerung**) gemeint, sondern ein realer Test.

Für einen Test unter (nahezu) realen Bedingungen gibt es den Befehl **Steuerung • Film testen** (Tastenkürzel: Strg + ↵).

Wenn Sie einen Film testen, wird er automatisch in eine **swf**-Datei umgewandelt. **swf** ist die Dateierweiterung für veröffentlichte, also auch nicht mehr editierbare Flash-Filme. Editierbare Flash-Dateien haben die Endung **fla**.



In der Testumgebung werden Sie viel Zeit verbringen. Sie können hier auch beispielsweise mit dem Bandbreiten-Profiler (**Ansicht • Bandbrei-**

ten-Profiler) das Download-Verhalten des Flash-Films und das Streaming beobachten und verschiedene Qualitätsstufen testen.



Wir werden Ihnen noch einiges über das Testen von Filmen erzählen, und zwar dort, wo Sie es vielleicht gar nicht erwarten, nämlich in Kapitel 15, Debugging. Was hat Testen mit Fehlersuche zu tun? In ActionScript sehr viel. Sie testen nicht nur, um zu sehen, wie ihr Film im Player auf dem Rechner des Anwenders aussehen wird; Sie testen auch, um eventuelle Fehler und Probleme in Ihrem Code zu finden.

Die Testumgebung hat gegenüber dem normalen Flash Player im Browser noch eine Besonderheit: das **Ausgabe-Fenster (Fenster • Ausgabe oder [F2])**. Es kann auch schon zum Testen von Code in der Entwicklungsumgebung angezeigt werden und enthält auftretende Fehler und Ausgaben, die in ActionScript mit der Funktion `trace` erzeugt wurden.

`trace` ist eigentlich ein reines Testvehikel. Mit folgender Code-Zeile geben Sie einen Text im **Ausgabe-Fenster** aus:

```
trace("Eine Ausgabe!");
```



Übrigens, auch wenn das **Ausgabe-Fenster** ausgeblendet ist, wird es von einer `trace`-Anweisung automatisch angezeigt.



Abbildung 8.14 Die Ausgabe

8.2.8 Komponenten

Komponenten sind eigentlich Movieclips. Sie haben festgesetzte Parameter, die übergeben werden. In den Komponenten werden ActionScript-Funktionen definiert, die mit den Parametern arbeiten.

Um es etwas einfacher zu erklären: Komponenten sind wieder verwendbare Movieclips, die durch ActionScript einige zusätzliche Möglichkeiten erhalten haben. In Flash gibt es ein eigenes Bedienfeld für die Komponenten (**Fenster • Komponenten**). Hier finden Sie anfangs die vorgefertigten Komponenten (siehe Abbildung 8.1), können aber auch eigene Komponenten hinzufügen.

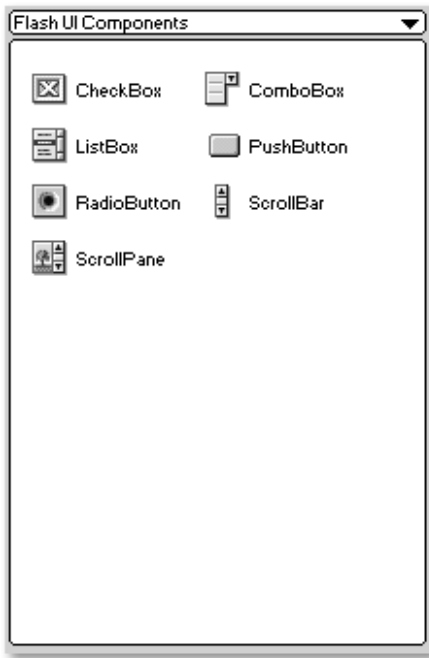


Abbildung 8.15 Das Bedienfeld **Komponenten**

Mehr zu Komponenten erfahren Sie in Kapitel 20, Komponenten von Flash MXÜberprüfen. Dort werden Sie sehen, wie man mit den vorgefertigten Komponenten ein einfaches Formular erstellt.



8.3 **ActionScript in Flash**

Sie haben in diesem Kapitel schon erfahren, an welchen Stellen in Flash ActionScript hinzugefügt werden kann und dass dafür immer ein Ereignis nötig ist. Hier finden Sie noch einmal eine kurze Zusammenfassung der wichtigsten Stellen, an denen ActionScript zum Einsatz kommt:

- ▶ in einem Schlüsselbild,
- ▶ in der Zeitleiste eines Movieclips in einem Schlüsselbild,
- ▶ in der Instanz eines Movieclips,
- ▶ in der Instanz einer Schaltfläche.

Es gibt viele Stellen, an denen ActionScript-Code auftauchen kann, aber nur eine Stelle, an der er eingegeben wird: im **Aktionen**-Bedienfeld. Jede Zeile, jedes Zeichen, jeder Code-Schnipsel in diesem Werk wird im **Aktionen**-Bedienfeld eingegeben.



Bevor wir Ihnen das **Aktionen**-Bedienfeld genauer erklären, bleibt uns nur noch eine Ermahnung: Achten Sie immer darauf, wo Sie sich gerade befinden. Bearbeiten Sie gerade einen Movieclip, haben Sie ein Schlüsselbild oder eine Instanz eines Symbols aktiviert?

Um festzustellen, wo man sich gerade befindet, gibt es einige ganz probate Mittel. Wenn Sie folgende Dinge überprüfen, sollten Sie wissen, wo Sie sich gerade befinden:

1. Die Leiste über der Bühne zeigt, ob man sich in einem Symbol oder Movieclip befindet und welche Szene gerade aktuell ist.



Abbildung 8.16 Die Leiste über der Bühne zeigt, ob man sich in einem Movieclip befindet.

2. In der Zeitleiste sehen Sie, in welcher Ebene Sie sich gerade befinden. Gewöhnen Sie sich daran, eine eigene **Skript**-Ebene zu erstellen und nur dort ActionScript einzufügen. Wenn Sie dann diese Ebene verlassen haben, wissen Sie, dass Sie aus Versehen eine Symbol-Instanz angeklickt haben.
3. Achten Sie darauf, ob eine Symbolinstanz oder ein anderes Element auf der Bühne ausgewählt ist.

8.3.1 Aktionen-Bedienfeld

Im **Aktionen**-Bedienfeld spielt also die Musik, was ActionScript betrifft. Dieses Bedienfeld hat standardmäßig zwei verschiedene Modi, die es auf den ersten Blick sehr unterschiedlich aussehen lassen:

- Im *Normalmodus* klicken Sie verschieden Aktionen und Befehle aus einer langen Liste auf der linken Seite des **Aktionen**-Bedienfelds zusammen und tragen Parameter in Textfelder auf der rechten Seite ein. Diese werden dann in das Skript übernommen, das unten in einem kleinen Ausschnitt angezeigt wird (siehe Abbildung 8.17).
- Im *Expertenmodus* sind Sie wesentlich freier in Ihren Möglichkeiten. Der Skript-Teil nimmt fast den gesamten Teil des Bedienfelds ein. Die ActionScript-Elemente können allerdings auch mit der Werkzeugleiste auf der linken Seite eingefügt werden. Die Parameter werden aber nicht mehr vorgegeben, sondern müssen vom Anwender von Hand eingegeben werden (siehe Abbildung 8.18).

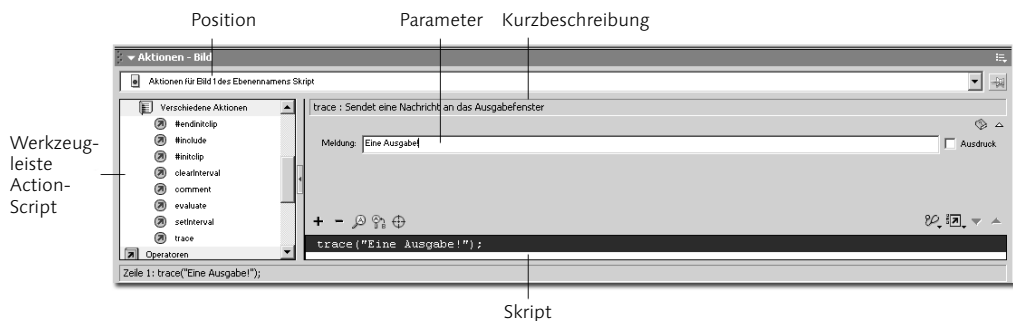


Abbildung 8.17 Der Normalmodus



Abbildung 8.18 Der Expertenmodus

Die interessanteste Frage ist natürlich, wie man vom einen in den anderen Modus wechselt. Dafür gibt es zwei Methoden. Für eine gibt es ein eigenes Symbol, das Sie in Abbildung 8.19 sehen.

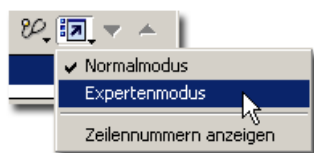


Abbildung 8.19 Den Modus wechseln

Die Alternative ist der Befehl im Menü des Bedienfelds (siehe Abbildung 8.4). Dort sehen Sie übrigens auch noch die Tastenkürzel für die beiden Modi: **Strg** + **⇧** + **N** für den Normalmodus und **Strg** + **⇧** + **E** für den Expertenmodus.

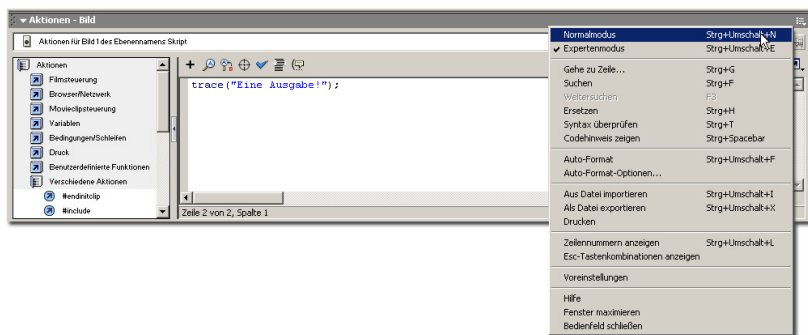


Abbildung 8.20 Den Modus noch einmal wechseln



Im Folgenden werden wir zuerst die Gemeinsamkeiten der beiden Modi, dann die Unterschiede näher erläutern. In diesem Buch verwenden wir dann nur noch den Expertenmodus. Der Grund liegt auf der Hand: Wir wollen Ihnen nicht zeigen, wie Sie Funktionen zusammenklicken, sondern wir wollen Ihnen näher bringen, wie man mit ActionScript richtig programmiert. Sie können allerdings durchaus zwischendurch einmal in den Normalmodus wechseln, wenn Sie eine übersichtliche Darstellung der Parameter für eine Funktion oder Ähnliches sehen möchten.

Hilfen in beiden Modi

Die Gemeinsamkeiten der beiden Modi muss man nicht lange suchen. Zunächst ist die Werkzeugleiste auf der linken Seite eine Betrachtung wert. Wir werden ihr noch oft begegnen. Sie finden dort alle, wirklich alle Programmierwerkzeuge wie Funktionen, Operatoren und Methoden von ActionScript. Unser besonderes Augenmerk gilt an dieser Stelle noch dem letzten Eintrag der Werkzeugleiste, dem Index. Hier werden alle ActionScript-Elemente alphabetisch aufgelistet. Mithilfe des Index finden Sie sehr schnell etwas, wenn Sie ungefähr wissen, wie es heißt.

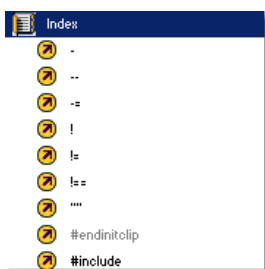


Abbildung 8.21 Der Index

Ein weiteres Hilfsmittel im **Aktionen**-Bedienfeld ist Ihnen vielleicht schon aufgefallen, die **Position** oben im Feld. Sie gibt an, für welche Stelle im Flash-Film das ActionScript-Skript gilt. Hier finden Sie also einen sehr expliziten Hinweis, wo Sie sich gerade befinden, und das ist ja, wie oben schon erwähnt wurde, eines der wichtigsten Dinge in Flash.

Sie können in der Auswahlliste jede Stelle in einem Flash-Film auswählen, an dem ActionScript-Code eingefügt werden kann. Mit dieser Liste haben Sie also einen direkten Zugriff auf die Skripte in Ihrer Datei. Dies gilt allerdings nur für die Skripte für die Instanzen und Schlüsselbilder der Zeitleiste. Skripte, die in einer anderen Zeitleiste, beispielsweise einem Movieclip-Symbol, eingefügt werden, sind hier nicht eingetragen. Dazu müssen Sie zuvor zu dem Movieclip wechseln. Sie sehen, es ist trotz einiger Hilfestellungen immer wichtig, zu wissen, wo man sich gerade befindet.

Ausgewähltes Skript fixieren

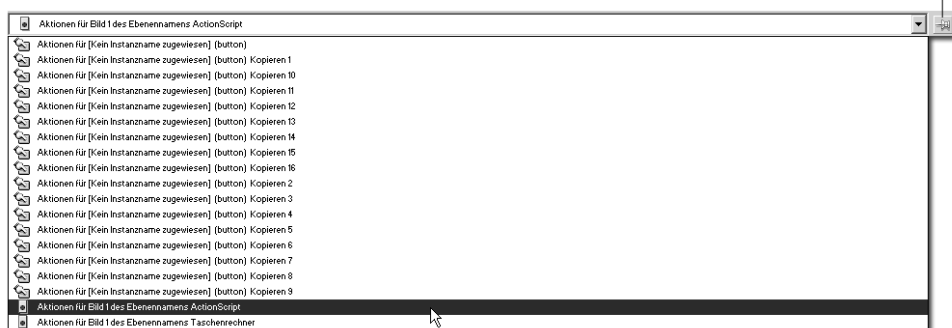


Abbildung 8.22 Die verschiedenen Stellen für Skripte

Wenn Sie das Nagel-Symbol neben der Auswahlliste anklicken, wird das ausgewählte Skript fixiert. In Flash wird diese Option **Aktuelles Skript immer vorn** genannt. Das Skript bleibt dann immer aktiv im **Aktionen**-Bedienfeld. Sie können auch die Auswahlliste erst wieder verwenden, wenn der Nagel gelöst ist. Diese Option wird hauptsächlich verwendet, wenn Sie im Flash-Film Instanzen und Ebenen durchgehen möchten und dabei immer ein bestimmtes Skript im **Aktionen**-Bedienfeld benötigen.

Für Programmierer besonders praktisch sind die Möglichkeiten, im Quelltext zu suchen und Gesuchtes zu ersetzen. Diese Möglichkeiten gibt es ebenfalls in beiden Modi.

Sie greifen darauf entweder mit den Befehlen im Menü des Bedienfelds (**Suchen** und **Ersetzen**) oder mit den Symbolen zu, die in Abbildung 8.23 zu sehen sind.

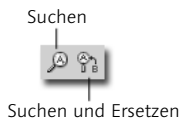


Abbildung 8.23 Die zwei Symbole für Suchen und Suchen und Ersetzen

Beim Suchen gibt es nur eine Einstellung, nämlich, ob zwischen Groß- und Kleinbuchstaben unterschieden werden soll. Ein Blick auf die Dialogfelder verrät, dass Sie mit der Option **Weitersuchen** einen Suchbegriff bei mehrmaligem Vorkommen immer ab der aktuellen Fundstelle noch einmal suchen können. Dazu können Sie übrigens auch das Tastenkürzel **[F3]** verwenden, das auch außerhalb der Dialogfenster funktioniert.

Beim Ersetzen können Sie jeweils das nächste Vorkommen ersetzen (Schaltfläche **Ersetzen**) oder Sie ersetzen gleich alle Vorkommen.

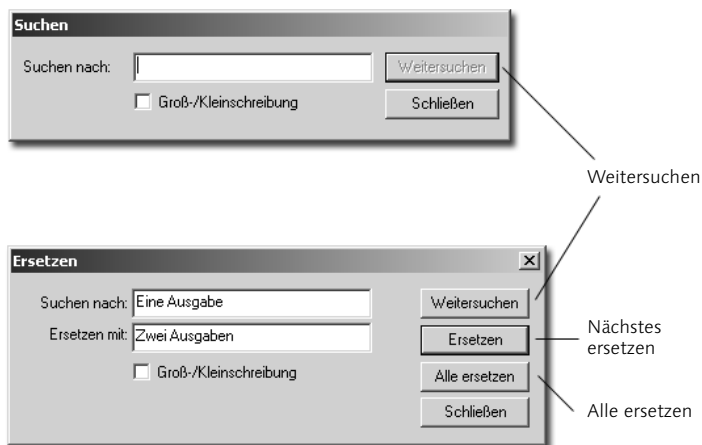


Abbildung 8.24 Die Dialogfelder **Suchen** und **Ersetzen**

Wenn Ihr Code umfangreicher wird, ist es hilfreich, wenn Sie sich schnell zu bestimmten Zeilennummern fortbewegen können. Auch Fehlermeldungen im **Ausgabe**-Fenster verweisen oft auf Zeilennummern. Was liegt also näher, als direkt in die Zeile zu springen? Dies funktioniert mit dem Befehl **Gehe zu Zeilennummer** aus dem Menü des Bedienfelds oder mit der Tastenkombination **[Strg] + [G]**.



Abbildung 8.25 Auf Zeile 10 springen

Wie erkennen Sie aber, in welcher Zeile Sie sich gerade befinden? Im Expertenmodus ist das einfach, die Zeilennummer steht links unter dem Skript im **Aktionen**-Bedienfeld, wenn sich der Mauszeiger im Skript befindet. Dort steht sogar noch, wie lang das Skript insgesamt ist und in welcher Spalte sich der Cursor befindet. Im Normalmodus steht die Zeilennummer nur dann links unten im Bedienfeld, wenn eine Zeile aktiviert ist, die Spaltennummer fehlt.

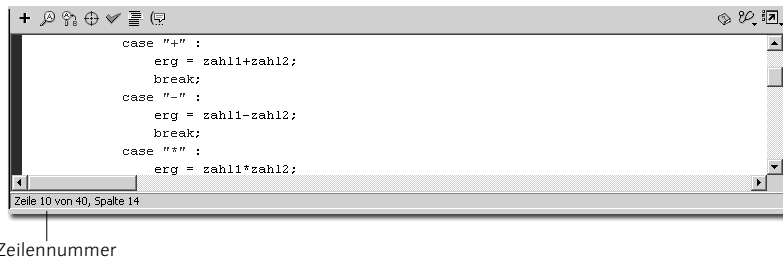


Abbildung 8.26 Zeilennummer und Spalte im Expertenmodus

Um im Normalmodus die Zeilennummer zu erhalten, benötigen Sie den Befehl **Zeilennummer anzeigen** aus dem Menü des Bedienfelds. Sie finden denselben Befehl auch noch einmal hinter dem Symbol, mit dem Sie auch zwischen Normal- und Expertenmodus wechseln.

Die Zeilennummern können in beiden Modi gute Dienste leisten. Ob Sie sie einsetzen, ist reine Geschmacksache.

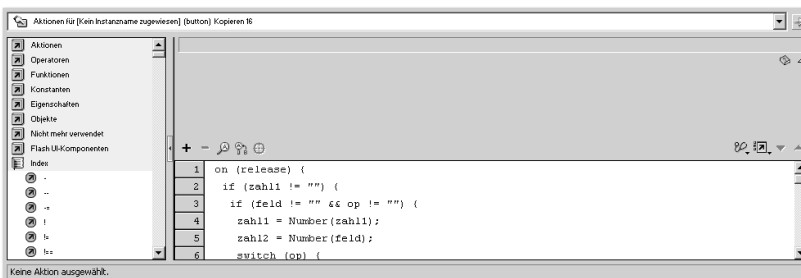


Abbildung 8.27 Zeilennummern im Normalmodus

Normalmodus

In diesem Abschnitt wollen wir Sie noch auf einige Möglichkeiten aufmerksam machen, die vor allem im Normalmodus zum Einsatz kommen.

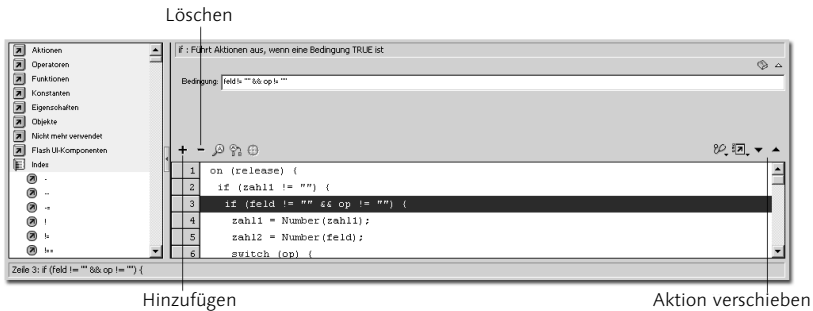


Abbildung 8.28 Besonderheiten im Normalmodus

In Abbildung 8.1 sehen Sie einige Funktionen, die im Normalmodus öfter verwendet werden oder dort sogar exklusiv vorhanden sind. Es gibt im Normalmodus drei Möglichkeiten, eine Aktion hinzuzufügen. Bevor wir näher darauf eingehen, noch ein kurzes Wort zum Begriff Aktion. Eine Aktion im weiteren Sinne umfasst die meisten Sprachelemente von ActionScript und ist nicht trennscharf. Sie werden in den folgenden Kapiteln viele Begriffe kennen lernen, die einzelne Elemente der Sprache wesentlich besser beschreiben, dennoch sei hier angemerkt, dass der Begriff Aktion durchaus manchmal sehr weit gefasst wird und auch von Macromedia nicht vollständig definiert ist.

Nun aber zu den Möglichkeiten Aktionen, oder besser: Sprachelemente, hinzuzufügen:

- ▶ Sie klicken ein Element in der Werkzeugleiste des **Aktionen**-Bedienfelds doppelt an.
- ▶ Sie ziehen ein Element aus der Werkzeugleiste auf das Skript-Fenster im **Aktionen**-Bedienfeld.
- ▶ Die vorherigen Möglichkeiten unterstellen, dass die Werkzeugleiste des **Aktionen**-Bedienfelds eingeblendet ist. Da man sie durch Anklicken des Pfeils an ihrem rechten Rand auch ausblenden kann, muss es noch eine Alternative geben. Diese ist das Plus-Symbol im Bedienfeld, das in Abbildung 8.29 mit **Hinzufügen** beschriftet ist. Es öffnet eine Art aufklappbare Mini-Werkzeugleiste mit allen Action-Script-Elementen.

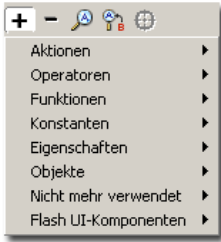


Abbildung 8.29 Eine Aktion hinzufügen

Das Gegenstück zum Plus-Symbol ist das Minus-Symbol. Es dient dazu, die ausgewählte Aktion zu löschen. Mit den beiden Pfeil-Symbolen wird die ausgewählte Zeile um eins nach oben oder unten verschoben.

Auf eine weitere Besonderheit möchten wir noch hinweisen: Wenn Sie eine Änderung im Skript vornehmen, wird die Änderung für den jeweiligen Parameter sofort übernommen und umgekehrt.

Sie können übrigens mit gedrückter `[Strg]`-Taste auch mehrere Aktionen markieren. Mit  markieren Sie mehrere übereinander liegende Aktionen. Sobald mehrere Zeilen markiert sind, fehlen allerdings viele Möglichkeiten, diese zu bearbeiten. Unter anderem sind die Parameter verschwunden, da zwei markierte Aktionen ja unterschiedliche Parameter haben.



Theoretisch ist die Verwendung des Normalmodus nun geklärt. Der Theorie soll noch ein sehr kurzes Praxisbeispiel folgen. Wir wollen mit der `trace`-Anweisung eine Ausgabe machen.

1. Zuerst gehen wir in der Werkzeugleiste oder im Menü des Plus-Symbols auf die Suche. Unser Ziel ist **Aktionen • Verschiedene Aktionen** und dort **trace**.

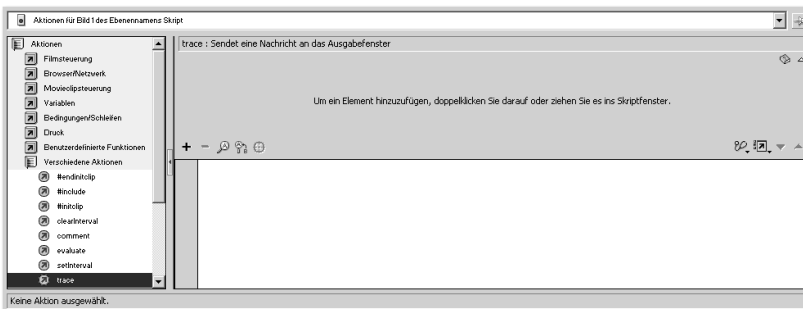


Abbildung 8.30 Das Element suchen



Alternativ könnten Sie auch den Index verwenden, da Sie ja schon wissen, wie die Funktion heißt.

2. Nun müssen Sie **trace** hinzufügen. Tun Sie dies durch Doppelklick, Drag-and-Drop oder Anklicken des Plus-Symbols.
3. Nun wird die `trace`-Anweisung in das Skript-Fenster geschrieben und oben nach den Parametern gefragt. Bei `trace` ist das nur die Meldung, die ausgegeben werden soll.



Wenn Sie das Kontrollkästchen **Ausdruck** aktivieren, können Sie statt des Textes einen programmierten Ausdruck wie beispielsweise eine Variable eingeben. Diese muss allerdings schon vorher definiert worden sein.

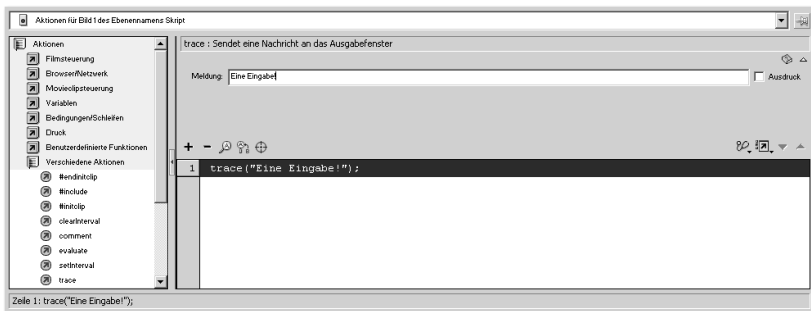


Abbildung 8.31 Die Meldung eingeben

4. Wenn Sie eine Meldung eingegeben haben, ist Ihr erstes Skript auch schon fertig. Wir haben es nicht gewagt, die Meldung »Hallo Welt« zu nennen, weil dies in hunderten von Programmierbüchern das Standardbeispiel ist, aber sehr einfallsreich waren wir auch nicht ☺.

Expertenmodus

Den Expertenmodus werden Sie in diesem Buch noch sehr oft sehen, verwenden und hoffentlich auch lieben lernen. Es ist einfach angenehmer, Quellcode zu schreiben, statt ihn zusammenzuklicken. Natürlich mag man einwenden: Das ist Geschmacksache, für Designer ist Code-Schreiben schrecklich und die Strichpunkte vergisst man auch dauernd. Auf der anderen Seite gibt es eine Syntaxprüfung, die Sie in Kapitel 15, Debugging, kennen lernen werden. Der Überblick über den Code ist im Expertenmodus gut gewährleistet, Sie können den Code automatisch formatieren lassen und Code-Hinweise gibt es auch.

Aber der Reihe nach. Sie schreiben gerade im Skript-Fenster des Expertenmodus, wenn der Balken links in Blau statt in Einheitsgrau erscheint.

Die wichtigen und noch nicht bekannten Optionen im Expertenmodus sehen Sie in Abbildung 8.32. Wir gehen sie im Anschluss kurz durch.

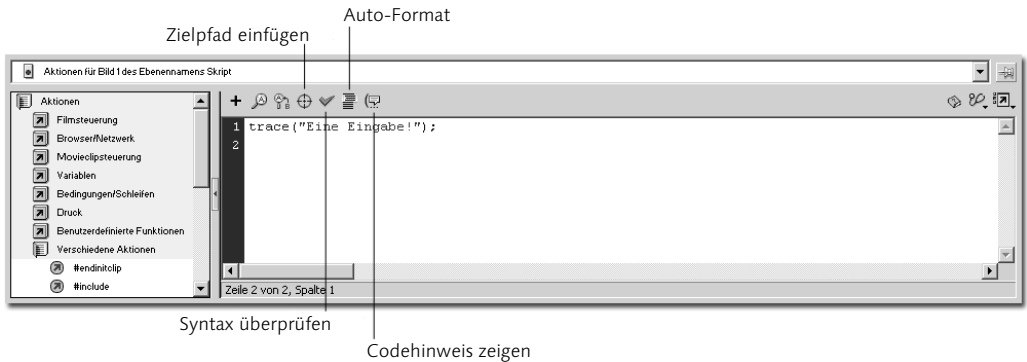


Abbildung 8.32 Neue und wichtige Optionen im Expertenmodus

- **Zielpfad einfügen** erlaubt Ihnen, mit einem einfachen Dialogfeld Verweise auf verschachtelte Movieclips einzufügen. Das sagt Ihnen noch nicht viel? Keine Angst, wir lüften alle Geheimnisse rund um Movieclips in Kapitel 12.



Abbildung 8.33 Das Dialogfeld Zielpfad einfügen

- **Syntax überprüfen** erlaubt Ihnen einen automatischen Check der Syntax des aktuellen Skripts. Eventuelle Fehler werden im Ausgabe-Fenster angezeigt. Mehr dazu lesen Sie in Kapitel 15, Debugging.
- **Auto-Format** rückt den Quellcode automatisch ein und sorgt für korrekte Abstände. Dafür muss der Quellcode allerdings syntaktisch kor-

rekt sein. Für die automatische Formatierung gibt es noch einige Optionen. Sie rufen sie mit dem Befehl **Auto-Format-Optionen** aus dem Menü des **Aktionen**-Bedienfelds auf. Vor allem wird gesteuert, wie geschweifte Klammern bei Kontrollstrukturen und Funktionen behandelt werden. Außerdem wird festgelegt, ob zwischen dem Schlüsselwort `function` und dem Funktionsnamen ein Leerzeichen stehen soll oder nicht. Da Sie bis jetzt vermutlich noch keine Ahnung haben, welche dieser Optionen sinnvoll sind, geben wir eine kleine Empfehlung. Die **Leerstellen um Operatoren** sind nützlich, da Operationen so übersichtlicher werden. Die anderen Optionen können Sie sicherlich auch später noch verändern.

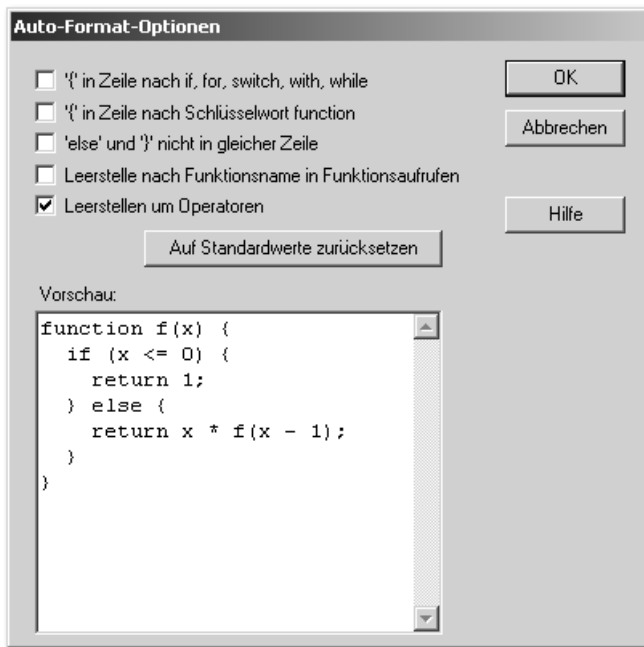


Abbildung 8.34 Die Optionen für die automatische Formatierung

- **Codehinweise anzeigen** liefert eine sehr praktische Auto-Vervollständigung für den gerade eingegebenen Befehl. Wenn Sie ein ActionScript-Element eingeben, werden Ihnen die weiteren Optionen oder die benötigten Parameter angezeigt. Dies ist in ActionScript zwar nicht so komfortabel wie in komplexen Entwicklungsumgebungen wie z.B. Visual Studio .NET, aber doch im Programmier-Alltag eine gute Hilfe.

```

1 trace {
2   trace(Meldung);

```

Abbildung 8.35 Codehinweis für die trace-Anweisung

Im Menü des **Aktionen**-Bedienfelds findet sich auch der Befehl **Drucken**. Er druckt keine Bilder des Flash-Films, sondern das aktuelle Skript. Manchmal ist es gerade bei längeren Skripten im Expertenmodus so, dass man im Papiausdruck Fehler leichter findet. Wenn Sie also ein Papiermensch sind, nur zu.



8.3.2 Externer Code

In Flash können Sie Quellcode auslagern. Hier gibt es zum einen die einmalige Auslagerung, das heißt Quellcode, den man nur einmal in eine externe Datei speichert und von dort wieder laden kann. Dieser Code wird aber nicht mit der Flash-Datei verknüpft. Zum anderen können Sie Code auch direkt in Flash importieren. Er wird dann live vom Flash Player aus einer externen **swf**-Datei geholt.

Um Code einmalig auszulagern, verwenden Sie den Befehl **Als Datei exportieren** aus dem Menü des **Aktionen**-Bedienfelds. Normalerweise wird die Datei mit der Endung **.as** gespeichert. Sie können allerdings auch eine andere Endung verwenden. Die Datei ist eine reine Textdatei und kann jederzeit mit einem normalen Texteditor wie den bei den Betriebssystemen mitgelieferten gelesen und bearbeitet werden.

Soll ein Skript aus einer Datei wieder in das aktuelle Skript eingefügt werden, verwenden Sie den Befehl **Aus Datei importieren** aus dem Menü des **Aktionen**-Bedienfelds.

Beim Importieren wird das gesamte aktuelle Skript gelöscht. Der Import kann allerdings mit dem Menübefehl **Bearbeiten • Rückgängig** beziehungsweise **[Strg] + [Z]** rückgängig gemacht werden.



Wenn Sie einen externen Code dauerhaft mit einer Flash-Datei verknüpfen möchten, bietet sich **#include** als Lösung an. Folgender Code fügt an der Stelle im Code, an der er steht, den Code aus der Datei **skript.as** ein.

```
#include "skript.as";
```

Da außer dem Dateinamen kein anderer Pfad angegeben ist, muss sich die Datei im selben Ordner befinden. Alternativ kann ein vollständiger relativer oder absoluter Pfad angegeben werden.

```
#include "../skript/skript.as";
```

Mit obiger Zeile wird beispielsweise ein Skript eingefügt aus dem Ordner **skript**, der auf der gleichen Ebene liegt wie der aktuelle Ordner.



Sie können auch Quellcode in exportierte Movieclips einer anderen Flash-Datei auslagern. Wenn Sie die Movieclips aus der fremden Bibliothek (öffnen Sie diese beispielsweise mit **Datei • Als Bibliothek öffnen**) als Instanz in die eigene Datei ziehen, werden sie mit der swf-Datei der anderen Flash-Datei verknüpft. Sie müssen natürlich beide **swf**-Dateien mitliefern bzw. auf Ihren Webserver stellen, haben dann aber eine perfekte Skriptbibliothek.

8.3.3 Voreinstellungen

Für das **Aktionen**-Bedienfeld und allgemein für ActionScript gibt es einige Voreinstellungen. Sie finden die **Voreinstellungen** als Befehl im Menü des **Aktionen**-Bedienfelds. Alternativ können Sie auch den Menübefehl **Bearbeiten • Voreinstellungen** verwenden und dort in das Register **ActionScript-Editor** wechseln.

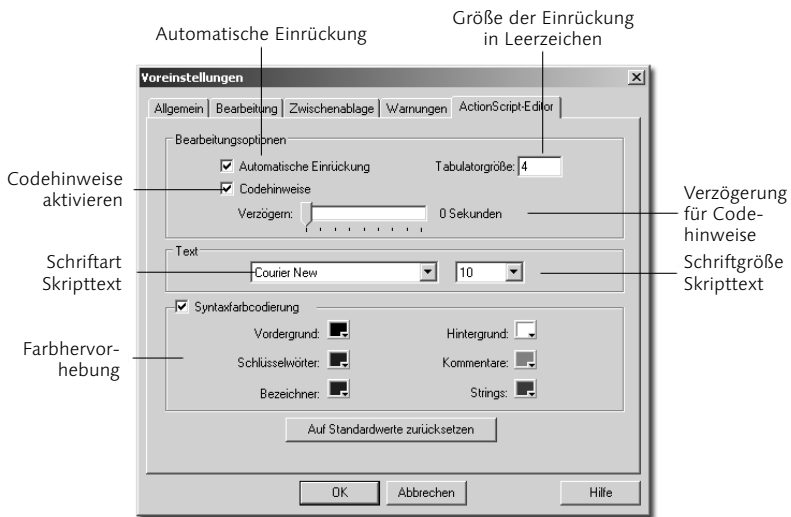


Abbildung 8.36 Die Voreinstellungen für den ActionScript-Editor

Folgende Voreinstellungen sind für unsere weitere Arbeit wichtig:

- ▶ **Automatische Einrückung** sorgt dafür, dass der Quellcode eingerückt wird. Damit werden beispielsweise Anweisungen in einer if-Kontrollstruktur eingerückt.
- ▶ Die Größe der Einrückung definiert die **Tabulatorgröße**. Sie gibt an, wie viele Leerzeichen ein Tabulator zur Einrückung haben soll.
- ▶ Das Kontrollkästchen **Codehinweise** aktiviert die Codehinweise dauerhaft, während das Symbol im Expertenmodus des **Aktionen**-Bedienfelds nur für die aktuelle Anweisung Codehinweise einblendet.
- ▶ Der Schieberegler zum **Verzögern** besagt, wie viel Zeit vergehen soll, bis ein Codehinweis angezeigt wird.
- ▶ Im Bereich **Text** können Sie die Schriftart und Schriftgröße des Code-Textes festlegen.

Als Schriftart sollten Sie eine diktengleiche Schrift verwenden. Bei einer solchen Schrift sind alle Zeichen gleich breit. Dadurch werden die Zeilen mit ihren Zeichen sauber untereinander angeordnet. Die bekannteste diktengleiche Schrift ist Courier, die deswegen normalerweise für Skripte verwendet wird.



- ▶ Über das Kontrollkästchen **Syntaxfarbcodierung** können Sie einstellen, ob im ActionScript-Code bestimmte Code-Elemente eine eigene Farbe erhalten. Diese Option sollten Sie prinzipiell immer wählen, da dies nicht nur hübsch aussieht, sondern auch nützlich ist. Allerdings ist meist die Frage, ob die vorgegebenen Grau- und Blauwerte günstig sind. Teilweise unterscheiden sie sich zu wenig voneinander. Eventuell sollten Sie hier Werte ändern.

Ein kurzer Blick über den Tellerrand, also in die anderen Register der Voreinstellungen, offenbart uns, dass auch dort noch eine interessante Option für ActionScript-Programmierer zu finden ist.

Im Register **Warnungen** können Sie eine Warnung anzeigen lassen, wenn Formatierungen aus dem Expertenmodus verloren gehen.

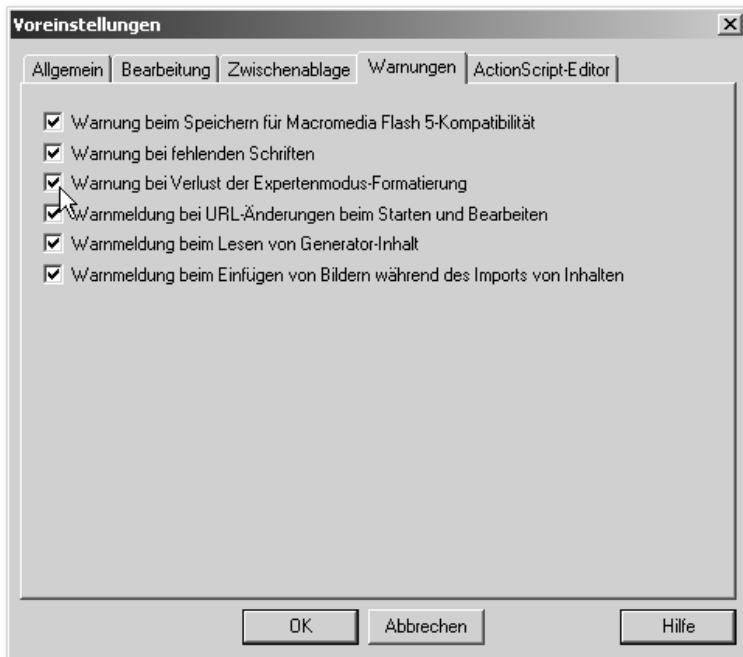


Abbildung 8.37 Das Register **Warnungen**

Ein Verlust von speziellen Formatierungen wie beispielsweise Einrückungen oder zusätzlichen Leerzeilen passiert immer erst, wenn Sie die entsprechende Aktion im Normalmodus editieren. Eine Um- und Rückumwandlung übersteht die Formatierung problemlos. Bevor Flash ändert, fragt es dann mit einer Warnmeldung, wie sie in Abbildung 8.38 zu sehen ist, nach.

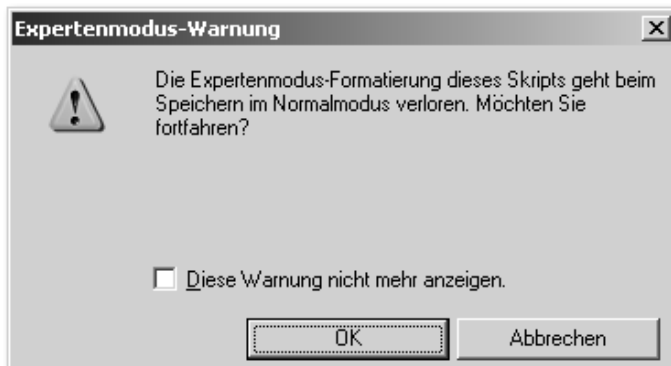


Abbildung 8.38 Warnmeldung wegen Verlust der Formatierungen

Sollten Sie bei der Warnmeldung **Diese Warnung nicht mehr anzeigen** aktivieren, wird automatisch die Voreinstellung deaktiviert und kann auch nur über das Register **Warnungen** in den Voreinstellungen erneut aktiviert werden.

Mit dem Menübefehl **Bearbeiten • Tastenkombinationen** erreichen Sie ein Dialogfenster, in dem Sie alle Tastenkürzel von Flash anpassen können. Aus Sicht des ActionScript-Anwenders sind Anpassungen allerdings eher selten nötig.



8.3.4 Ansichten

Flash rühmt sich – größtenteils zu Recht –, dass es sowohl für Programmierer als auch für Grafiker die richtige Umgebung zum Entwickeln bietet.

Deswegen gibt es auch unter **Fenster • Bedienfeldsätze** mehrere Standardlayouts für Grafiker oder Entwickler. Der Unterschied liegt hauptsächlich darin, welche Bedienfelder ein- beziehungsweise ausgeblendet sind. Die Layouts liegen außerdem für unterschiedliche Bildschirmauflösungen vor.

Wir empfehlen hier bewusst kein Standardlayout, da die Arbeitsweisen verschiedener Menschen einfach sehr unterschiedlich sind. Entgegen aller Gerüchte gibt es sogar innerhalb der Spezies der echten Programmierer echte Unterschiede 😊.

8.3.5 Referenz

Eines der wichtigsten Hilfsmittel für den Entwickler haben wir bisher noch nicht vorgestellt: die Referenz. Sie ist ein normales Bedienfeld, in dem aber eine Menge Informationen stecken.

Um die Referenz einzublenden gibt es, wie bei so vielem, mehrere Wege. Im **Aktionen**-Bedienfeld hat die Referenz ein eigenes Symbol, mit der sie angezeigt werden kann.



Abbildung 8.39 Das Symbol für die Referenz im **Aktionen**-Bedienfeld

Abbildung 8.40 Alternative Wege führen über das Menü **Fenster • Referenz** oder das Tastenkürzel  + .

Wenn Sie die Referenz zum ersten Mal sehen, werden Sie vermutlich kurz stutzen und denken, dass sie genauso aussieht wie das Aktionen-Bedienfeld. Das trifft für die Werkzeugleiste mit allen ActionScript-Elementen zu. In der Referenz dient sie allerdings eher als Verzeichnisleiste. Sie können bequem auf alle Elemente zugreifen und sich eine Beschreibung der Elemente in das Fenster rechts im Bedienfeld laden.



Abbildung 8.41 Die Referenz



Der Beschreibungstext stammt aus der Referenz in der Dokumentation von Flash. Er entspricht auch größtenteils der Online-Dokumentation. Zusätzliche Hilfe erhalten Sie dennoch oft unter **Hilfe • Flash verwenden**. Vor allem die Such-Funktion funktioniert recht gut. Die Tutorials (**Hilfe • Tutorials**) sollten Sie sich ebenfalls ansehen, auch wenn das Beispiel für die ActionScript-Einführung für Einsteiger doch ein wenig zu komplex erscheint.

8.3.6 Film-Explorer

Einen Blick wert ist der Film-Explorer. Er kann alle Movieclips, sonstige Symbole, Ebenen, Videos, Sounds etc. anzeigen und übersichtlich hierarchisch darstellen. Sie blenden den Film-Explorer mit dem Menübefehl **Fenster • Film-Explorer** oder dem Tastenkürzel **[Alt] + [F3]** ein.

Für ActionScript bietet der Film-Explorer die Option, es ein- oder auszublenken. Den Code erkennen Sie an einem nach rechts oben gerichteten Pfeil.



Ein besonderes Schmankerl des Film-Explorers ist, dass Sie nach Namen von Elementen suchen können. Die Symbolnamen werden hinter den Elementen immer zuerst angezeigt, dann folgt bei Instanzen der Instanzname.

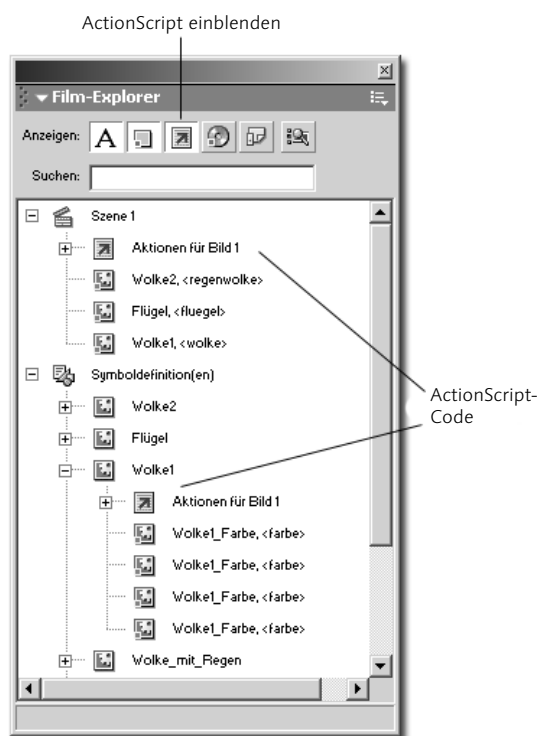


Abbildung 8.42 ActionScript im Film-Explorer

9 ActionScript: Grundlagen

Aller Anfang ist leicht – wenn man ihn mit dem Ende vergleicht.

– Gerhard Uhlenbruck

Wegen der vielen Parallelen zu JavaScript halten wir die Spracheinführung kurz und knackig. Dennoch verzichten wir nicht auf die wichtigsten Fakten.

9.1 Variablen und Datentypen

Wenn Sie Ihre Flash-Filme mit ActionScript interaktiver und spannender gestalten wollen, benötigen Sie eines ganz bestimmt: Variablen. Eine Variable ist ein Datenspeicher, dessen Wert sich ändern kann. In ActionScript müssen eine Menge Informationen in einer Variablen gespeichert werden. Das können Anwendereingaben sein, Daten aus einer Datenbank, Datumswerte, Texte und vieles mehr.

In ActionScript werden Variablen wie in JavaScript mit dem Schlüsselwort `var` definiert. Die Definition einer Variable nennt man auch Deklaration. Sie können jede Variable nur einmal deklarieren.

```
var a;
```

Das war's schon, die erste Variable ist definiert. Nun können Sie mit ihr arbeiten. Moment, bisher haben Sie noch nicht viel, mit dem Sie arbeiten können. Sie haben der Variablen ja noch keinen Wert zugewiesen, sondern nur einen Namen. Einen Wert weisen Sie mit dem Gleichheitszeichen `=` zu.

```
var a;
```

```
a = 2;
```

Oder etwas kürzer:

```
var a = 2;
```

Diese Kurzform nennt man auch Initialisierung. Hier wird der erste Variablenwert bei der Deklaration festgelegt. In diesem Fall hat die Variable natürlich sofort einen Wert.



Achten Sie immer darauf, dass alle Anweisungen in ActionScript wie in JavaScript mit einem Strichpunkt beendet werden müssen.

Wenn einer Variablen noch kein Wert zugewiesen wurde, ist ihr Wert standardmäßig `undefined` (noch nicht definiert). Wird eine Variable mit dem Wert `undefined` beispielsweise als Zahl in Rechnungen verwendet, wird `undefined` als Wert 0 angesehen. Das hat unter Umständen Auswirkungen auf die Rechnung und ist eine häufige Fehlerquelle, beispielsweise in Form einer Division durch 0. Als String, das ist eine Zeichenkette, ist `undefined` in ActionScript übrigens ein leerer String, in JavaScript dagegen ein String mit dem Wert `undefined`.

Möchten Sie mehrere Variablen hintereinander deklarieren, geht dies ebenfalls sehr einfach, indem die Variablen durch Kommata getrennt werden.

```
var a, b, c;
```

Natürlich können Sie mehreren Variablen hintereinander auch gleich einen Wert zuweisen:

```
var a = 1, b = 2, c = 3;
```



ActionScript ist sehr tolerant. Sie können einer Variablen auch einfach einen Wert zuweisen, ohne sie vorher mit `var` zu definieren. Was auf den ersten Blick wie ein großes Geschenk aussieht, kann sich aber auch ins Negative verkehren, denn Sie haben dann keine Kontrolle mehr darüber, welche Variablen bereits deklariert sind. Unsere Empfehlung ist also, Variablen immer mit `var` zu deklarieren.

9.1.1 Namenskonventionen

Bevor Sie jetzt loslegen und wild Variablen definieren, noch ein paar einschränkende Worte zu den Variablennamen. Damit es keine Missverständnisse und Verwechslungen gibt, müssen Variablennamen in ActionScript festgelegten Konventionen folgen:

- ActionScript unterscheidet zwischen Groß- und Kleinschreibung. Geben Sie hier gut Acht, da dies eine der häufigsten Fehlerquellen ist, die sich in komplexeren Skripten oft nur schwer finden lässt. Wenn Sie eine Variable deklarieren und beispielsweise beim Zuweisen des Wertes Groß- und Kleinschreibung verwechseln, haben Sie eine neue Variable definiert:

```
var a;  
a = 2;
```

- Ein Variablenname darf nur mit einem Unterstrich (`_`) oder einem Buchstaben beginnen. Nicht erlaubt ist ausdrücklich der Beginn mit einer Zahl.
- Ein Variablenname darf nach dem ersten Buchstaben nur Unterstriche, Buchstaben und Zahlen enthalten.
- Nicht erlaubt sind Umlaute, Sonderzeichen, Leerzeichen, Punkte und Kommata.
- Ein Variablenname darf keinem Schlüsselwort in ActionScript entsprechen. Manche Schlüsselwörter oder Werte wie `undefined` sind zwar mögliche Variablennamen, aber in einer Rechnung werden sie nicht als solche, sondern als Wert o erkannt.

break	case	continue	default
delete	do	else	for
function	if	in	instanceof
new	null	on	onClipEvent
return	switch	this	typeof
undefined	var	void	while
with			

Tabelle 9.1 Die Schlüsselwörter in ActionScript

Neben den hier aufgeführten neuen Schlüsselwörtern sollten Sie noch andere missverständliche Bezeichner vermeiden. So lösen alte Schlüsselwörter aus Flash 4 und Begriffe aus der Programmierung zwar keine Fehlermeldung aus, beim Ansehen dreht sich einem aber der Magen um. Folgender Code funktioniert, wir wollen ihn aber in keinem Ihrer Projekte sehen J:



```
var float = 2;
var Return = 3;
var Class = float + Return;
trace(Class);
```

Wenn Sie den Abschnitt über Namenskonventionen aufmerksam gelesen haben, können Sie natürlich auch gleich sagen, welche der folgenden Variablendeklarationen erlaubt sind und welche nicht:

```
var 666;
var m3;
```

```
var switch;
var ____;
var zieh an;
var zieh.aus;
```

Erlaubt ist aus diesem illustren Grüppchen die zweite, `m3`, und die vierte mit drei Unterstrichen. Die erste beginnt mit einer Ziffer, `switch` ist ein Schlüsselwort, die fünfte enthält ein Leerzeichen, die sechste einen Punkt.

Nächste Frage: Welche der Deklarationen ist sinnvoll? Eigentlich keine, denn auch die zwei gültigen sagen wenig aus über den Inhalt der Variablen. Das aber ist wichtig, damit man auch nach längerer Zeit den Code lesen oder ihn auch mal einem Kollegen schicken kann, ohne dass dessen Aspirin-Verbrauch deutlich ansteigt. Achten Sie also immer auf die Einhaltung der Namenskonventionen, auf sprechende Namen und kommentieren Sie Variablen.

Die Kommentarzeichen in ActionScript sind die gleichen wie in JavaScript: zwei Schrägstriche (Slashes: `//`) für einzeilige Kommentare, mehrzeilige Kommentare werden mit `/*` begonnen und mit `*/` beendet.



Mehrzeilige Kommentare werden im **Normalmodus** und beim Wechsel in den **Normalmodus** automatisch als einzeilige Kommentare angezeigt. Wechseln Sie zurück, werden sie wieder rückumgewandelt.



Sie sollten Filmsequenzen und Objekten und Variablen innerhalb eines Flash-Films nicht die gleichen Namen geben, da es sonst Probleme geben kann. Bei größeren Projekten empfiehlt es sich, spezielle Namenskonventionen zu verwenden. Beispielsweise könnte man allen Filmsequenzen `fs_` voranstellen.

9.1.2 Gültigkeitsbereich

Mit die wichtigste Frage bei Variablen in Flash ist natürlich: Wo sind sie gültig? Bei JavaScript ist diese Frage einfach zu klären: in einer Funktion oder in einem Skript. In Flash hingegen gibt es Ereignisse auf Schaltflächen, Aktionen in Schlüsselbildern der Zeitleiste und vieles mehr.

Glücklicherweise gibt es in Flash einige sehr klare Regeln für die Gültigkeit von Variablen:

- ▶ Variablen, die innerhalb einer Funktion definiert werden, sind nur dort gültig.
- ▶ Variablen außerhalb einer Funktion sind für die gesamte Zeitleiste gültig. Das hört sich unbedeutend an, ist es aber nicht. Die Tragweite dieser Aussage ist groß, denn dies sorgt für folgende Besonderheiten:
 - ▶ Egal, wo die Variable definiert wird, sie gilt für alle Schaltflächen, Objekte und Bilder der Zeitleiste.
 - ▶ Sie gilt nicht für andere Zeitleisten, also beispielsweise nicht für Movieclip-Symbole, die eine eigene Zeitleiste haben. Variablen, die dort definiert sind, können nicht direkt adressiert werden. Mehr dazu lesen Sie im Abschnitt »Movieclip«.

9.1.3 Datentypen

Bisher haben wir uns bei der Deklaration von Variablen nicht um den Datentyp des Wertes gekümmert. Der Datentyp ist die Art des Wertes. Wenn der Variablenname für den zugehörigen Wert die Antwort auf die Frage »Wer bist Du?« darstellt, so ist der Datentyp die Antwort auf »Was bist Du?« oder konkreter »Von welcher Art bist Du?«.

Es gibt in ActionScript einfache Datentypen, die sehr oft verwendet werden:

- ▶ Strings – Zeichenketten oder allgemeiner Text. Dieser Datentyp wird immer in Anführungszeichen eingeschlossen.

```
var s = "Ein beliebiger String";
```

- ▶ Zahlen – Diesen Datentyp haben wir bisher verwendet. Er steht in ActionScript sowohl für Integer (ganzzahlige Werte), als auch für Gleitkommazahlen (auch Fload, Werte mit Nachkommastellen). Sie werden einer Variablen einfach zugewiesen.

```
var a = 2;
```

- ▶ Boolean – auch Wahrheitswert. Ein Datentyp, der angibt, ob etwas wahr (`true`) oder falsch (`false`) ist. Die Antwort auf die Frage »Bellt ein Hund?« wäre also `true`, die Antwort auf »Bellt eine Katze?« wäre `false`.
- ▶ `undefined` – ist ein Datentyp mit einem Wert, nämlich `undefined`.
- ▶ `null` – ist ebenfalls ein eigener Datentyp mit dem einzigen Wert `null`.

ActionScript hält außerdem noch einige komplexere Datentypen bereit:

- ▶ **Array** – Ein Array ist eigentlich eine Liste von Werten, die mit einem Index versehen sind. So kann auf jeden Wert einzeln zugegriffen werden. Mehr dazu im Abschnitt »Arrays«.
- ▶ **MovieClip** – Dieser Datentyp erlaubt die direkte Ansprache von MovieClips. Dazu gibt es die Methoden des MovieClip-Objekts. Mehr dazu im Abschnitt »MovieClip«.
- ▶ **Object** – Variablen von diesem Typ können alle Objekte aufnehmen, die von eigenen oder mitgelieferten Klassen abgeleitet werden. Mehr hierzu im Abschnitt »Klassen und Objekte«.

Mit dem Befehl `typeof` können Sie den Datentyp auslesen. Als Ergebnis wird ein String zurückgeliefert. Folgendes Beispiel liefert als Ergebnis `boolean`.

```
var x = true;
var typ = typeof x;
trace(typ);
```

`typeof` liefert für alle Datentypen den korrekten Namen. Einzig für ein Array wird `object` zurückgeliefert.

Strings

Wie wir schon gehört haben, werden Zeichenketten immer in Anführungszeichen eingeschlossen. Dabei ist es egal, ob es sich um doppelte oder einfache Anführungszeichen handelt. Die folgenden Varianten funktionieren beide:

```
var y = 'Einfache';
var z = "Doppelte";
```

Ein Problem taucht bei Strings auf, wenn Sie mit ActionScript Text mit Zeilenumbruch ausgeben möchten. Hierfür gibt es zwei Möglichkeiten; für beide benötigen Sie einen dynamischen Text.

Erstellen Sie ein Textfeld. Wechseln Sie in der Auswahlliste links oben im Eigenschafteninspektor auf **Dynamischer Text**. Ein dynamisches Textfeld zeichnet sich dadurch aus, dass sich der darin enthaltene Text über eine Variable steuern lässt. Diese Variable enthält also, wie nicht anders zu erwarten, Daten vom Typ String.

Für unser Beispiel ändern Sie den Umbruch des Textfeldes bitte auf **Mehrzeilig**. Das erlaubt ein mehrzeiliges Textfeld mit automatischem

Umbruch. Die Variable hat ein eigenes Feld im Eigenschafteninspektor. Vergeben Sie für sie den Namen **text**.

Mit der Eingabe in das Feld **Var** im Eigenschafteninspektor ist die Variable deklariert. Sie müssen sie nicht noch einmal extra im ActionScript-Code deklarieren.

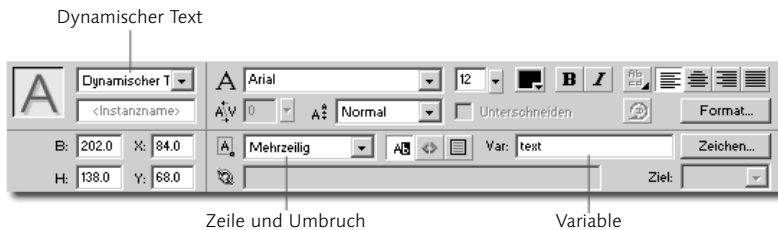


Abbildung 9.1 Der Eigenschafteninspektor für das dynamische Textfeld

Nun geht es an den ActionScript-Code. Wir vergeben ihn für das erste Bild in der Zeitleiste. Klicken Sie dazu einfach das Bild an und geben Sie folgenden Code in das Aktionen-Bedienfeld ein:

```
text = "Ein sehr langer Text, der nicht ganz in das Feld passt.  
Ein zweiter Absatz, der sich vom ersten abheben soll."
```

Dieser Code weist dem Textfeld einen Text zu. Wenn Sie den Film testen (**Strg** + **↵**), sehen Sie, dass der Umbruch zwar automatisch vorgenommen wird, der zweite Absatz aber noch nicht vom ersten getrennt ist.

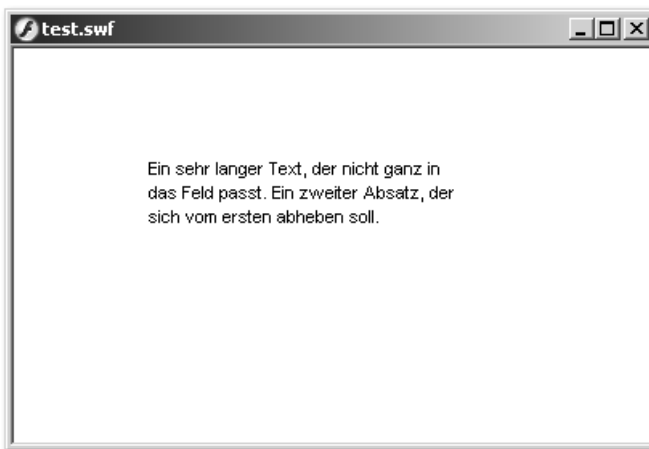


Abbildung 9.2 Die Absätze sind nicht getrennt.

Wie trennt man also Absätze und fügt Zeilenumbrüche ein? Versuchen Sie es nicht mit einem Zeilenumbruch im Code. Dies würde zu einer Fehlermeldung führen. Für solche Aufgaben gibt es Sonderzeichen. Für einen Zeilenumbruch benutzen Sie das Sonderzeichen `\n`.

Wir verändern also leicht den Code:

```
text = "Ein sehr langer Text, der nicht ganz in das Feld passt.\n\nEin zweiter Absatz, der sich vom ersten abheben soll."
```

Das Ergebnis ist wie gewünscht. Es werden zwei Zeilenumbrüche eingefügt. Der automatische Zeilenumbruch ist nach wie vor vorhanden.

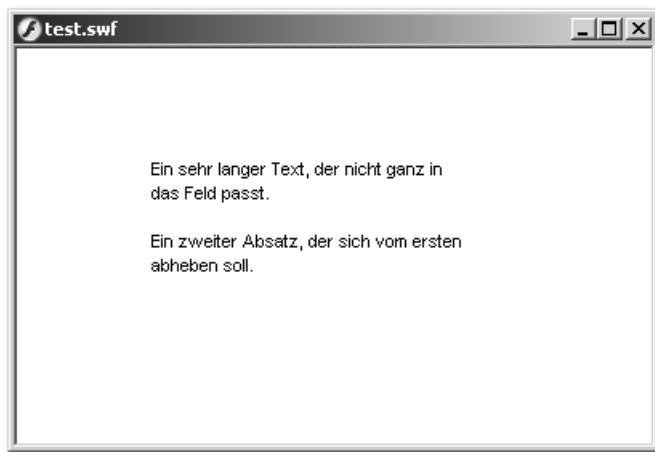


Abbildung 9.3 Mit `\n` haben wir die Absätze auseinander gebracht.

Neben dem Sonderzeichen für den Zeilenumbruch, das Sie eben kennen gelernt haben, gibt es noch einige andere Sonderzeichen. Allen gemeinsam ist der Start mit einem Backslash (`\`). Der Backslash ist in diesem Kontext ein Entwertungszeichen, das heißt danach folgende Zeichen verlieren ihre Funktion als Programmierzeichen und werden zu einem normalen String. Nur so können viele Zeichen, wie beispielsweise Anführungszeichen, ausgegeben werden. Ohne vorangestellten Backslash würde der Interpreter denken, der String sei schon zu Ende.

In der folgenden Tabelle finden Sie eine Übersicht der wichtigsten Sonderzeichen.

Zeichen	Beschreibung
\\	Einfacher Backslash
\'	Einfaches Anführungszeichen
\"	Doppeltes Anführungszeichen
\n	Zeilenumbruch
\t	Tabulator
\u	Beliebiges Unicode-Zeichen in hexadezimaler Schreibweise. Der Code wird angehängt. \u00A9 steht beispielsweise für das Copyright-Zeichen (©).
\x	Hexadezimaler Wert
\b	Rückschritt
\r	Wagenrücklauf
\f	Seitenvorschub

Tabelle 9.2 Die wichtigsten Sonderzeichen

Flash verwendet den Zeichencode Latin-1. Er ist unter dem Namen ISO-8859-1 standardisiert. Die Unicode-Zeichensätze utf-8 und utf-16 werden nicht verwendet, obwohl sie wesentlich komplettere Sammlungen möglicher Zeichen sind. Das liegt daran, dass Unicode so viele Zeichen umfasst; der Flash-Film würde dadurch ganz einfach zu groß werden.



Dynamische Textfelder werden Ihnen in diesem Buch noch öfter begegnen. Sie können in diese Felder beispielsweise auch HTML-Code eintragen, der dann ausgeführt wird. Dazu müssen Sie die Option **Text als HTML wiedergeben** aktivieren, wie es in Abbildung 9.4 zu sehen ist. Mit HTML könnten Sie mittels <p>-Tags ebenfalls die Absätze realisieren oder mit
 einen Zeilenumbruch einfügen.

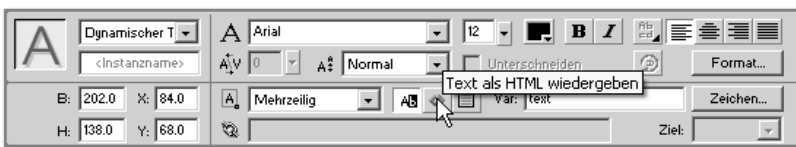


Abbildung 9.4 Die Option Text als HTML wiedergeben aktivieren

Zahlen

In vielen Programmiersprachen wird von Haus aus in Integer (ganze Zahlen) und Float (Gleitkommazahlen, Zahlen mit Nachkommastellen) unter-

schieden. Oftmals gibt es noch weitere Datentypen wie Single oder Double, die unterschiedlich viele Nachkommastellen und Werteräume unterstützen.

In ActionScript ist das einfacher. Hier gibt es nur einen Zahlentyp. Nachkommastellen werden durch einen Punkt abgetrennt.

```
var a = 2.4256;
```

Die Trennung der Nachkommastellen durch einen Punkt ist im angelsächsischen und amerikanischen Sprachraum üblich und deswegen auch in Programmiersprachen Usus. Deutschsprachigen macht dies oftmals aber Probleme, da wir gewohnt sind, ein Komma zu setzen. Schalten Sie hier am Anfang beim Programmieren den Kopf ein und seien Sie sich dieser Fehlerquelle bewusst.

Sehr große Zahlen können Sie auch mit Exponent schreiben. Dazu wird einfach ein e (auch E) angefügt:

```
var a = 15.5e5;
```

e5 in unserem Beispiel steht für 10^5 . Ausgeschrieben lautete die Zahl also 1550000. e kann auch negative Werte annehmen. $10e-2$ entspräche beispielsweise 0,1.

evtl. HexadezimalDie möglichen Zahlenwerte in Flash sind sowohl nach oben als auch nach unten begrenzt. Die beiden Grenzen liegen bei 1,79e308, also $1,79 * 10^{308}$ im positiven Bereich und $5e-324$, also $5 * 10^{-324}$. Die beiden Grenzen finden Sie in den Eigenschaften MAX_VALUE und MIN_VALUE der Klasse Number. Auch wenn Sie mit dem Zugriff auf Klassen und deren Unterklassen sowie Fallunterscheidungen noch keine Erfahrung haben, zeigen wir Ihnen eine kurze Überprüfung, ob eine Zahl innerhalb dieser Grenzen liegt. Mit Ihren Kenntnissen in JavaScript werden Sie sie verstehen, oder Sie lesen einfach später noch einmal hier nach.

```
function testgroesse(zahl) {  
    if ((zahl>=Number.MIN_VALUE) && (zahl<=Number.MAX_VALUE)) {  
        trace("Zahl ist ok");  
    } else {  
        trace("Zahl geht nicht");  
    }  
}
```

Die obige Funktion übernimmt eine Zahl (Variable zahl) und testet dann in einer Fallunterscheidung, ob die Zahl größer ist als der minimal mögli-

che Wert und gleichzeitig kleiner als der maximal mögliche. Ist dies der Fall, wird ausgegeben, dass die Zahl in Ordnung ist, wenn nicht, erscheint eine Fehlermeldung.

Der folgende Code ruft die obige Funktion mit einer zu kleinen Zahl auf. Es wird der Text »Zahl geht nicht« ausgegeben.

```
var x = 3e-325;  
testgroesse(x);
```

Die Unendlichkeit ist in Flash durch die Schlüsselworte `Infinity` (positive Unendlichkeit) und `-Infinity` (negative Unendlichkeit) angegeben. Sie können diese Unendlichkeit sogar zum Rechnen verwenden. Positive Unendlichkeit findet sich in der Eigenschaft `Number.POSITIVE_INFINITY`, negative in `Number.NEGATIVE_INFINITY`.

Die Unendlichkeit lässt sich außerdem mit der Funktion `isFinite` testen.

```
isFinite(3e-325);
```

ergäbe also `false`.

9.1.4 Automatische Typkonvertierung

ActionScript erleichtert das Leben eines Programmierers bei der Handhabung von verschiedenen Datentypen erheblich. Wenn der Wert einer Variablen einen anderen Typ annimmt, wird diese Änderung von Flash automatisch vorgenommen. Dieses Prinzip nennt man automatische Typkonvertierung.

Die Variable selbst nimmt dabei immer den Datentyp ihres zugewiesenen Wertes an. Wird ein Wert mit einem anderen Datentyp zugewiesen, ändert sich auch der Datentyp der Variablen.

```
var a = "text";  
a = 2;
```

Im obigen Beispiel ist die Variable `a` erst String, dann Zahl.

Der Wert selbst kann in Rechenoperationen automatisch geändert werden. Dabei ändert sich allerdings der Datentyp der Variablen nicht. Ein einfaches Beispiel verdeutlicht dies:

```
var a = "2";  
var b = 4;  
var prod = a * b;  
trace(prod);
```

Der obige Code gibt das Produkt eines Strings und einer Zahl aus. Das Ergebnis ist trotzdem korrekt (8 ☺). Flash verwandelt den String für die Operation in eine Zahl. Im Hintergrund bleibt der Wert der Variablen `a` allerdings ein String.

Perfekt, denken Sie, da muss man ja überhaupt nichts mehr tun. Schade, falsch gedacht. Testen Sie einmal folgendes Skript:

```
var a = "2";  
var b = 4;  
var summe = a + b;  
trace(summe);
```

Wenn Sie das gleiche Ergebnis erhalten haben, das in Abbildung 9.5 zu sehen ist, müssen Sie nicht mit dem Rechenschieber nachrechnen. Hier ist die Typkonvertierung nach hinten losgegangen. Das `+` ist nicht nur das Rechensymbol (der Operator) für die Addition, sondern es dient auch zum Verbinden von Strings. In Abbildung 9.5 zieht der Interpreter den String der Zahl vor, macht aus der Zahl einen String und fügt die beiden Strings aneinander. Was Sie also im Ausgabefenster sehen, ist ein String.



Abbildung 9.5 Hoppla, da stimmt was nicht ...

Wenn die automatische Typkonvertierung nicht funktioniert, müssen Sie selbst Hand anlegen und die Typkonvertierung vornehmen. Dies nennt man explizite Typkonvertierung; sie wird im nächsten Abschnitt erläutert.

9.1.5 Explizite Typkonvertierung

Für die explizite Typkonvertierung stellt ActionScript drei Funktionen zur Verfügung:

- ▶ `String()` zur Umwandlung in eine Zeichenkette,
- ▶ `Number()` zur Umwandlung in eine Zahl,
- ▶ `Boolean()` zur Umwandlung in einen Wahrheitswert.

Mit diesem Wissen ist es einfach, das Problem aus dem vorangegangenen Abschnitt zu lösen:

```
var a = "2";
var b = 4;
var summe = Number(a) + b;
trace(summe);
```

Die Variable `a` wird einfach mit `Number` in eine Zahl umgewandelt. Die Umwandlung von Strings in Zahlen ist allerdings nicht immer ganz schmerzfrei. Ist der String keine gültige Zahl, so wird der Wert `NaN` (Not a Number) zurückgeliefert.

Wollen Sie nur den Anfang eines Strings in eine Zahl umwandeln, helfen die beiden Funktionen `parseInt()` (extrahiert die erste ganze Zahl) und `parseFloat()` (extrahiert die erste Gleitkommazahl).

Dieses Beispiel gewinnt die Gleitkommazahl `4.56` aus einem längeren String:

```
var gk = "4.56 m über dem Erdboden";
var dist = parseFloat(gk);
trace(dist);
```

In der folgenden Tabelle finden Sie eine Übersicht aller Umwandlungen.

Von	In	Beschreibung
Boolean	String	Aus <code>true</code> wird <code>"true"</code> , aus <code>false</code> wird <code>"false"</code> .
Boolean	Zahl	Aus <code>true</code> wird <code>1</code> , aus <code>false</code> wird <code>0</code> .
String	Boolean	<code>"</code> wird zu <code>false</code> , alle übrigen Strings zu <code>true</code> .
String	Zahl	Umwandlung, wenn gültige Zahl, sonst <code>NaN</code>
Zahl	Boolean	<code>0</code> wird <code>false</code> , alle übrigen Zahlen werden <code>true</code> .
Zahl	String	Umwandlung in eine Zeichenkette

Tabelle 9.3 Explizite Typumwandlung von einfachen Datentypen

Für die Umwandlung in einen String gibt es zur Funktion `String()` noch eine Alternative, die Sie vielleicht aus anderen Programmiersprachen kennen, und zwar ist das `toString()`. Diese Funktion ist eigentlich eine Methode des String-Objekts und wird an die Variable mit dem String angehängt:

```
var a = 5;  
var text = a.toString();
```

Wenn die Funktion `toString()` direkt an eine Zahl angehängt wird, muss diese in Klammern stehen:

```
(2.5).toString();
```



Innerhalb der Klammern von `toString` können Sie noch einen Parameter übergeben, nämlich die Basis des der Zahl zugrunde gelegten Zahlensystems. 10 stünde für das dezimale System, 16 für das hexadezimale. Damit lassen sich Werte gut vom einen Zahlensystem in das andere konvertieren.

9.2 Operatoren

Variablen sind mächtige Werkzeuge, aber nur, wenn man mit ihnen arbeiten kann. Für diese Arbeit gibt es die Operatoren. Der bekannteste Operator ist sicherlich das `+`-Zeichen für die Addition, das wir in den vorangegangenen Abschnitten schon öfter verwendet haben. Es hat sogar eine doppelte Funktion: Zahlen addiert es, Strings verbindet es.



Das Verbinden von Strings nennt man auch Konkatenation. Eine Alternative zum `+`-Operator ist `add`. Allerdings gilt dieser Konkatenations-Operator als veraltet und wird nur noch wegen der Abwärtskompatibilität zu Flash 4 beibehalten.

Einen Operator, den wir andauernd einsetzen, werden Sie in den folgenden Kategorien vergeblich suchen. Der Zuweisungsoperator (`=`) dient dazu, Variablen mit einem Wert zu versehen. Bei den arithmetischen Operatoren werden Sie allerdings auf eine Besonderheit stoßen: kombinierte Operatoren aus arithmetischem Symbol und Zuweisungsoperator.

9.2.1 Arithmetische Operatoren

Die arithmetischen Operatoren sind, wie der Name schon sagt, für Rechenoperationen da. Dabei gelten dieselben Regeln wie in der Schule und allgemein in der Mathematik. Beispielsweise ist die Reihenfolge der Rechenoperationen immer durch »Punkt vor Strich« festgelegt. Außerdem können natürlich runde Klammern (`()`) verwendet werden.

In der folgenden Tabelle finden Sie eine Liste aller arithmetischen Operatoren inklusive deren Beschreibung und einem Beispiel mit echten Zahlen.

Operator	Beispiel	Beschreibung
+	2 + 3 ergibt 5	Addition von Zahlen oder Verbindung von Strings
++	2++ ergibt 3	Zählt um 1 hoch. Wird auch als Inkrement bezeichnet und häufig bei Zählern eingesetzt (siehe »Schleifen«)
+=	var x = 4; x += 5; ergibt x = 9	Addition zum aktuellen Wert der Variablen. Das Ergebnis wird der Variablen wieder zugewiesen.
-	3-2 ergibt 1	Subtraktion von Zahlen. Negativen Zahlen wird ebenfalls ein Minus-Symbol vorangestellt.
--	3-- ergibt 2	Zählt um 1 nach unten. Wird auch als Dekrement bezeichnet und häufig bei Zählern eingesetzt (siehe »Schleifen«)
-=	var x = 5; x -= 4; ergibt x = 1	Subtraktion vom aktuellen Wert der Variablen. Das Ergebnis wird der Variablen wieder zugewiesen.
*	2 * 3 ergibt 6	Multiplikation von Zahlen. Wird bei Angabe ohne Klammern immer vor Addition und Subtraktion ausgeführt
*=	var x = 5; x *= 4; ergibt x = 20	Multiplikation mit aktuellem Wert der Variablen. Das Ergebnis wird der Variablen wieder zugewiesen.
/	4 / 2 ergibt 2	Division von Zahlen
/=	var x = 10; x /= 2; ergibt x = 5	Aktueller Wert der Variablen wird dividiert. Das Ergebnis wird der Variablen wieder zugewiesen.
%	3 / 2 ergibt 1	Ganzzahliger Rest einer Division wird angegeben (auch Modulo).
%=	var x = 10; x %= 3; ergibt x = 1	Aktueller Wert der Variablen wird dividiert. Modulo wird als Ergebnis zurückgegeben und der Variablen wieder zugewiesen.

Tabelle 9.4 Arithmetische Operatoren

Gerade die Kurzformen der Operatoren, also beispielsweise += als kombinierter arithmetischer und Zuweisungsoperator oder auch Inkrement (++) und Dekrement (--), sparen oft Tipparbeit und machen den Code kürzer und meist auch überschaubarer. Allerdings ist auch gegen die Langform nichts zu sagen, wir veranstalten schließlich keinen Perl-Wettbewerb um das kürzeste und kryptischste Skript ☺.



Am Ende dieses Kapitels werden wir als kleine Beispielanwendung einen Taschenrechner erstellen, der unter anderem auch die Operatoren verwendet.

9.2.2 Vergleichsoperatoren

Kontrollstrukturen machen die Programmierung eigentlich erst interessant. Nein, das hier ist nicht die falsche Überschrift. Kontrollstrukturen werden aber erst im weiteren Verlauf dieses Kapitels behandelt, denn ohne Vergleichsoperatoren machen Kontrollstrukturen keinen Sinn.

Kontrollstrukturen haben im Allgemeinen Bedingungen, die überprüft werden. Bei dieser Überprüfung werden meistens Werte verglichen. Dafür benötigt man Vergleichsoperatoren.

Vergleichsoperatoren vergleichen zwei Werte und liefern als Ergebnis einen Wahrheitswert. `true` bedeutet, dass der Vergleich zutrifft, `false` bedeutet, dass er nicht zutrifft.

Folgendes Beispiel illustriert dies. Zuerst werden zwei Variablen definiert. Dann wird mit `if` eine Bedingung geprüft. Nur, wenn `b > a` `true` zurückliefert, wird die folgende Anweisung ausgeführt. Da das aber nicht der Fall sein wird, wird die Anweisung nicht ausgeführt.

```
var a = 5;
var b = 4;
if (b > a)
    trace("Boris Becker gewinnt ein Sandplatzturnier");
```

In der folgenden Tabelle finden Sie einen Überblick über die Vergleichsoperatoren.

Operator	Beispiel	Beschreibung
<code>==</code>	<code>2 == 3</code> ergibt <code>false</code>	Ist gleich. Überprüft, ob zwei Werte gleich sind
<code>></code>	<code>2 > 3</code> ergibt <code>false</code>	Größer als. Überprüft, ob der erste Wert größer als der zweite ist
<code>>=</code>	<code>2 >= 3</code> ergibt <code>false</code> <code>4 >= 4</code> ergibt <code>true</code>	Größer gleich. Überprüft, ob der erste Wert größer oder gleich dem zweiten ist
<code><</code>	<code>2 < 3</code> ergibt <code>true</code>	Kleiner als. Überprüft, ob der erste Wert kleiner als der zweite ist

Tabelle 9.5 Vergleichsoperatoren

<code><=</code>	<code>2 <= 3</code> ergibt <code>true</code> <code>4 <= 4</code> ergibt <code>true</code>	Kleiner gleich. Überprüft, ob der erste Wert kleiner oder gleich dem zweiten ist
<code>!=</code>	<code>2 != 3</code> ergibt <code>true</code>	Ungleich. Überprüft, ob zwei Werte ungleich sind

Tabelle 9.5 Vergleichsoperatoren (Forts.)

Den Zuweisungsoperator (`=`) und den Vergleichsoperator für die Gleichheit (`==`) dürfen Sie auf keinen Fall verwechseln. Dieser Fehler wird häufig begangen, da das Gleichheitszeichen (`=`) in der Mathematik eine Gleichheit impliziert.



Die Werte `undefined` und `null` sind in einem Vergleich identische Werte. Das heißt Sie müssen nur auf einen der beiden Werte prüfen. `NaN` kann nicht verglichen werden, da es immer `false` liefert. Hier muss mit der Funktion `isNaN()` geprüft werden.



Strings vergleichen

Auch Strings lassen sich vergleichen. Was sich auf den ersten Blick erstaunlich anhört, ist eigentlich ganz einfach, wenn man die Besonderheiten beachtet:

- ▶ Zeichen haben in dem Zeichensatz von `Flash`, `Latin-1`, eine bestimmte Position. Das Zeichen mit der höheren Position ist größer.
- ▶ Die Position der Zeichen entspricht der alphabetischen Reihenfolge. `a` ist also kleiner als `z`.
- ▶ Großbuchstaben kommen allerdings immer vor Kleinbuchstaben. Das heißt Großbuchstaben sind immer kleiner als Kleinbuchstaben.
- ▶ Ziffern kommen in `Latin-1` vor Großbuchstaben, sind also kleiner als Buchstaben.
- ▶ Umlaute und Sonderzeichen stehen weiter hinten als normale Buchstaben, sie sind also größer.
- ▶ Strings werden von vorne nach hinten verglichen. Sobald ein unterschiedliches Zeichen entdeckt wird, ergibt sich daraus der Vergleich. Das erste unterschiedliche Zeichen zählt also.
- ▶ Haben zwei Strings nur gleiche Zeichen, einer ist aber länger, so ist der längere auch größer.

Probieren wir das in der Praxis aus. Folgende Vergleiche ergeben `true`:

```
X < a
z > f
ü > z
4 < A
```



Für den Vergleich von Strings gab es in Flash 4 noch eigene Operatoren. Sollten Sie Ihren Film als Flash 4 exportieren wollen, müssen Sie diese noch beachten. Die folgende Tabelle bietet eine kurze Übersicht.

Flash MX und 5	Flash 4 (Beschreibung)
<code>==</code>	<code>eq</code> (equal = gleich)
<code>></code>	<code>gt</code> (greater than = größer als)
<code>>=</code>	<code>ge</code> (greater equal = größer gleich)
<code><</code>	<code>lt</code> (less than = kleiner als)
<code><=</code>	<code>le</code> (less equal= kleiner gleich)
<code>!=</code>	<code>ne</code> (not equal = ungleich)

Tabelle 9.6 Vergleich der String-Operatoren in Flash 5 und MX contra Flash 4



Der Vergleich von Strings funktioniert in JavaScript genauso. Allerdings wird dort der ASCII-Code herangezogen, wohingegen in Flash Latin-1 die Zeichentabelle ist.

Verschiedene Datentypen vergleichen

Nach der Grundregel sollen möglichst nur gleiche Datentypen verglichen werden. Sind die Datentypen verschieden, übernimmt der Interpreter wiederum die Typkonvertierung. Die Regeln für Vergleichsoperatoren haben wir in einer Tabelle zusammengefasst.

Wert1	Wert2	Beschreibung
String	Zahl	Der String wird in eine Zahl konvertiert.
Boolean	String	Boolean wird zu einer Zahl, String ebenso.
Boolean	Zahl	Boolean wird zu einer Zahl.

Tabelle 9.7 Typkonvertierung bei Vergleichsoperatoren

Auch der Datentyp `Object` lässt sich vergleichen, wenn er in einen einfachen Datentyp umgewandelt werden kann.



Im folgenden Code sehen Sie einen Vergleich zwischen einer Zahl und einem String.

```
var a = 2;
var b = "2";
var w = a == b;
trace(w);
```

Dieser Vergleich hat als Ergebnis `true`, da der String in eine Zahl konvertiert wird. Interessant ist, dass dieses Konvertierungsverhalten genau umgekehrt ist als bei arithmetischen Operatoren, wo String-Konkatenation vor Addition geht.

Neu in Flash MX sind die Vergleichsoperatoren `===` (strikte Gleichheit) und `!==` (strikte Ungleichheit). Sie lassen die gerade eben beschriebene automatische Typkonvertierung weg und verhalten sich ansonsten genauso wie ihre Pendanten Gleichheit (`==`) und Ungleichheit (`!=`).

Folgender Vergleich ergibt also ...

```
var a = 2;
var b = "2";
var w = a === b;
trace(w);
```

... `false`. String und Zahl sind also nicht strikt gleich, auch wenn sie die gleiche Ziffer enthalten. Im Umkehrschluss ergibt `a !== b` also `false`.

9.2.3 Logische Operatoren

Wir haben immer noch keine Kontrollstrukturen, und dennoch werden wir ihnen in diesem Abschnitt noch mehr Möglichkeiten verleihen. Ein einfacher Vergleich ist ja sehr schön, die logischen Operatoren erlauben Ihnen aber, mehrere Vergleiche zu koppeln.

Sehen wir uns ein Beispiel an. Der Operator `||` ist das logische ODER. Es ergibt `true`, wenn auch nur einer der Ausdrücke `true` ist.

```
var a = 2;
var b = 3;
var c = 4;
var verg1 = (a > c) || (b > a) || (b > c);
trace(verg1);
```

In unserem Beispiel ergibt der erste Vergleich `false`, der zweite `true`. Dass der dritte noch einmal `false` ergibt, ist schon unerheblich, da ein `true` reicht, um beim logischen ODER als Ergebnis `true` zu liefern.

In der folgenden Tabelle finden Sie eine Übersicht der logischen Operatoren.

Operator	Beispiel	Beschreibung
<code>&&</code>	<code>(2 < 3) && (3 < 4)</code> ergibt <code>true</code>	Logisches UND. Ergibt nur <code>true</code> , wenn beide Vergleiche <code>true</code> ergeben
<code> </code>	<code>(2 < 3) (3 > 4)</code> ergibt <code>true</code>	Logisches ODER. Ergibt <code>true</code> , wenn nur eine der Vergleiche <code>true</code> ergibt. Sobald ein Vergleich <code>true</code> ergibt, wird die Überprüfung beendet, da das Ergebnis <code>true</code> schon feststeht.
<code>!</code>	<code>!(2 == 3)</code> ergibt <code>true</code>	Logisches NICHT. Ändert das Ergebnis des Vergleichs in das Gegenteil. Aus <code>true</code> wird <code>false</code> , aus <code>false</code> wird <code>true</code> .

Tabelle 9.8 Logische Operatoren



Die logischen Operatoren kommen in der Rangfolge immer hinter normalen Vergleichsoperatoren. Zuerst wird also der Vergleich mit dem Vergleichsoperator durchgeführt, dann erst kommt der logische Operator ins Spiel.



Statt der normalen Operatoren gibt es auch die bitweisen Operatoren. Für das logische UND ist das beispielsweise `&`. Dieser Operator vergleicht Zahlen Bit für Bit. Allerdings kommen diese Operatoren nur selten zum Einsatz und werden daher hier nicht behandelt.

9.3 Kontrollstrukturen

Nun ist es endlich soweit, die grundlegendsten Themen sind abgehakt, alle Vorbereitungen für die Kontrollstrukturen sind getroffen. Kontrollstrukturen bergen sehr viele Möglichkeiten für die Programmierung. Wir unterscheiden hier grob in Fallunterscheidungen und Schleifen.

9.3.1 Fallunterscheidungen

Eine Fallunterscheidung ist immer gleich aufgebaut. Sie besteht aus einer Bedingung und einer oder mehreren Anweisungen. Trifft die Bedingung zu, wird die Anweisung ausgeführt.

if

Die if-Fallunterscheidung kennen Sie bereits aus JavaScript. Die Syntax ist einfach:

```
if (Bedingung) {  
    Anweisung;  
}
```

Im folgenden Beispiel wird der Wert 5 ausgegeben, weil die Bedingung $a > 4$ zutrifft.

```
var a = 5;  
if (a > 4) {  
    trace(a);  
}
```

Fallunterscheidungen haben den Vorteil, sehr gut lesbar zu sein. Vorgelesen würde das obige Beispiel lauten: »Wenn a größer als 4, dann gib a aus.«

Natürlich kann eine Fallunterscheidung auch mehrere Anweisungen haben. Folgendes Beispiel gibt a erst aus und erhöht den Wert von a dann um 1.

```
if (a > 4) {  
    trace(a);  
    a++;  
}
```

Kurzformen

Es gibt für die if-Anweisung auch zwei mögliche Kurzformen. Diese funktionieren aber nur mit einer Anweisung.

```
if (Bedingung) Anweisung;
```

und

```
if (Bedingung)  
    Anweisung;
```

Wenn Sie im **Aktionen**-Bedienfeld die Option **Auto-Format** auswählen, wird aus der Kurzform die Langform gebildet.

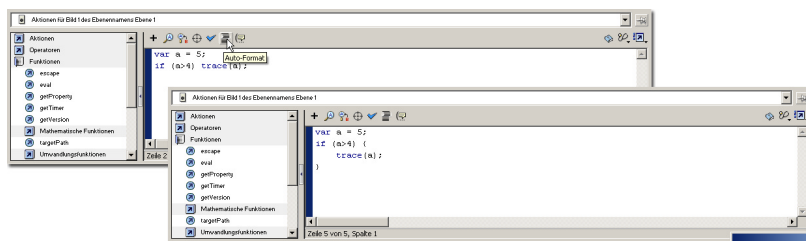


Abbildung 9.6 Mit **Auto-Format** wird aus der Kurz- die Langform.

Verschachteln

Anweisungen innerhalb einer if-Anweisung können natürlich auch weitere if-Anweisungen sein. Dadurch lassen sich if-Anweisungen beliebig tief verschachteln.

Im folgenden Beispiel prüfen wir zuerst, ob *a* größer als 4 ist, in der zweiten Fallunterscheidung testen wir, ob *a* kleiner als 10 ist. Erst dann wird die Anweisung ausgeführt.

```
var a = 5;
if (a>4) {
    if (a<10) {
        trace(a);
    }
}
```



if-Anweisungen lassen sich zwar beliebig verschachteln. Oftmals lässt sich eine Verschachtelung aber besser mit einem logischen Operator lösen. So ist obiges Beispiel auch mit dem logischen Operator **&&** (logisches UND) lösbar, da dort in der ersten if-Fallunterscheidung keine Anweisung eingefügt ist:

```
var a = 5;
if (a>4 && a<10) {
    trace(a);
}
```

else

Die optimale Ergänzung zu `if` ist `else`. `else` fängt alle Fälle ab, bei denen keine der vorherigen `if`-Anweisungen zutrifft. Deswegen kommt `else` auch nie ohne vorheriges `if` vor.

Die Syntax sieht folgendermaßen aus:

```
if (Bedingung) {  
    Anweisung;  
} else {  
    Anweisung  
}
```

Wir illustrieren das an einem kurzen Beispiel:

```
var a = 5;  
if (a<4) {  
    trace(a);  
} else {  
    trace("Bedingung nicht erfüllt");  
}
```

Wenn man das Beispiel vorläse, würde es sich in etwa so anhören: »Wenn `a` kleiner als 4, gib `a` aus. Ist diese Bedingung nicht erfüllt, gib »Bedingung nicht erfüllt« aus.« In unserem Fall ist die Bedingung nicht erfüllt, es wird also der Text der `else`-Anweisung ausgegeben.

Kurzformen

Auch die `else`-Anweisung lässt sich in Kurzform schreiben.

```
var a = 5;  
if (a<4) trace(a);  
else trace("Bedingung nicht erfüllt");
```

oder etwas länger:

```
if (a<4)  
    trace(a);  
else  
    trace("Bedingung nicht erfüllt");
```

Die Kurzform von `else` wird allerdings ebenso wie die `if`-Anweisung mit der Schaltfläche **Auto-Form** im **Aktionen**-Bedienfeld in die Langform umgewandelt.

else if

Mehrere if-Anweisungen hintereinander sind eigentlich kein Problem, allerdings wird dann jede Bedingung einzeln überprüft. Einen Ausweg bietet `else if`. `else if` ist eigentlich ein Mittelding zwischen `if` und `else`. Es enthält im Gegensatz zu `else` eine eigene Bedingung, wird aber nur geprüft, wenn nicht vorher schon eine if-Bedingung oder eine andere else-if-Bedingung eingetreten ist.

Die Syntax der else-if-Anweisung ist genau wie die mehrerer if-Anweisungen:

```
if (Bedingung) {
    Anweisung;
} else if (Bedingung) {
    Anweisung;
} else if (Bedingung) {
    Anweisung;
} else {
    Anweisung;
}
```

Wir zeigen Ihnen den Unterschied zu mehreren if-Anweisungen an einem Beispiel. Folgender Code wurde mit if-Anweisungen realisiert. Jede Bedingung wird überprüft. Die der dritten und der vierten if-Anweisung sind erfüllt, es werden also die Zahlen 3 und 4 ausgegeben.

```
var a = 5;
if (a<4) {
    trace(1);
} if (a>=7) {
    trace(2);
} if (a<6) {
    trace(3);
} if (a == 5) {
    trace(4);
} else {
    trace(5);
}
```

Wenn wir dieselben Bedingungen und Zahlen in einen Anweisungsblock mit else-if-Anweisungen packen, ergibt sich ein anderes Bild. Nachdem die dritte else-if-Bedingung erfüllt wurde, wird die Fallunterscheidung

verlassen. Die vierte wird nicht mehr geprüft. Es wird also auch nur 3 ausgegeben.

```
var a = 5;
if (a<4) {
    trace(1);
} else if (a>=7) {
    trace(2);
} else if (a<6) {
    trace(3);
} else if (a == 5) {
    trace(4);
} else {
    trace(5);
}
```

switch case

`switch case` ist eine Fallunterscheidung, die in manchen Fällen einfacher ist als ein `if-else-if-else`-Konstrukt. Dies gilt für alle Fälle, bei denen für eine Variable überprüft werden soll, welcher Wert zutrifft. Also beispielsweise, ob es April, Mai oder Juni ist.

Entsprechend beginnt die Syntax mit dem Schlüsselwort `switch`. Es enthält die Variable, die geprüft werden soll. In den `case`-Abschnitten werden Werte angegeben. Hat die Variable den Wert einer `case`-Anweisung, so werden deren Anweisungen ausgeführt.

```
switch (variable) {
case Wert1 :
    Anweisung;
    break;
case Wert2 :
    Anweisung;
    break;
default :
    Anweisung;
}
```

Auf zwei Besonderheiten wollen wir noch hinweisen. Zum einen sehen Sie am Ende die `default`-Anweisung. Sie ist mit `else` vergleichbar und kommt zum Einsatz, wenn keine der vorherigen Bedingungen erfüllt wurde.

Die zweite Besonderheit ist der Befehl `break`. Er besagt, dass die Fallunterscheidung verlassen werden soll. Man fügt ihn immer dann ein, wenn man nicht möchte, dass die übrigen Fälle noch überprüft werden, wenn davor schon einer zutrifft.

Im folgenden Beispiel verwenden wir die `switch-case`-Anweisung in einem Fall aus der Praxis, der gar nicht so selten vorkommt. In ActionScript erhält man das Tagesdatum des Rechners des Anwenders mithilfe der Klasse `Date`. Auch wenn Sie bisher noch nicht viel mit Klassen anfangen können: Diese Klasse enthält das Systemdatum. Auf dieses Datum können Sie mit unterschiedlichen Methoden zugreifen. Eine davon ist `getMonth()`. Sie liefert den Monat als Zahl. Die Zählung beginnt aber nicht bei 1, sondern bei 0. Das heißt 0 steht für Januar, 11 für Dezember.



Mehr zu Klassen und den mitgelieferten Klassen von ActionScript erfahren Sie im nächsten Kapitel.

0 bis 11 sind natürlich ein bisschen ungeschickt als Ausgabe des Monats. Wir wollen die Zahlen lieber in Monatsnamen umwandeln. Dafür eignet sich die `switch-case`-Anweisung hervorragend:

1. Zuerst benötigen wir ein Objekt der Klasse `Date`. Ok, Sie wissen nicht, was das ist, wichtig für unser Beispiel ist aber nur, dass man es mit dem Schlüsselwort `new` erhält.

```
var datum = new Date();
```

2. Mit der Methode `getMonth` laden wir den aktuellen Monat in die Variable `monat`.

```
var monat = datum.getMonth();
```

3. Diese Variable ist die Basis für die `switch-case`-Anweisung, da ihr Wert mit allen möglichen Werten für die zwölf Monate verglichen wird.

Hier sehen Sie den vollständigen Code.

```
var datum = new Date();
var monat = datum.getMonth();
switch (monat) {
case 0 :
    feld = "Januar";
    break;
case 1 :
    feld = "Februar";
    break;
```

```

case 2 :
    feld = "März";
    break;
case 3 :
    feld = "April";
    break;
case 4 :
    feld = "Mai";
    break;
case 5 :
    feld = "Juni";
    break;
case 6 :
    feld = "Juli";
    break;
case 7 :
    feld = "August";
    break;
case 8 :
    feld = "September";
    break;
case 9 :
    feld = "Oktober";
    break;
case 10 :
    feld = "November";
    break;
case 11 :
    feld = "Dezember";
    break;
default :
    feld = "Kein Monat!";
}

```

Listing 9.1 select-case-Fallunterscheidung (aus `monat.fla`)

Wir haben die Anweisung zum Testen in das erste Schlüsselbild eines Flash-Films gepackt und ein dynamisches Textfeld mit der Variablen `feld` hinzugefügt. In diesem Textfeld wird die Variable aus dem obigen Code ausgegeben.

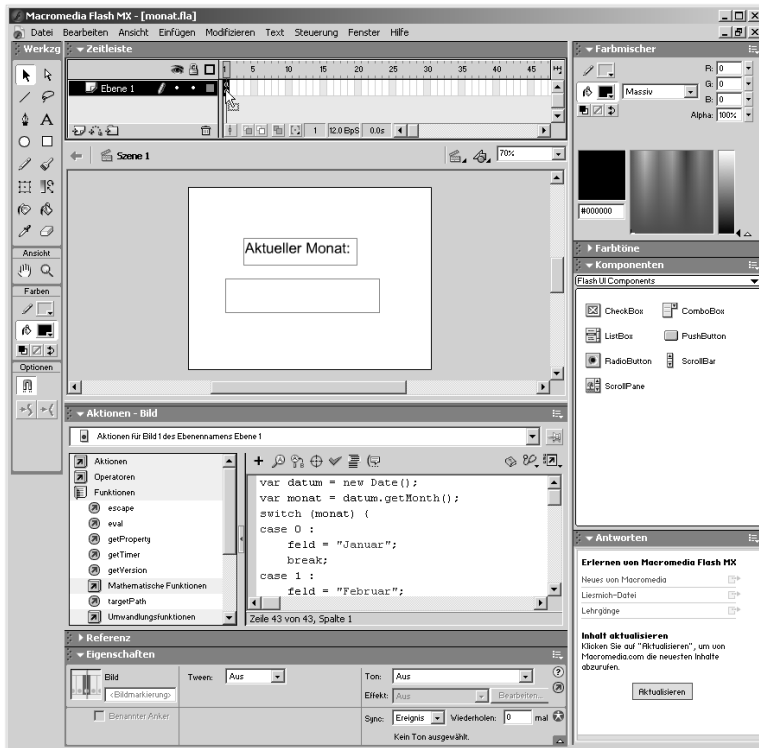


Abbildung 9.7 Die Anweisungen sind im ersten Schlüsselbild zu sehen.

Wenn Sie den Film testen, sehen Sie, dass der aktuelle Monat ausgegeben wird (siehe Abbildung 9.8). Die Datei zum Beispiel finden Sie unter dem Namen **monat.fla** auf der CD-ROM.



Abbildung 9.8 Dieses Beispiel entstand im April.

Eine switch-case-Anweisung lässt sich natürlich auch immer mit einer if-else-if-Fallunterscheidung lösen. Allerdings muss dann auch zwölfmal die Bedingung ausführlich mit `monat ==` geprüft werden. Das Beispiel finden Sie unter **monat_if fla** auf der CD-ROM.



```
if (monat == 0) {
    feld = "Januar";
}else if (monat == 1) {
    feld = "Februar";
}else if (monat == 2) {
    feld = "März";
}else if (monat == 3) {
    feld = "April";
}else if (monat == 4) {
    feld = "Mai";
}else if (monat == 5) {
    feld = "Juni";
}else if (monat == 6) {
    feld = "Juli";
}else if (monat == 7) {
    feld = "August";
}else if (monat == 8) {
    feld = "September";
}else if (monat == 9) {
    feld = "Oktober";
}else if (monat == 10) {
    feld = "November";
}else if (monat == 11) {
    feld = "Dezember";
}
```

9.3.2 Schleifen

Die zweite wichtige Art von Kontrollstrukturen sind Schleifen. Eine Schleife ist eine immer wiederkehrende Aktion, die so lange durchgeführt wird, solange eine vorgegebene Bedingung erfüllt ist. Keine Angst, gleich wird es konkreter.

while

Die while-Schleife ist die einfachste Form der Schleife. Ein Blick auf die Syntax verrät eigentlich schon alles:

```
while (Bedingung) {
    Anweisung;
}
```

Zuerst wird die Bedingung geprüft. Ist diese `true`, wird die Anweisung ausgeführt. Dann wird die Bedingung wieder geprüft. Ist sie erfüllt, ... und so weiter, bis die Bedingung nicht mehr erfüllt ist.

Wo die wirkliche Stärke von Schleifen steckt, zeigt schon ein einfaches Beispiel:

```
var i = 1;
while (i<=10) {
    trace(i);
    i++;
}
```

Dieses Beispiel gibt die Zahlen von 1 bis 10 aus (siehe Abbildung 9.9). Es zeigt gleich mehrere Besonderheiten der `while`-Schleife:

- ▶ Die Variable, deren Bedingung geprüft wird, wurde bereits vorher definiert.
- ▶ Die Variable `i` ist ein inoffizieller Standard für so genannte Zählervariablen. Sie werden innerhalb einer Schleife verändert, um einen Vorgang durchzuzählen.
- ▶ Hier wird ein Inkrement verwendet, um die Zählervariable um jeweils eins zu erhöhen.

In einer solchen oder einer ähnlichen Form werden uns Schleifen noch öfter begegnen, beispielsweise beim Auslesen von Arrays.



Abbildung 9.9 Die Zahlen von 1 bis 10

while-Schleifen können wie alle anderen Schleifen auch ineinander verschachtelt werden. Natürlich lassen sich auch andere Fallunterscheidungen und Anweisungen in die Schleife einfügen.



Endlosschleifen

Was fällt Ihnen an der folgenden Schleife auf?

```
var i = 1;
while (i<10) {
    trace(i);
}
```

Richtig, die Bedingung dieser Schleife ist immer erfüllt. Dies nennt man eine Endlosschleife. Eine solche Schleife entsteht beispielsweise immer dann, wenn die Zählervariable nicht verändert wird.

Nun gut, dann läuft die Schleife halt ewig, könnte man denken. Das Problem ist aber, dass dies einige Ressourcen verbraucht. Nicht jede Schleife ist so einfach wie die in unserem Beispiel. Und selbst die erfordert Rechenleistung. Deswegen sollten Sie in der Programmierung Endlosschleifen auf jeden Fall vermeiden.

Tritt doch einmal eine auf, wird der Flash-Player auf den Plan gerufen. Nach 15 Sekunden gibt er eine Meldung aus, ob man nicht gerne das Skript beenden würde (siehe Abbildung 9.10). Sie können zwar auch auf **Nein** klicken, das hilft aber bei einer Endlosschleife nicht viel, die Meldung kommt gleich wieder. Wenn der Anwender mit **Ja** bestätigt, werden alle Skripte, nicht nur die Schleife, beendet. Sie sehen, Endlosschleifen sollte man wirklich vermeiden.

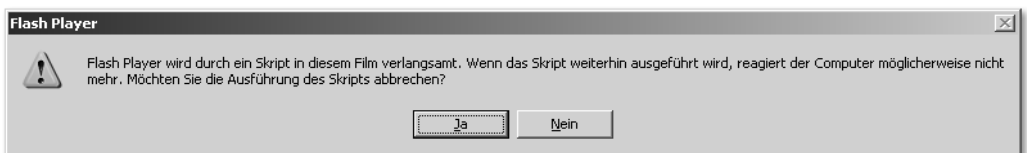


Abbildung 9.10 Der Flash-Player gibt eine Warnmeldung aus.

Wir haben noch nicht geklärt, wann der Flash Player die Meldung ausgibt. Es verstreichen etwa 15 Sekunden, bis die Meldung erscheint. Dies war auch schon im Flash Player 5 so. Allerdings sollte man mit einer normalen Schleife diese Zeitspanne so gut wie nie benötigen.



Der Flash Player 4 beschränkt die maximale Zahl der Schleifendurchläufe auf 200 000.

do while

Die do-while-Schleife ist eine enge Verwandte der while-Schleife. Sie hat nur eine Besonderheit: Sie prüft die Bedingung erst am Ende. Das heißt vorher wird auf jeden Fall die Schleife einmal durchlaufen.

```
do {  
    Anweisung;  
} while (Bedingung);
```

Dass die Anweisung mindestens einmal ausgeführt wird, ist an sich schon sehr bemerkenswert. Werfen Sie einen Blick auf das nächste Beispiel:

```
var a = 5;  
do {  
    trace(a);  
    a--;  
} while (a>6);
```

Hier wird eine Zählervariable mit einem Inkrement verwendet. Die Bedingung `a größer 6` ist überhaupt nicht erfüllt. Dennoch wird die Variable `a` einmal ausgegeben.

for

Bei der for-Schleife sind einige Einstellungen mehr vorgeschrieben als bei der while-Schleife. Im Prinzip funktioniert sie aber ähnlich, und welche man einsetzt, ist Geschmacksache.

Ein Blick auf die Syntax zeigt, dass die for-Schleife vor allem für den Umgang mit Zählervariablen geeignet ist:

```
for (Anfangswert; Bedingung; Änderung) {  
    Anweisung;  
}
```

Sie können für eine oder mehrere Variablen einen Anfangswert eingeben. Durch Strichpunkt getrennt folgt die Bedingung, die erfüllt sein muss, damit die Anweisung ausgeführt wird. Die Änderung ist die Wertänderung, die nach jedem Schleifendurchlauf durchgeführt wird.

Das Beispiel mit der Ausgabe der Zahlen von 1 bis 10 lässt sich mit einer for-Schleife sogar mit zwei Zeilen weniger Tippaufwand lösen:


```
for (var i = 1; i<=10; i++) {
    trace(i);
}
```

Zuerst wird die Zählervariable `i` definiert, dann folgt die Bedingung, anschließend das Inkrement. Die Anweisung ist die Ausgabe des aktuellen Wertes von `i`.

for in

Die for-in-Schleife liest alle Elemente aus einem Objekt aus.

```
for (var element in objekt) {
    Anweisung;
}
```

Gesprochen hört sich das so an: »Führe für jedes Element in diesem Objekt die Anweisung aus, bis kein Objekt mehr vorhanden ist«.

Um die Vorzüge der for-in-Schleife erklären zu können, müssen wir ein wenig vorgreifen. Sie dient hauptsächlich zum Auslesen von Arrays, die im nächsten Kapitel ausführlich behandelt werden. Hier legen wir aber schon einmal ein paar Grundlagen, da Ihnen Arrays schon aus dem JavaScript-Teil bekannt sind.

Ein Array ist eine Sammlung mehrerer Daten in einer Variablen. Diese Variable hat den Datentyp `Object`. Jeder Datensatz im Array hat einen Index, mit dem er identifiziert werden kann. Bei einem Array beginnt der Index normalerweise bei 0 und zählt in ganzen Zahlen nach oben.

Ein solches, sehr einfaches Array wollen wir nun auslesen.

```
var tage = ["Montag", "Dienstag", "Mittwoch", "Donnerstag"];
for (var tag in tage) {
    trace(tage[tag]);
}
```

Das Array selbst wird mit eckigen Klammern (`[]`) gekennzeichnet. Die einzelnen Elemente sind durch Kommata getrennt. In unserem Fall sind es einzelne Strings mit ein paar Wochentagen.

Die for-in-Schleife nimmt in die Variable `tag` in jedem Schleifendurchlauf ein Element des Arrays `tage` auf. Die Anweisung mit der Funktion `trace()` gibt es dann aus. Dies geschieht so lange, bis in `tage` kein Element mehr vorhanden ist.



Abbildung 9.11 Die Wochentage wurden ausgegeben.

In Abbildung 9.11 sehen Sie das Ergebnis. Wir werden die for-in-Schleife noch genauer kennen lernen. Nur so viel sei schon verraten: Sie kann nicht nur Arrays auslesen, sondern auch die Eigenschaften von Objekten liefern oder verändern. Eine sehr nützliche Funktion, der Sie schon im nächsten Kapitel begegnen werden.

9.4 Beispielanwendung – Taschenrechner

Diese Beispielanwendung verwendet die ActionScript-Grundlagen aus diesem Kapitel und zeigt Ihnen, wie Sie mit den einfachen Mitteln, die Sie bis jetzt kennen gelernt haben, schon eine ganze Menge erreichen können.

Ziel ist es, einen einfachen Taschenrechner zu erzeugen. Folgendes soll er beherrschen:

- ▶ die Grundrechenarten Addition, Subtraktion, Multiplikation und Division,
- ▶ Modulo,
- ▶ mehrere Rechenoperationen hintereinander.

Der Taschenrechner muss allerdings keine Gleitkommazahlen unterstützen. Auch auf komplexere mathematische Funktionen und die Implementierung von »Punkt vor Strich« wird verzichtet.

9.4.1 Vorbereitungen – die Flash-Datei

Wir steigen in das Beispiel mit ein wenig Optik ein und zeigen Ihnen, wie der Taschenrechner gemacht wurde. Für alle, die ein wenig zu faul sind, selbst in Flash zu basteln: Sie finden den Taschenrechner noch völlig ohne ActionScript in der Datei **taschenrechner_only fla** auf der CD-ROM.

Für alle, die vom Arbeiten in Flash noch nicht genug haben, gibt es hier eine Schnellanleitung, wie er erstellt wurde. Aber werfen Sie doch, bevor wir starten, einen Blick auf Abbildung 9.12. Dort sehen Sie das Ergebnis, das wir erreichen möchten.

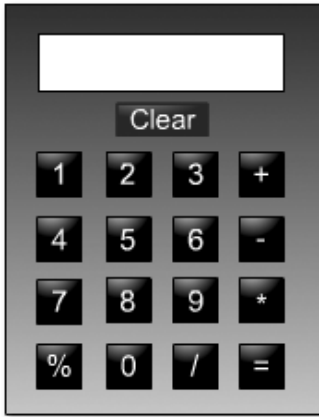


Abbildung 9.12 Der fertige Taschenrechner

1. Noch ein kurzer Hinweis zur Zeitleiste dieses Beispiels: Die ganze Aktion findet im ersten Schlüsselbild und später hauptsächlich in ActionScript statt.
2. Zunächst benötigen wir den Hintergrund und den Schatten. Beiden haben wir jeweils eine eigene Ebene spendiert. Der Schatten besteht aus einem grauen Rechteck (RGB: (204, 204, 204), entspricht hexadezimal #CCCCCC). Der Hintergrund, also der Korpus des Taschenrechners, ist ein Rechteck mit schwarzem Rahmen und einem linearen Verlauf von hellem Grau (RGB: (218, 218, 218), hexa: #DADADA) zu dunklem Grau (RGB: (27, 27, 27), hexa: #1B1B1B). Wir haben den Verlauf mit dem Füllungstransformations-Werkzeug ein wenig angepasst. Der Schatten liegt leicht versetzt auf einer Ebene unter dem Korpus (siehe Abbildung 9.13).
3. Nun benötigen wir eine vierte Ebene. Wir haben sie **Taschenrechner** genannt. Sie enthält die Tasten und das Ausgabefeld des Taschenrechners. Die Tasten sind alle von einer Grundform abgeleitet, die wir zuerst erstellt haben. Die Grundform ist ein Rechteck mit schwarzem Rand und einem radialen Verlauf als Füllung. Der Verlauf geht von Weiß nach Schwarz. Sein Zentrum haben wir mit dem Füllungstransformations-Werkzeug nach links oben verschoben, so dass der Button ein dreidimensionales Aussehen hat.

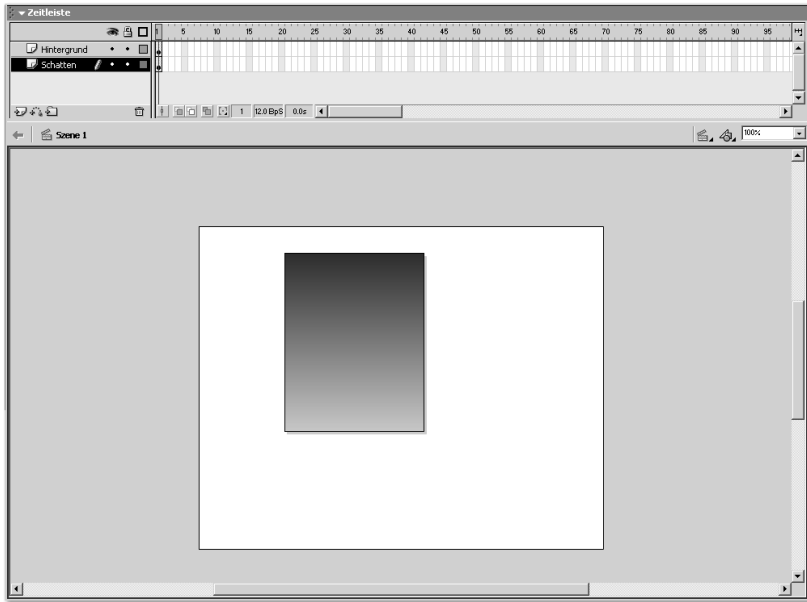


Abbildung 9.13 Hintergrund und Schatten

4. Den Button konvertieren wir anschließend in ein Schaltflächen-Symbol namens `button`. Dieses eine Symbol ist die Grundlage für alle Tasten auf dem Taschenrechner. Das spart Speicherplatz und ist praktisch.
5. Nun bearbeiten wir das Symbol noch ein wenig. Wir fügen für den Zustand **Drücken** ein Schlüsselbild ein und verschieben den Farbverlauf für dieses Schlüsselbild in die Mitte des Buttons, so dass er gedrückt aussieht (siehe Abbildung 9.14).
6. Nun ist das Schaltflächen-Symbol betriebsbereit. Ziehen Sie es dreizehnmal aus der Bibliothek auf die Bühne, um die dreizehn für die Tasten benötigten Instanzen zu erzeugen.
7. Die folgende Arbeit ist recht stupide. Zuerst müssen Sie die Instanzen mit weißem statischem Text beschriften. Wir haben Arial in der Größe 16 pt verwendet.
8. Anschließend müssen Sie die Tasten aneinander ausrichten. Dabei leistet das Bedienfeld **Ausrichten** gute Dienste. Sie benötigen die beiden Befehle **Vertikal zentrieren** und **Horizontal zentrieren**. Ein Beispiel zum vertikalen Zentrieren sehen Sie in Abbildung 9.15.



Abbildung 9.14 Der Zustand **Drücken**

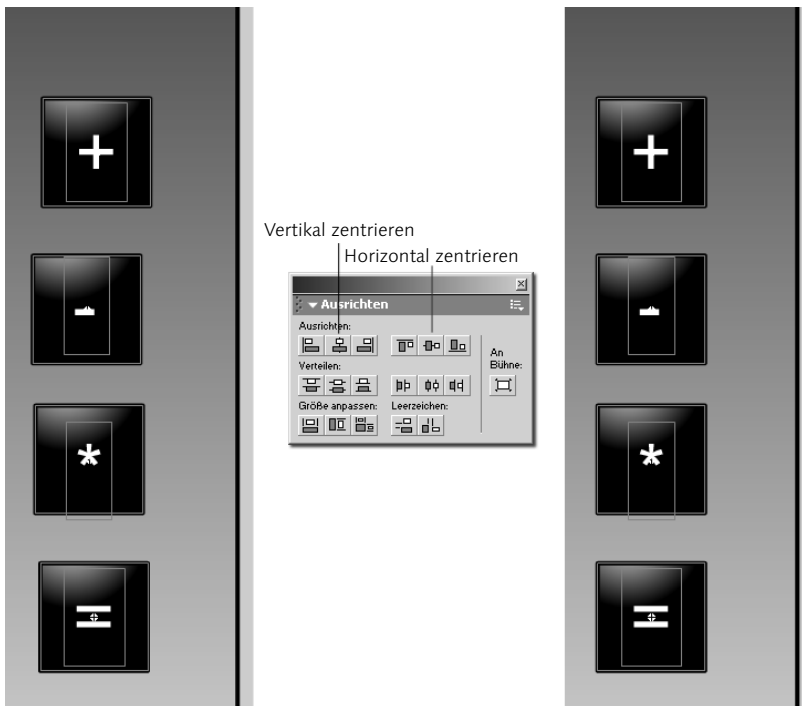


Abbildung 9.15 Vertikal ausrichten

9. Zuletzt kümmern wir uns um die Clear-Taste. Sie soll etwas flacher, dafür aber breiter sein als die übrigen Schaltflächen. Dies erledigen wir mit dem Frei-transformieren-Werkzeug. Die Farbe wollen wir auch ein wenig verändern. Dazu ändern wir die Option **Farbe** im Eigenschafteninspektor auf Farbtön. Als Farbe wählen wir ein dunkles Rot (RGB: (153, 0, 0), hexa: #990000). Die Deckkraft wird auf 50% heruntergeschraubt.



Abbildung 9.16 Die Einstellungen für die Clear-Schaltfläche

10. Das letzte Element ist ein dynamisches Textfeld. Es ist sozusagen das Ausgabefenster unseres Taschenrechners. Die Variable, mit der wir per ActionScript auf das dynamische Textfeld zugreifen können, nennen wir `feld`. Wir schalten als Option **Rahmen um Text zeigen** im Eigenschafteninspektor ein und verwenden einen weißen Hintergrund mit schwarzem Rahmen. Die Ausrichtung des Textfeldes stellen wir auf rechtsbündig. Es soll außerdem einzellig bleiben, damit die Zahlen von rechts nach links in einer Zeile angezeigt werden. Schriftart ist Arial, Schriftgröße 24 pt.

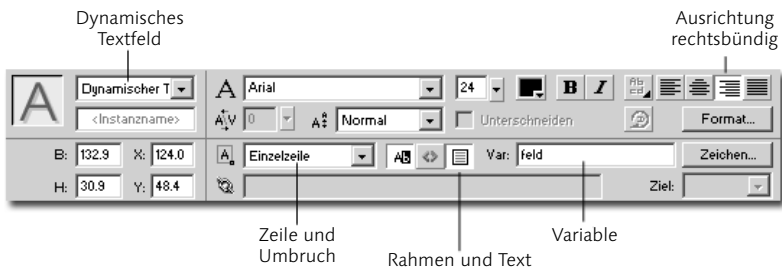


Abbildung 9.17 Die Einstellungen für das dynamische Textfeld

11. Eine vierte Ebene heißt ActionScript und nimmt später die Variablen-deklarationen für unser Beispiel auf.

Wenn Sie noch nicht so viel Flash-Erfahrung mitbringen, war diese Beschreibung sicher zu kurz für Sie. Wir wollen hier allerdings nicht weiter ins Detail gehen, um mehr Platz für ActionScript zu haben. Wir empfehlen Ihnen deswegen, bei Flash-Fragen entweder einen Blick in die Hilfe von Flash zu werfen oder ein gutes Flash-Buch neben unseres zu legen, z.B. Das Flash-Werk von Sascha Wolter, ebenfalls bei Galileo Press erschienen.



9.4.2 Variablendeklaration

Am Ende des vorangegangenen Abschnitts haben Sie noch eine vierte Ebene erstellt. Das erste Schlüsselbild dieser Ebene soll die nötigen Variablendeklarationen für unser Beispiel aufnehmen.

```
var erg = "";  
var zahl1 = "";  
var zahl2 = "";  
var op = "";
```

Es gibt hier insgesamt vier Variablen. Die Variable `erg` soll später beim Anklicken des Gleichheitszeichens das Ergebnis der Rechnung aufnehmen, die Variablen `zahl1` und `zahl2` die beiden Zahlen, mit denen gerechnet werden soll. `op` ist die Variable für den Operator, der vom Anwender angeklickt wurde.

Die Variable `feld`, mit der wir den Inhalt des dynamischen Textfelds, unseres Ausgabefensters, ändern können, ist übrigens schon mit der Angabe des Namens im Eigenschafteninspektor deklariert.

9.4.3 Die Zifferntasten

Bevor Sie die Tastatur zur Hand nehmen, überlegen Sie kurz, was die Zifferntasten eigentlich leisten müssen. Nun, die Antwort ist einfach: Sie müssen nur die entsprechende Ziffer in das Textfeld eintragen. Richtig, für die 1 geht das folgendermaßen:

```
on (release) {  
    feld += "1";  
}
```

`on (release)` fängt das Ereignis ab, wenn der Anwender die Maus loslässt. In diesem Fall wird zum Ausgabefeld einfach die Ziffer 1 hinzugefügt.

Die anderen Ziffern übergeben analog ebenfalls ihre Ziffern an das Ausgabefeld. Für die Fünf lautete das Skript beispielsweise:

```
on (release) {  
    feld += "5";  
}
```

Dasselbe Verfahren gilt also für alle Ziffern. Wirklich für alle Ziffern? Nein, eine kleine Ausnahme gibt es. Da wir keine Eingabe von Gleitkommazahlen zulassen, sollte die 0 keiner Zahl vorangestellt werden. Dies passiert aber immer, wenn die 0 in ein leeres Textfeld eingegeben wird. Wir verhindern dies mit einer einfachen Fallunterscheidung. Sie trägt die 0 nur ein, wenn das Textfeld nicht leer ist oder nur aus einer 0 besteht (die beispielsweise durch Subtraktion entstanden sein kann).

```
on (release) {  
    if (feld != "" && feld != "0") feld += "0";  
}
```

9.4.4 Operatoren

Die Operatoren sind schon ein wenig schwieriger als die Ziffern. Was soll passieren, wenn der Anwender auf einen Operator drückt? Die Zahl, die aktuell im Ausgabefenster steht, soll zur ersten Zahl der Operation werden und der Operator soll festgelegt werden.

Um das komplikationsfrei zu erreichen, sind zwei if-Anweisungen nötig:

- In der ersten wird geprüft, ob das Ausgabefeld leer ist oder den Wert NaN hat, also keine Zahl ist. Nur wenn dies nicht der Fall ist (!= bzw. logischer Operator NICHT), wird der Wert des Feldes als `zahl1` gespeichert.

```
on (release) {  
    if (feld != "" && !(feld.isNaN)) zahl1 = feld;
```

- Anschließend wird das Feld geleert, damit der Anwender eine neue Zahl eingeben kann.

```
    feld = "";
```

- Nun folgt die zweite Fallunterscheidung. Wenn `zahl1` kein leerer String ist, also einen Wert besitzt, wird der Operator als Zeichenkette festgelegt. Wir zeigen dies beispielhaft für das +-Symbol.


```

        if (zahl1 != "") op = "+";
    }

```

Nun müssen Sie nur noch dieses kurze Skript für alle Operatoren einfügen. Für das `*`-Symbol sähe das beispielsweise so aus:

```

on (release) {
    if (feld != "" && !(feld.isNaN)) zahl1 = feld;
    feld = "";
    if (zahl1 != "") op = "*";
}

```

9.4.5 Clear

Die Schaltfläche Clear übernimmt eine Sonderrolle. Sie stellt alle Variablen zurück und leert das Ausgabefenster.

```

on (release) {
    feld = "";
    op = "";
    zahl1 = "";
    zahl2 = "";
}

```

9.4.6 Ausrechnen

Jetzt geht es ans Ausrechnen. Dies geschieht im `on (release)`-Ereignis der Gleichheitszeichen-Schaltfläche.

Da wir hier mehrere ineinander verschachtelte `if`-Anweisungen und eine `switch-case`-Konstruktion verwenden, beginnen wir nach dem Zwiebelschalenprinzip von innen nach außen.

- Zuerst überprüfen wir, ob es die Variable `zahl1` überhaupt gibt. Wenn ja, werden die Anweisungen ausgeführt, die wir im nächsten Punkt besprechen, wenn nein, dann wird sowohl das Ergebnis als auch der Operator auf einen leeren String gesetzt. Wir gehen hier sehr hart mit Fehleingaben des Anwenders um, um Fehlermeldungen wie `NaN` (Not a Number) zu vermeiden.

```

if (zahl1 != "") {
    //Anweisung folgt
} else {
    erg = "";
}

```

```

        op = "";
    }

```

- Im Allgemeinen wird `zahl1` definiert sein, die Bedingung also zutreffen. Wir müssen nun noch prüfen, ob es `zahl2` gibt. Diese müsste beim Anklicken des Gleichheitszeichens im Ausgabefenster stehen. Des weiteren müssen wir prüfen, ob in `op` ein Operator existiert. Dazu koppeln wir beide Prüfungen mit einem logischen Operator – fertig ist die `if`-Anweisung.

Wird die eben aufgestellte `if`-Bedingung erfüllt, so folgen weitere Anweisungen, die wir im nächsten Punkt besprechen. Vorher regeln wir aber noch, was passiert, wenn die Bedingungen nicht erfüllt werden. Dann kommt die `else`-Anweisung ins Spiel. Sie legt fest, dass die Ergebnis-Variable `erg` den Wert von `zahl1` erhält und diesen ausgibt. Sie erinnern sich: `zahl1` muss an dieser Stelle definiert sein, da wir das bereits in der äußeren `if`-Anweisung geprüft haben. Um den Wert nicht zu verlieren, geben wir ihn einfach wieder aus. Der Anwender kann dann damit weiter rechnen.

```

if (feld != "" && op != "") {
    //Anweisung folgt
} else {
    erg = zahl1;
}

```

- Wie versprochen, kommen jetzt die Anweisungen, was passiert, wenn sowohl `zahl1` als auch `feld` und der Operator vorliegen. Zuerst machen wir `zahl1` mit der bereits bekannten Funktion `Number` zu einer Zahl. Dann lesen wir `zahl2` aus dem Ausgabefenster aus und verwandeln die Variable ebenfalls in eine Zahl.

```

zahl1 = Number(zahl1);
zahl2 = Number(feld);

```

- Nun haben wir beide Zahlen und können rechnen. Mit einer `switch`-case-Fallunterscheidung überprüfen wir die verschiedenen Werte für `op`. Je nachdem, welcher Operator in `op` als String vorhanden ist, verwenden wir zur Rechnung einen arithmetischen Operator und weisen das Ergebnis der Variablen `erg` zu. In jede `case`-Anweisung fügen wir außerdem die Funktion `break` ein, damit die Fallunterscheidung verlassen wird, wenn wir den richtigen Operator gefunden haben.

```

switch (op) {
    case "+" :

```

```

        erg = zahl1+zahl2;
        break;
    case "-" :
        erg = zahl1-zahl2;
        break;
    case "*" :
        erg = zahl1*zahl2;
        break;
    case "/" :
        erg = zahl1/zahl2;
        break;
    case "%" :
        erg = zahl1%zahl2;
        break;
    default :
        erg = "";
}

```

Die Unterscheidung der Operatoren lässt sich alternativ auch mit einem if-Anweisungsblock realisieren. Was man schließlich bevorzugt, ist Geschmacksache, allerdings ist diese Anwendung ein Paradebeispiel für switch case. Hier sehen Sie den Code mit if:



```

if (op == "+") {
    erg = zahl1+zahl2;
} else if (op == "-") {
    erg = zahl1-zahl2;
} else if (op == "*") {
    erg = zahl1*zahl2;
} else if (op == "/") {
    erg = zahl1/zahl2;
} else if (op == "%") {
    erg = zahl1%zahl2;
} else {
    erg = "";
}

```

- Unser Ergebnis ist jetzt in der Variablen `erg` gespeichert, aber noch nicht ausgegeben. Wir verwandeln es deswegen mit der Funktion `toString` in einen String und schreiben es in unser dynamisches Textfeld, unser Ausgabefenster.

```
erg.toString;  
feld = erg;
```

- Nun stellen wir noch die Variablen zurück. `op`, `zahl2` und `erg` werden auf einen leeren String zurückgesetzt. `zahl1` wird noch das Ergebnis der Rechnung zugewiesen.

```
op = "";  
zahl2 = "";  
zahl1 = erg;  
erg = "";
```

Im Folgenden finden Sie den kompletten Code, damit Sie sich einen Überblick verschaffen können:

```
on (release) {  
    if (zahl1 != "") {  
        if (feld != "" && op != "") {  
            zahl1 = Number(zahl1);  
            zahl2 = Number(feld);  
            switch (op) {  
                case "+" :  
                    erg = zahl1+zahl2;  
                    break;  
                case "-" :  
                    erg = zahl1-zahl2;  
                    break;  
                case "*" :  
                    erg = zahl1*zahl2;  
                    break;  
                case "/" :  
                    erg = zahl1/zahl2;  
                    break;  
                case "%" :  
                    erg = zahl1%zahl2;  
                    break;  
                default :  
                    erg = "";  
            }  
        } else {  
            erg = zahl1;  
        }  
    }  
}
```

```

    } else {
        erg = "";
        op = "";
    }
    erg.toString();
    feld = erg;
    op = "";
    zahl2 = "";
    zahl1 = erg;
    erg = "";
}

```

Listing 9.2 Das Skript für das Gleichheitszeichen (aus **taschenrechner fla**)

In Abbildung 9.8 sehen Sie Schritt für Schritt, wie eine Addition mit dem Taschenrechner durchgeführt wird. Man könnte dem Taschenrechner natürlich noch viele andere Funktionen, Formeln und Fähigkeiten hinzufügen, aber auch das Bisherige hat sicherlich gereicht, Ihnen das in diesem Kapitel Erlernte noch einmal in Erinnerung zu rufen und Ihnen zu zeigen, was Sie schon mit relativ wenigen Mitteln in ActionScript erreichen können.

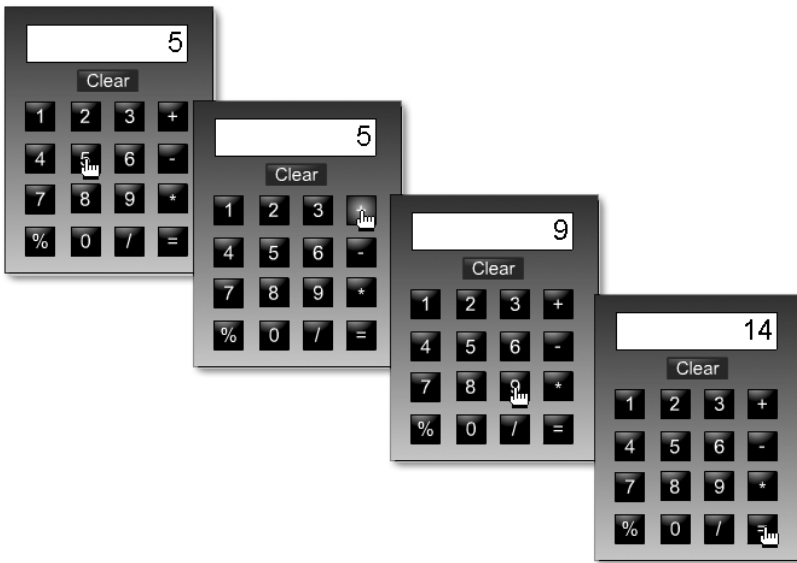


Abbildung 9.18 Eine einfache Addition im Zusammenschnitt

10 Fortgeschrittene Programmierung

Fortschritt wäre wunderbar – wenn er einmal aufhören würde.

– Robert Musil

Das Fundament ist im vorangegangenen Kapitel gelegt worden. Dieses Kapitel baut darauf auf und erweitert Ihre Kenntnisse um fortgeschrittene Programmierkonzepte. Sie lernen Funktionen kennen und werden herausfinden, wie man sowohl eigene Funktionen nutzt als auch mit vordefinierten in Flash arbeitet. Außerdem werden Arrays als komplexer Datentyp näher beschrieben.

10.1 Funktionen

Funktionen sammeln Funktionalität (deswegen der Name ☺), die man dann beliebig oft aufrufen kann. Denken Sie beispielsweise an eine Währungsumrechnung von DM in Euro. Sie benötigen sie an mehreren Stellen im Flash-Film. Kein Problem: Schreiben Sie statt Copy-and-Paste einfach eine Funktion und rufen Sie sie jedes Mal auf, wenn Sie die Umrechnung benötigen. Das hat große Vorteile:

- ▶ Der Code wird strukturierter.
- ▶ Änderungen in der Funktionalität können an zentraler Stelle durchgeführt werden.
- ▶ Eine Funktion lässt sich beliebig oft aufrufen.
- ▶ Sie müssen sich um den Inhalt der Funktion nicht mehr kümmern, nachdem Sie sie einmal erstellt haben.

Bisher haben wir uns mit eigenen Funktionen beschäftigt. ActionScript stellt allerdings auch eine Reihe von vorgefertigten Funktionen zur Verfügung. Diesen ist der Abschnitt 10.1.5, »Vorgefertigte Funktionen«, gewidmet.

10.1.1 Eigene Funktionen

Eine Funktion wird immer mit dem Schlüsselwort `function` definiert. Dann folgt sie folgender Syntax:

```
function Name () {  
    Anweisungen;  
}
```

Nach `function` wird der Funktionsname angegeben. Er folgt denselben Konventionen wie ein Variablenname. Wir haben die Regeln hier noch einmal in aller Kürze zusammengestellt. Ausführlicher finden Sie sie in Kapitel 9, *ActionScript-Grundlagen*, im Abschnitt »Namenskonventionen«.

- ▶ Ein Funktionsname darf nur mit einem Unterstrich (`_`) oder einem Buchstaben beginnen.
- ▶ Ein Funktionsname darf nach dem ersten Buchstaben nur Unterstriche, Buchstaben und Zahlen enthalten.
- ▶ Verboten sind Umlaute, Sonderzeichen, Leerzeichen, Punkte und Kommata.
- ▶ Ein Funktionsname darf keinem Schlüsselwort in *ActionScript* entsprechen.
- ▶ Bei gleichnamigen Funktionen überschreibt die später definierte die zuerst definierte. Sie sollten Gleichnamigkeit also vermeiden.

Die Anweisungen einer Funktion werden dann in geschweiften Klammern geschrieben.

Beginnen wir mit einem ganz einfachen Beispiel:

```
function ausgabe() {  
    trace("Eine Ausgabe");  
}
```

Diese Funktion zeigt immer, wenn sie aufgerufen wird, den Text »Eine Ausgabe« im Ausgabefenster an. Der Aufruf ist ebenfalls sehr einfach. Es reicht folgende Zeile:

```
ausgabe();
```

Der Aufruf kann vor oder nach der Funktion stehen. Sobald ein Funktionsaufruf erfolgt, sucht der Interpreter automatisch nach der gleichnamigen Funktion. Nehmen wir an, Sie schreiben folgenden Code in ein Schlüsselbild:

```
ausgabe();  
function ausgabe() {  
    trace("Eine Ausgabe");  
}
```

Die Ausgabe funktioniert, weil sich der Interpreter die passende Funktion innerhalb der Zeitleiste sucht.

Eine Funktion, die mit `function` so wie in diesem Beispiel definiert wurde, gilt nämlich für die Zeitleiste, in der sie vorkommt. Mehr dazu erfahren Sie im Abschnitt »Gültigkeitsbereich«.

Die runden Klammern am Ende des Funktionsnamens müssen sowohl bei der Definition als auch beim Aufruf vorhanden sein. Vergessen Sie die runden Klammern beim Aufruf, wird die Funktion nicht ausgeführt, es wird aber auch keine Fehlermeldung ausgegeben, da der Funktionsname als undeklarierte Variable behandelt wird. Denken Sie also bei der Fehlersuche daran!



Funktionen als Variablenwerte

Es gibt noch eine andere Art, Funktionen zu deklarieren. Das überrascht Sie nicht? Zugegeben, für fast alles gibt es in ActionScript mehrere Wege. Die zweite Art der Funktionsdeklaration ist allerdings schon bemerkenswert.

Die Funktion wird als Wert einer Variablen deklariert. Dies nennt man auch ein Funktionsliteral. Dabei wird der Funktionsname überflüssig, da die Variable ja schon einen Namen hat.

```
var ausgabe = function () {trace("Eine Ausgabe");};
```

Obiger Code erzeugt die gleiche Funktion wie im vorangegangenen Abschnitt, nur eben als Variablenwert.

Eine Variable, die eine Funktion enthält, hat den Datentyp `Object`.

Der Aufruf bleibt ebenfalls der gleiche:

```
ausgabe();
```



Normalerweise kann der Aufruf einer Funktion auch erst nach ihrer Definition erfolgen. Das funktioniert bei einer Funktion als Variable nicht. Hier muss zuerst die Variable definiert sein, dann erst darf der Funktionsaufruf folgen.



Gültigkeitsbereich von Funktionen

Wenn wir vom Gültigkeitsbereich sprechen, meinen wir zwei Dinge. Zum einen den Gültigkeitsbereich der gesamten Funktion, zum anderen den Gültigkeitsbereich der Variablen, der Thema des nächsten Abschnitts ist.

Eine Funktion gilt immer für die Zeitleiste, in der sie definiert wurde. Wenn eine Funktion also beispielsweise in einem Movieclip definiert wurde, gilt sie für dessen Zeitleiste.

Wenn Sie auf die Funktion aus einem anderen Movieclip oder einem anderen Film zugreifen möchten, müssen Sie diese adressieren.

```
_root.ausgabe()
```

ruft beispielsweise aus der Hauptzeitleiste (`_root`) die Funktion `ausgabe()` auf. Mehr zum Adressieren von Funktionen und Elementen über Zeitleisten hinweg erfahren Sie in Kapitel 12, Movieclips, Verweis prüfen im Abschnitt »Movieclips«.

Es kann aber manchmal schon recht umständlich sein, Funktionen aus irgendeinem Movieclip in irgendeinem anderen Movieclip aufzurufen. Für Funktionen, die man wirklich überall braucht, gibt es das Schlüsselwort `_global`. Damit werden sie zu globalen Funktionen, sind also von überall her direkt über ihren Namen adressierbar.

```
_global.ausgabe = function() {  
    trace("Eine Ausgabe");  
}
```

Der Aufruf aus einem anderen Movieclip ist ganz einfach:

```
ausgabe();
```



Globale Funktionen gibt es erst seit Flash MX. Wollen Sie also Ihre Datei im Flash 5-Format speichern, müssen Sie auf globale Funktionen verzichten, was ziemlich schade ist.

Gültigkeitsbereich von Variablen

Eine Variable gilt normalerweise immer für eine Zeitleiste. Variablen, die in Funktionen deklariert werden, sind aber so genannte lokale Variablen. Sie existieren nur innerhalb der Funktion. Ist die Funktion beendet, sind auch die Variablen verloren.

Ein Beispiel:

```
function variable() {  
    var a = 5;  
    trace(a);  
}
```

```
variable();  
trace(a);
```

Im obigen Beispiel wird eine Variable erst innerhalb einer Funktion definiert, dann innerhalb der Funktion ausgegeben. Die Funktion wird von außen aufgerufen, wodurch die Variable einmal ausgegeben wird. Die zweite Ausgabe der Variablen scheitert, da die Variable nur innerhalb der Funktion gilt (siehe Abbildung 10.1).

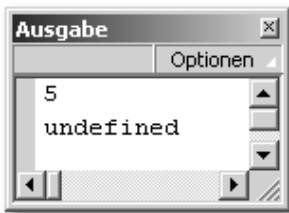


Abbildung 10.1 Die Variable ist außerhalb der Funktion nicht definiert.

Eine Variable, die außerhalb einer Funktion definiert wurde, gilt hingegen auch innerhalb der Funktion. Folgendes Skript gibt also den Wert 5 aus:

```
var a = 5;  
function variable() {  
    trace(a);  
}  
variable();
```

Ist eine Variable nicht außerhalb einer Funktion deklariert, wird aber auch nicht innerhalb der Funktion definiert, so wird sie automatisch deklariert. Ihr Gültigkeitsbereich ist dann nicht nur die Funktion, sondern die gesamte Zeitleiste.

```
function variable() {  
    a = 5;  
    trace(a);  
}  
variable();  
trace(a);
```

Obiger Code gibt also zweimal 5 aus (siehe Abbildung 10.2). Wem das etwas merkwürdig vorkommt, der hat nicht Unrecht. Sauberer Stil ist diese Art der Programmierung auf keinen Fall. Deswegen raten wir auch ausdrücklich davon ab.

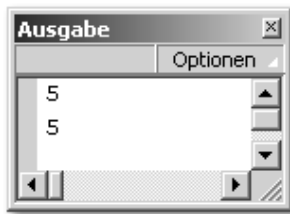


Abbildung 10.2 Die Variable gilt für die Zeitleiste.

Soll eine Variable nicht nur für eine Zeitleiste, sondern für alle Zeitleisten gelten, so kann sie ebenfalls wie eine Funktion mit dem Schlüsselwort `_global` definiert werden.

```
_global.variablenname = "Wert";
```

Obiger Code erzeugt eine globale Variable mit dem Namen `variablenname` und dem String »Wert« als Wert.



Auch globale Variablen gibt es erst ab Flash MX. In früheren Flash Playern steht diese Funktion nicht zur Verfügung.

10.1.2 Parameter

Bisher haben Sie nur gesehen, dass Funktionen Anweisungen ausführen können. Ansonsten waren Funktionen in den bisherigen Beispielen in sich völlig geschlossene Gebilde. Das sind sie natürlich nicht. Funktionen können Werte von außen übernehmen und auch selbst Ergebniswerte weitergeben. Um dies zu realisieren, gibt es Parameter.

```
function name(Parameter1, Parameter2, Parameter3) {
    Anweisungen;
}
```

Die Parameter werden einfach zwischen die runden Klammern eingetragen. Mehrere Parameter trennt man mit Kommata.

Lassen Sie uns die Parameter gleich mit einem Beispiel testen. Wir wollen eine Funktion schreiben, die die Fakultät einer Zahl bildet, die wir als Parameter übergeben.



Wenn Sie in der Schule in Mathematik nicht so gut aufgepasst haben, ist das kein Problem. Die Fakultät einer Zahl ist das Produkt aller ganzen Zahlen, die kleiner oder gleich dieser Zahl sind. Die Fakultät von 4 ist also beispielsweise $4 * 3 * 2 * 1 = 24$.

Wir beginnen mit der Definition einer Variablen mit dem Wert der Zahl, deren Fakultät wir ausgeben möchten. Anschließend rufen wir die Funktion auf und übergeben die definierte Zahl als Parameter.

```
var a = 10;
fakultaet(a);
```

Die Funktion selbst übernimmt den Parameter. Er muss dabei übrigens nicht denselben Variablennamen haben. Im Funktionsaufruf muss noch nicht einmal eine Variable vorkommen, ein Wert reicht völlig. Sie könnten mit `fakultaet(10)`; auch einfach die Fakultät von 10 ausgeben lassen.

```
function fakultaet(a) {
    var erg = a;
    while (a>1) {
        a--;
        erg = erg * a;
    }
    trace(erg);
}
```

In der Funktion wird sodann eine zweite Variable `erg` definiert. Sie soll die einzelnen Zwischenstände und am Schluss das Ergebnis unserer Rechnung aufnehmen.

Die eigentliche Rechnung findet in einer `while`-Schleife statt. Hier wird zuerst `a` um eins verringert (Dekrement), dann wird `erg` mit `a` multipliziert und daraus ein neues `erg` gebildet. Die Schleife wird durchlaufen, bis `a` nicht mehr größer als 1 ist. Anschließend wird das Ergebnis ausgegeben.

Besonderheiten

Eine Funktion kann mehrere Parameter haben. In ActionScript müssen beim Aufruf noch nicht einmal alle Parameter übergeben werden. Das folgende Skript löst also keine Fehlermeldung aus.

```
var a = 2;
var b = 3;
summe(a,b);

function summe(a, b, c) {
    var summe = a + b;
    trace(summe);
}
```

Bei diesem Beispiel würde es auch keinen Fehler ergeben, wenn der dritte Parameter in der Rechnung verwendet würde:

```
var a = 2;
var b = 3;
summe(a,b);

function summe(a, b, c) {
    var summe = a + b + c;
    trace(summe);
}
```

Das ist aber eine Besonderheit der Addition und wird in den meisten Fällen anders sein. Der nicht mit übergebene Parameter wird nämlich automatisch auf `undefined` gesetzt und `undefined` wird ja, wie Sie bereits wissen, in Rechnungen als 0 angesehen.

Der ActionScript-Interpreter zeigt sich außerdem auch großzügig, wenn zu viele Parameter übergeben werden. In dem folgenden Beispiel spuckt er ebenfalls das richtige Ergebnis 5 aus.

```
var a = 2;
var b = 3;
var c = 4;
summe(a, b, c);

function summe(a, b) {
    var summe = a + b;
    trace(summe);
}
```



Wir hatten in diesem Abschnitt bei unseren drei Beispielen enorm viel Glück. Der ActionScript-Interpreter ist zwar sehr gnädig, dennoch ist es nicht sehr sauber, mehr oder weniger als die festgelegte Zahl an Parametern zu übergeben. Außerdem verwirrt es andere Entwickler, die sich mit solchem Fleckerlteppich-Code auseinander setzen müssen.

Werte zurückgeben

Funktionen sind schon sehr praktisch: Sie legen einmal eine Funktion für die Berechnung von Fakultäten (oder irgendetwas anderem) an und rufen Sie mit wechselnden Parametern auf. Wenn die Funktion nur irgendetwas ausgeben soll, reicht das auch. Wenn sie aber auch einen Wert zurücklie-

fern muss, benötigen wir einen weiteren Befehl: `return`. Ein Blick auf die Syntax verrät sofort, wie `return` funktioniert:

```
return wert;
```

`return` gibt einen beliebigen Wert zurück. Wenn wir unser Beispiel aus dem vorangegangenen Abschnitt etwas abwandeln, können wir also die Fakultät als Variable zurückgeben und mit dem Wert weiterarbeiten.

```
function fakultaet(a) {  
    var erg = a;  
    while (a>1) {  
        a--;  
        erg = erg*a;  
    }  
    return erg;  
}
```

Nun werfen wir noch einen Blick auf den Aufruf der Funktion.

```
var a = 15;  
var fk = fakultaet(a);  
trace(fk);
```

Hier wird die Funktion als Wert einer Variablen aufgerufen. Dabei wird die Variable `a` als Parameter übergeben. Die Variable erhält dann von der Funktion als Rückgabewert die Variable `erg`, also die Fakultät. Diese können wir dann – außerhalb der Funktion – ausgeben.

Zu `return` möchten wir noch einiges ergänzen. Es übergibt nicht nur einen Wert, sondern es beendet auch die Funktion umgehend. Deswegen steht es normalerweise am Ende der Funktion. Wird mit `return` kein Wert übergeben, wird damit nur die Funktion beendet. Dies kann beispielsweise bei einer Fallunterscheidung sinnvoll sein, wenn eine Bedingung den Funktionsabbruch auslösen soll.

`return` ohne Parameter ähnelt funktional der `break`-Anweisung bei Schleifen.

Wenn Sie `return` keinen Wert zuweisen, die Funktion aber als Wert einer Variablen aufrufen, wird dieser Variablen der Wert `undefined` zugewiesen.



Parameter auslesen

Die Parameter einer Funktion werden von ActionScript in einem Array gesammelt. Dieses Array heißt `arguments`.

Obwohl wir Arrays erst noch behandeln werden, haben Sie aus dem JavaScript-Teil ja schon eine Vorstellung, wie man ein Array ausliest. In unserem Fall ist eine `for-in`-Schleife das Mittel der Wahl. Sie liest der Reihe nach alle Indizes des Arrays aus, bis keiner mehr vorhanden ist. In der Schleifen-Anweisung wird immer der Wert zum gerade ausgelesenen Index ausgegeben.

```
var a = 2;
var b = 3;
var c = 4;
name(a, b, c);
function name(a, b, c) {
    for (var par in arguments) {
        trace(arguments[par]);
    }
}
```

Ein Blick auf Abbildung 10.3 verrät, dass alle Parameter ausgelesen wurden.



Abbildung 10.3 Die Parameter werden ausgelesen.

10.1.3 Funktionen im Normalmodus

Wir haben Ihnen am Anfang versprochen, möglichst wenig im Normalmodus, sondern fast ausschließlich im Expertenmodus des **Aktionen**-Bedienfelds zu arbeiten. Auf eine Besonderheit in der Bedienung von Funktionen im Normalmodus müssen wir Sie allerdings noch hinweisen.

Wenn Sie eine eigene Funktion aufrufen möchten, verwenden Sie im Normalmodus `call function`. Sie finden diesen Befehl unter **Aktionen • Benutzerdefinierte Funktionen** im **Aktionen**-Bedienfeld.

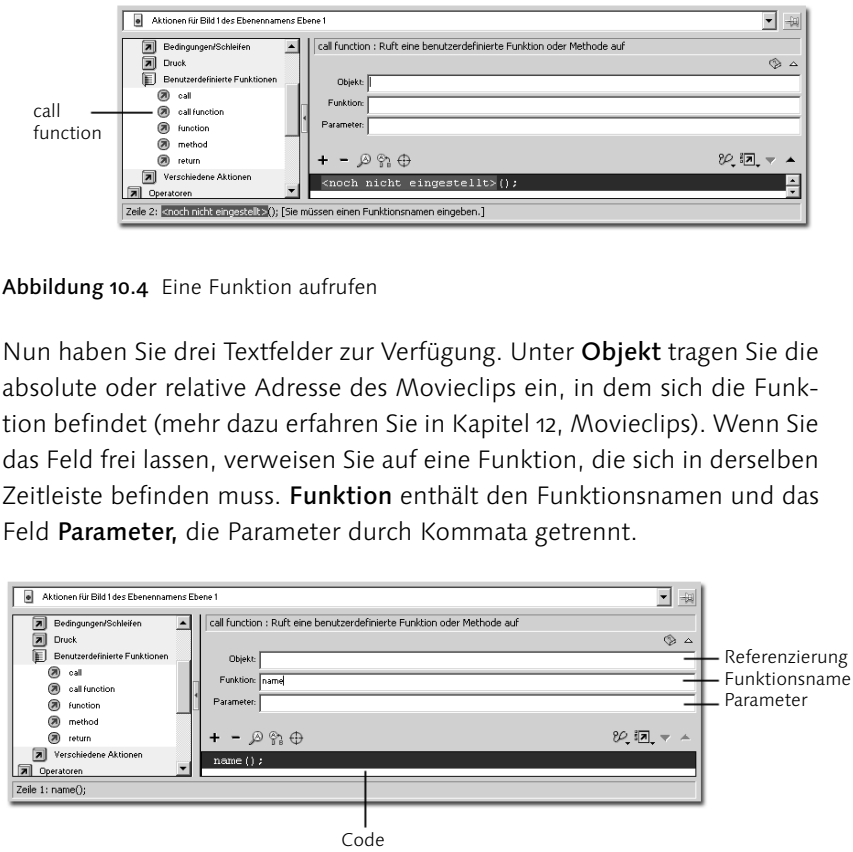


Abbildung 10.4 Eine Funktion aufrufen

Nun haben Sie drei Textfelder zur Verfügung. Unter **Objekt** tragen Sie die absolute oder relative Adresse des Movieclips ein, in dem sich die Funktion befindet (mehr dazu erfahren Sie in Kapitel 12, Movieclips). Wenn Sie das Feld frei lassen, verweisen Sie auf eine Funktion, die sich in derselben Zeitleiste befinden muss. **Funktion** enthält den Funktionsnamen und das Feld **Parameter**, die Parameter durch Kommata getrennt.

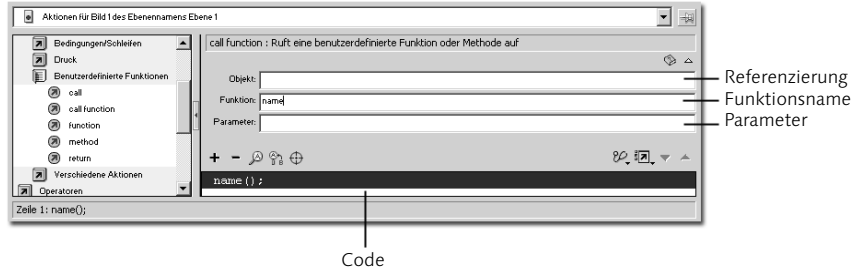


Abbildung 10.5 Die Daten eintragen

10.1.4 Code strukturieren

Der größte Nutzen, den Funktionen stiften, ist, den Code übersichtlicher zu gestalten. Funktionen verleihen dem Code Struktur und verhindern, dass man alles zehnmal abtippen muss.

Bevor wir Ihnen noch ein paar Möglichkeiten zum Programmieren und Strukturieren mit Funktionen zeigen, wollen wir Ihnen erst einmal an einem Beispiel illustrieren, was wir meinen. Erinnern Sie sich an den Taschenrechner aus dem vorangegangenen Kapitel? Dort musste bei jedem Operator ein Code eingegeben werden, der sich im Prinzip nur durch die Übergabe des richtigen Operators am Ende unterschied.


```

on (release) {
    if (feld != "" && !(feld.isNaN)) zahl1 = feld;
    feld = "";
    if (zahl1 != "") op = "+";
}

```

Diesen Code können Sie in eine Funktion packen, diese ins erste Schlüsselbild des Hauptfilms verfrachten und bei jedem Operator die Funktion aufrufen. Wenn Sie dann noch beim Funktionsaufruf die Art des Operators mitliefern, sparen Sie eine Menge Copy-and-Paste. Und so funktioniert es:

1. Erstellen Sie zuerst die Funktion im ersten Schlüsselbild des Films. Wir haben sie einfach unter den Variablendefinitionen angefügt. Die Funktion heißt `operator`, der Parameter ebenfalls. Am Schluss wird der globalen Variablen `op` einfach der Parameterwert statt des konkreten Operators übergeben.

```

var erg = "";
var zahl1 = "";
var zahl2 = "";
var op = "";

function operator(operator) {
    if (feld != "" && !(feld.isNaN)) zahl1 = feld;
    feld = "";
    if (zahl1 != "") op = operator;
}

```

2. Der Code bei den einzelnen Schaltflächen der Operatoren wird natürlich auch viel einfacher. Sie müssen lediglich die Funktion `operator` aufrufen und den Operator als String übergeben, hier beispielhaft für das `+` abgedruckt:

```

on (release) {
    operator("+");
}

```

3. Wenn Sie die Funktion für alle Operatoren aufgerufen haben, sind Sie auch schon fertig. Der Taschenrechner funktioniert wie gewohnt.



Sie finden das veränderte Taschenrechner-Beispiel unter dem Namen **taschenrechner_funktionen fla** auf der CD-ROM.

Rekursive Funktionen

Eine rekursive Funktion ist eine Funktion, die sich selbst aufruft. Mit rekursiven Funktionen lassen sich ähnliche Dinge anstellen wie in Schleifen. Tatsächlich dienen rekursive Funktionen in Programmiersprachen, die keine Schleifen haben, sogar als direkter Ersatz. Eine Sprache, die mit rekursiven Funktionen anstatt mit Schleifen arbeitet, ist beispielsweise Prolog.

Aber auch in Programmiersprachen mit Schleifen können rekursive Funktionen sehr nützlich sein. Die Anwendung zeigen wir gleich an einem Beispiel. Wir wollen die (positive) Potenz einer Zahl berechnen. Als Parameter übergeben wir drei Werte, zuerst die Zahl, dann die Potenz, anschließend noch einmal die Zahl. Diesen dritten Parameterwert benötigen wir für den Aufruf der rekursiven Funktion.

In der Funktion selbst prüfen wir per if-Anweisung, ob die Potenz größer als 0 ist. Wenn ja, wird die Ergebnisvariable neu gesetzt und mit `x` multipliziert. Anschließend wird die Potenz um eins verkleinert. Sie ist in diesem Fall also unsere Zählervariable. Dann rufen wir die Funktion neu auf. Als Parameter übergeben wir das unveränderte `x`, das um eins verkleinerte `a` und das neue Ergebnis `erg`. Jetzt beginnt das Ganze von vorne. Bei unseren Beispielszahlen wird die Funktion insgesamt zweimal neu aufgerufen.

```
function potenz(x, a, erg) {  
    if (a>0) {  
        erg *= x;  
        a--;  
        potenz(x, a, erg);  
    } else {  
        trace(erg);  
    }  
}  
potenz(10, 3, 10);
```

In Abbildung 10.6 sehen Sie das Ergebnis. Wer im Kopf mitgerechnet hat, wird feststellen: Es stimmt tatsächlich.



Abbildung 10.6 Die richtige Ausgabe



Rekursive Funktionen sind manchmal etwas umständlicher als Schleifen. Das obige Beispiel als Schleife ist schon ein wenig simpler. Unterschätzen Sie aber trotzdem rekursive Funktionen nicht. In einigen Fällen sind sie ausgesprochen interessante Lösungen.

```
function potenz(x, a) {
  var erg = x;
  while (a > 0) {
    erg = erg * x;
    a--;
  }
  trace(erg);
}
potenz(10, 3, 10);
```

Funktionen in Funktionen

Eine Funktion aus sich selbst heraus aufzurufen ist das eine (und Thema des vorhergehenden Abschnitts), Sie können aber auch Rückgabewerte von Funktionen als Parameter beim Aufruf einer neuen Funktion verwenden.

Im folgenden Beispiel mischen wir das Ergebnis einer Funktion zur Bildung der Potenz mit einer Funktion zur Bildung des Produkts.

```
function potenz(x, a) {
  var erg = x;
  while (a > 0) {
    erg = erg * x;
    a--;
  }
  return erg;
}
function produkt(a, b) {
```

```

    var prod = a * b;
    trace(prod);
}
var b = 2;
produkt(potenz(10,3), b);

```

Das funktioniert hervorragend, wie Abbildung 10.7 beweist.



Abbildung 10.7 Für dieses Ergebnis wurden zwei Funktionen bemüht.

Es geht aber noch weiter. Die Funktion `potenz` ließe sich auch direkt aus der Funktion `produkt` aufrufen:

```

function produkt(b) {
    var a = potenz(10,3);
    var prod = a * b;
    trace(prod);
}
var b = 2;
produkt(b);

```

Das Ergebnis bleibt das gleiche. Sie sehen, die Möglichkeiten, mit Funktionen strukturiert zu arbeiten, sind nahezu unbegrenzt. Bei einer solchen Fülle an Optionen bleiben natürlich auch Fehler nicht aus. Die häufigsten Fehlerquellen, auf die Sie achten sollten, sind:

- ▶ Funktionen, die nicht in derselben Zeitleiste liegen oder nicht global sind und falsch adressiert werden,
- ▶ fehlende oder unsauber definierte Parameter,
- ▶ falsch benannte Funktionen.

Wenn Sie Funktionen in Funktionen einsetzen, verwenden Sie einfach einmal eine `trace`-Anweisung, um den Rückgabewert der eingebundenen Funktion zu testen.



10.1.5 Vorgefertigte Funktionen

Eigene Funktionen zu definieren ist praktisch, strukturiert den Code und bietet eine Vielzahl von Möglichkeiten. Für viele Aufgaben müssen Sie allerdings noch nicht einmal mehr eigene Funktionen definieren, sondern können gleich auf die fertigen Funktionen von Flash zurückgreifen.

Vorgefertigte (auch: integrierte) Funktionen gibt es eine ganze Menge. Wir sortieren hier zunächst einmal die Begriffe aus und unterteilen die Funktionen danach, wo man sie findet und wie sie verfügbar sind.

- Im **Aktionen**-Bedienfeld unter **Funktionen** finden sich einige grundlegende Funktionen, die global verfügbar sind, also von überall her mit ihrem Namen aufgerufen werden können.

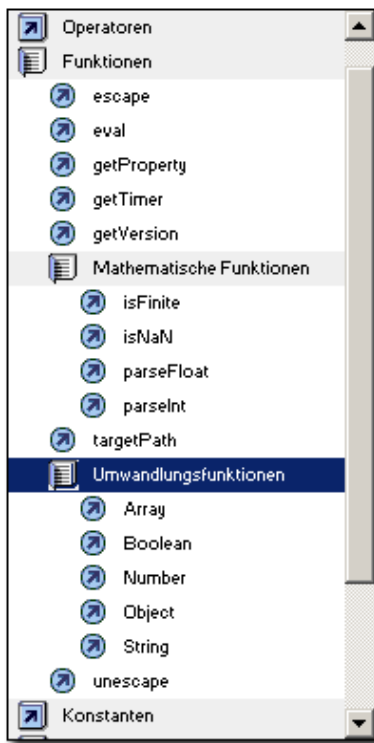


Abbildung 10.8 Überall verfügbare Funktionen

- Im **Aktionen**-Bedienfeld gibt es außerdem den Bereich **Aktionen**. Er enthält viele Schlüsselwörter wie `var`, `if` etc. Diese Schlüsselwörter bezeichnet man als Aktionen, Befehle oder Anweisungen. Viele davon

haben Sie schon im vorangegangenen Kapitel kennen gelernt und Sie werden im Verlauf dieses Buchs noch auf viele weitere stoßen. In den Aktionen gibt es aber auch viele »Aktionen«, die sich wie Funktionen verhalten und eigentlich auch global verfügbare Funktionen sind. Dazu gehört beispielsweise das schon vielfach verwendete `trace`.



Abbildung 10.9 In den Aktionen verbergen sich viele Funktionen.

- ▶ Außerdem gibt es einige veraltete Aktionen, die man in Flash MX nicht mehr verwenden sollte, die aber dennoch manchmal ganz nützlich sind. Sie finden sich im Bereich **Nicht mehr verwendet** im **Aktionen**-Bedienfeld.
- ▶ Zu guter Letzt ist jede Methode eines Objekts eigentlich eine Funktion. Ein Objekt ist zum Beispiel `Math`. Es enthält eine Menge Methoden für mathematische Rechnungen. Diese Methoden sind im Prinzip Funktionen, können aber nicht ohne ihr Objekt aufgerufen werden. Zu Objekten und Methoden finden Sie weitere Informationen in Kapitel 11, Objekte, Eigenschaften und Methoden.

Sie finden also an vier Stellen in ActionScript Funktionen, mit denen Sie arbeiten können. Die ersten drei stellen wir Ihnen jetzt vor, die letzten in Kapitel 11. Es ist dabei nicht wichtig, dass Sie alle beherrschen. Wir geben Ihnen hier nur eine Übersicht, die Sie aber in die Lage versetzt, die rich-

tige Funktion zu finden, und mit Ihren Kenntnissen über Parameter und das Aufrufen von Funktionen ist alles andere kein Problem mehr.

Funktionen aus Funktionen

Beginnen wir mit den globalen Funktionen, die auch in Flash wirklich als Funktionen bezeichnet werden. Sie sind in drei Sorten unterteilt: allgemeine, mathematische und Umwandlungsfunktionen.

Von den allgemeinen Funktionen sind Ihnen noch die meisten unbekannt. Deswegen finden Sie in der folgenden Tabelle eine kurze Übersicht.

Funktion	Beispiel	Beschreibung
<code>eval(String);</code>	<code>var a10 = 20; var b10 = eval("a" + "10");</code> b10 ergibt dann 20, da der Wert von a10 zugewiesen wird.	Dient der automatischen Generierung von Variablenamen. Wandelt eine Zeichenkette in einen Variablennamen um
<code>escape(String);</code>	<code>var x = escape("John Gärtner"); trace(x);</code> gibt John%20G%C3%A4rtner aus.	Codiert eine Zeichenkette in ein Format, das für die Übergabe in URLs geeignet ist. Wird beispielsweise bei der Übergabe von Formularwerten in der URL verwendet
<code>unescape(String);</code>	<code>var y = unescape(x);</code> verwandelt zurück in John Gärtner	Decodiert eine Zeichenkette, die mit escape codiert wurde
<code>getProperty(Film, Eigenschaft);</code>	<code>var prop = getPro- perty(this, _name);</code> liefert den Namen des aktuellen Movieclips bzw. der Instanz des Movieclips.	Diese Funktion liefert eine beliebige Eigenschaft eines Movieclips. Mehr dazu finden Sie in Kapitel 12, Movieclips.
<code>getTimer();</code>	<code>var t = getTimer();</code>	Gibt die Millisekunden zurück, die seit dem Start des Films verstrichen sind. Wird beispielsweise zum Festlegen der Startzeit einer Animation verwendet

Tabelle 10.1 Allgemeine Funktionen

Funktion	Beispiel	Beschreibung
<code>getVersion();</code>	<code>var string = getVersion();</code>	Liefert die Version des Flash Players in einer Zeichenkette. Funktioniert aber erst seit Flash 5.
<code>targetPath(Movieclip);</code>	<code>var pfd = targetPath(this);</code> <code>trace(pfd);</code>	Liefert den Zielpfad eines Movieclips. Mehr dazu finden Sie in Kapitel 12, Movieclips.

Tabelle 10.1 Allgemeine Funktionen (Forts.)

In zwei Beispielen der obigen Tabelle haben wir `this` verwendet. Damit wird der aktuelle Movieclip oder in anderen Kontexten das aktuelle Element, in dem sich der ActionScript-Code befindet, aufgerufen, ohne dass dessen Name bekannt sein muss.



Die vier mathematischen Funktionen kennen Sie alle schon aus dem vorangegangenen Kapitel. Deswegen stellen wir Sie Ihnen hier nur ganz kurz vor:

- ▶ `isFinite(Zahl)`; überprüft, ob die angegebene Zahl endlich ist.
- ▶ `isNaN(Ausdruck)`; prüft, ob es sich um einen Ausdruck handelt, der keine gültige Zahl ist (Not a Number).
- ▶ `parseFloat(String)`; extrahiert eine Gleitkommazahl vom Anfang eines Strings. Bei einer Zeichenkette »17.99 cm« liefert sie also die Zahl 17.99.
- ▶ `parseInt(String)`; extrahiert eine ganze Zahl aus dem Anfang einer Zeichenkette. »18 Meter« liefert also die Zahl 18.

Die Umwandlungsfunktionen haben Sie ebenfalls bei den Datentypen im vorhergehenden Kapitel kennen gelernt. Deswegen hier nur noch einmal eine kurze und knackige Zusammenfassung. Die Regeln für wichtige Datentypumwandlungen entnehmen Sie bitte Kapitel 8, ActionScript: Grundlagen.

- ▶ `Array` – wandelt in ein Array um.
- ▶ `Boolean` – wandelt in einen Wahrheitswert um.
- ▶ `Number` – wandelt in eine Zahl um.
- ▶ `Object` – wandelt in ein Objekt um.
- ▶ `String` – wandelt in eine Zeichenkette um.

Funktionen aus Aktionen

Die Funktionen im Bereich **Aktionen** des **Aktionen**-Bedienfelds sind in einigen Unterkategorien verteilt. Wir teilen die Übersicht in diesem Kapitel ebenfalls nach diesen Unterkategorien ein.

Filmsteuerung

Die Funktionen im Bereich **Filmsteuerung** tun, was der Name schon sagt: Sie steuern den Film und damit den Abspielkopf in der Zeitleiste. Die folgende Tabelle bietet eine Zusammenfassung.

Funktion	Beispiel	Beschreibung
<code>gotoAndPlay(Bild);</code> <code>gotoAndPlay(Szene, Bild);</code>	<code>gotoAndPlay(10);</code> springt zu Bild (=Frame) 10. <code>gotoAndPlay(szene 1;1);</code> springt zu Bild 1 der Szene scene 1.	Springt zu einem Bild oder zu einem Bild in einer Szene. Dort wird der Film weiter abgespielt.
<code>gotoAndStop(Bild);</code> <code>gotoAndStop(Szene, Bild);</code>	<code>gotoAndStop(10);</code> springt zu Bild (=Frame) 10 und hält den Abspielkopf an. <code>gotoAndStop(szene 1;1);</code> springt zu Bild 1 der Szene scene 1 und hält den Abspielkopf an.	Springt zu einem Bild oder zu einem Bild in einer Szene. Dort wird der Film angehalten.
<code>nextFrame();</code>		Springt zum nächsten Bild und hält an
<code>nextScene();</code>		Springt zur nächsten Szene und hält an
<code>prevFrame();</code>		Springt zum vorhergehenden Bild und stoppt
<code>prevScene();</code>		Springt zur vorhergehenden Szene und hält an
<code>play();</code>		Startet den Abspielkopf. Wird oft mit den vier darüber liegenden Funktionen gekoppelt, da diese den Abspielkopf anhalten

Tabelle 10.2 Funktionen zur Filmsteuerung

<code>stop();</code>		Hält den Abspielkopf sofort an. Wird beispielsweise zu Beginn eines Films verwendet, wenn ActionScript-Logik oder Anwendereingaben folgen sollen
<code>stopAllSounds();</code>		Hält die Soundwiedergabe an. Der Abspielkopf läuft weiter. Vorsicht: Wenn er Bilder mit Sound trifft, wird dieser abgespielt.

Tabelle 10.2 Funktionen zur Filmsteuerung (Forts.)

Wir wollen Ihnen die Filmsteuerung kurz an einem ganz einfachen Beispiel zeigen. Wir haben eine Datei mit einem einfachen Bewegungstween und wollen dazu drei Schaltflächen wie an einem CD-Player erstellen, die den Film pausieren, anhalten und auf das erste Bild setzen und wieder weiterspielen lassen. In Abbildung 10.10 sehen Sie, wie die fertige Datei aussehen soll.

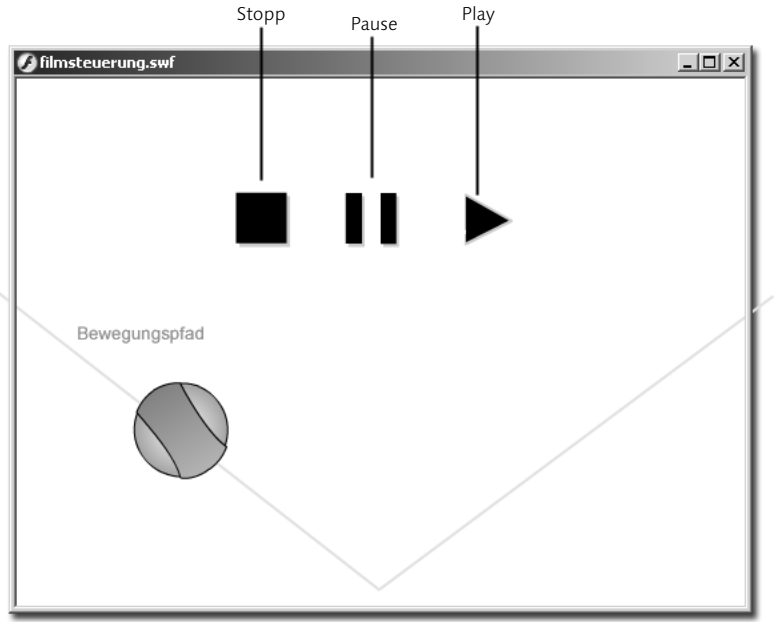


Abbildung 10.10 So soll es aussehen (der Bewegungspfad wurde nur zur Orientierung eingefügt).

Als Bewegungsobjekt haben wir ein Tennissymbol gewählt. Wir schildern Ihnen kurz, welche Vorbereitungen in Flash nötig sind. Sollte Ihnen das zu kompliziert sein, übergehen Sie die nächsten Schritte und legen gleich mit der fertigen Flash-Datei bei Schritt 6 los.

1. In der Datei **tennisball fla** finden Sie den Tennisball mit Bewegungstween von links nach unten und dann nach rechts.



Für die Interessierten: Der Tennisball wurde in Freehand erstellt und besteht aus zwei Einzelteilen (siehe Abbildung 10.11), der eine mit radialem Verlauf von Weiß nach Gelb, der andere mit linearem Verlauf von Hellgelb nach Dunkelgelb. Beide Einzelteile haben einen schwarzen Rahmen, damit man die charakteristische Form eines Tennisballs erkennen kann.

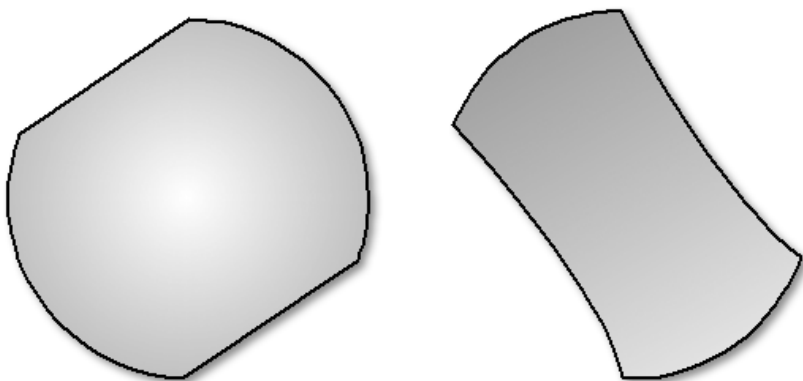


Abbildung 10.11 Die zwei Einzelteile des Tennisballs

2. Nachdem Tennisball und Bewegungstween schon fertig sind, benötigen wir nur noch die drei Schaltflächen. Wir haben alle drei auf eine neue Ebene gepackt.
3. Alle drei bestehen aus einfachen Grundformen, die dupliziert und grau eingefärbt als Schatten noch dahinter gelegt werden.
4. Anschließend werden sie einzeln in Schaltflächensymbole umgewandelt.
5. Für den gedrückten Zustand erhalten alle drei ein neues Schlüsselbild. Einen Drückeﬀekt erreichen wir, indem wir die Schaltfläche stärker über den grauen Schatten schieben.



Sollte Ihnen die Arbeit in Flash zu langweilig sein, weil Sie Ihnen schon so locker von der Hand geht (nicht geschwindelt?), finden Sie in der Datei **tennisball_mit_schalt fla** die fertige Version des Flash-Films, allerdings noch ohne ActionScript.

6. So, die Vorbereitungen sind erledigt. Jetzt kommen wir zur ActionScript-Programmierung. Wir wollen zunächst erreichen, dass die Animation endlos abläuft. Dazu wechseln wir in das letzte Schlüsselbild des Films und geben dort folgenden Code ein:

```
gotoAndPlay(1);
```

Damit springt der Abspielkopf am Ende der Flash-Datei wieder auf den Anfang. Denn Sie dürfen nicht vergessen, in der Testwiedergabe in der Flash-Entwicklungsumgebung läuft der Film – wenn gewünscht – in einer Endlosschleife ab, dies wirkt sich aber nicht auf die veröffentlichte swf-Datei aus. Für diese müssen wir die Schleife mit ActionScript einrichten.

7. Nun zu den Schaltflächen. Beginnen wir mit der Stopp-Schaltfläche. Sie soll auf das erste Bild springen und dort den Abspielkopf anhalten. Das erreichen wir mit der Funktion `gotoAndStop()`.

```
on (release) {  
    gotoAndStop(1);  
}
```

8. Die Pause-Schaltfläche funktioniert ähnlich wie die Stopp-Schaltfläche, nur sollte sie nicht auf das erste Bild zurückspringen. Die richtige Funktion ist also `stop()`.

```
on (release) {  
    stop();  
}
```

9. Die Play-Schaltfläche ist sehr einfach mit der Funktion `play()` zu realisieren.

```
on (release) {  
    play();  
}
```

Das vollständige Beispiel finden Sie in der Datei **filmsteuerung.fla** auf der CD-ROM.

Freuen Sie sich, schwieriger als dieses Beispiel ist die Filmsteuerung mit ActionScript nicht. Gut, das ist vielleicht ein bisschen übertrieben, aber mit sehr wenigen Befehlen können Sie schon eine Menge erreichen.

Browser und Netzwerk

Die nächste Kategorie im **Aktionen**-Bedienfeld ist **Browser/Netzwerk**. Hier wird die Kommunikation des Flash Players mit der Umgebung, also beispielsweise dem Browser, gesteuert.

Die folgende Tabelle bietet einen Überblick über die Funktionen.

Funktion	Beispiel	Beschreibung
<code>getURL(URL);</code> <code>getURL(URL, Fenster</code> od. <code>Frame)</code> <code>getURL(URL, Fenster</code> od. <code>Frame, Versandme-</code> thode)	<code>getURL(http://www.hauser-</code> wenz.de) <code>getURL(http://www.hauser-</code> wenz.de, "_top") <code>getURL(http://www.hauser-</code> wenz.de/script.php, "_ blank", POST)	Ruft eine Webadresse im Browser auf. Optional kann das Fenster bzw. der Frame angegeben werden. Neben dem Fenster- oder Framenamen sind Standardwerte: <code>_top</code> (im aktuellen Fenster auf oberster Ebene); <code>_parent</code> (im übergeordneten Frame); <code>_blank</code> (im neuen Fenster); <code>_self</code> (im selben Frame, Standardeinstellung). Die optionale Versandmethode bestimmt die Art des Formularversands: POST oder GET. Mehr dazu in Kapitel 6, Formulare.
<code>fscommand(Befehl,</code> String);		Übermittelt einen Befehl, der als Parameter einen String enthält, an den Flash Player oder die übergeordnete Anwendung, also meist den Webbrowser. So kann Flash beispielsweise mit JavaScript oder Lingo (Macromedia Director) kooperieren. Mehr dazu finden Sie in Kapitel 17, ActionScript mit JavaScript.

Tabelle 10.3 Funktionen für die Browsersteuerung und das Netzwerk

Funktion	Beispiel	Beschreibung
<pre>loadMovie(URL, Ziel); loadMovie(URL, Ziel; Methode);</pre>	<pre>loadMovie("beispiel.swf", _root);</pre> lädt eine swf-Datei in die Hauptzeitleiste.	Lädt einen swf-Film oder eine JPEG-Datei in den Flash-Film. So lassen sich mehrere Flash-Filme verbinden. Die URL gibt die Position der zu ladenden Datei an, das Ziel gibt an, in welchen Movieclip oder welche Dokumentebene der Film oder das Bild geladen werden soll. Optional ist die Angabe einer Versandmethode GET oder POST für Variablen. Mit GET können beispielsweise die Variablen des Ausgangsfilms auf den geladenen übertragen werden.
<pre>loadMovieNum(URL, Dokumentebene); loadMovieNum(URL, Dokumentebene, Methode);</pre>	<pre>loadMovieNum("beispiel.swf", 1);</pre> lädt einen Film in die Dokumentebene 1.	Lädt einen Film oder ein JPEG in eine Dokumentebene, die mit einem Index angegeben wird. Die Versandmethode kann ebenfalls bestimmt werden. Eine Dokumentebene entspricht nicht einer Ebene in der Zeitleiste. Vielmehr liegt in Flash jede swf-Datei in einer eigenen Dokumentebene. Der Hauptfilm hat immer den Index 0, andere Filme können beliebige Indizes annehmen.
<pre>unloadMovie(Ziel);</pre>	<pre>unloadMovie("_root.test");</pre> entfernt den Film aus dem Movieclip test.	Entfernt einen Film oder ein JPEG-Bild mit Angabe des Ziels
<pre>unloadMovieNum(Dokumentebene);</pre>	<pre>unloadMovieNum(1);</pre> entfernt den Film aus der Dokumentebene 1.	Entfernt den Film oder das Bild aus der Dokumentebene

Tabelle 10.3 Funktionen für die Browsersteuerung und das Netzwerk (Forts.)

Funktion	Beispiel	Beschreibung
<code>loadVariables(URL, Ziel);</code> <code>loadVariables(URL, Ziel, Methode);</code>	<code>loadVariables("skript.php", "_root");</code> lädt Variablen aus einem PHP-Skript. Die Variablen müssen URL-encodiert sein, also dem Format genügen, dass zum URL-Versand benötigt wird. Die ActionScript-Funktionen zum Codieren und Decodieren sind <code>escape</code> und <code>unescape</code> . Solche Methoden hält jede wichtige Skriptsprache bereit.	Lädt Variablen aus einer externen Datei. URL der Datei und Ziel (Movieclip), wohin die Variablen geladen werden sollen. Optional lässt sich die Versandmethode POST oder GET festlegen. Damit ist auch die Skriptanbindung möglich.
<code>loadVariablesNum(URL, Dokumentebene);</code> <code>loadVariablesNum(URL, Dokumentebene, Methode);</code>	<code>loadVariablesNum("skript.aspx", 1);</code> lädt Variablen aus einem ASP.NET-Skript auf Dokumentebene 1.	Lädt Variablen in eine bestimmte Dokumentebene, die mit ihrem Index bestimmt wird. Versandmethode ist optional.

Tabelle 10.3 Funktionen für die Browsersteuerung und das Netzwerk (Forts.)

`fscommand` hält – hauptsächlich für den eigenständigen Flash Player – ein paar Befehle bereit, die dessen Aussehen und dessen Interaktion mit der Umgebung steuern. Im Einzelnen sind das:

- `quit` – beendet den Standalone-Player, benötigt keinen Parameter.

```
fscommand("quit");
```

- `allowscale` – wenn `true`, werden die Elemente des Films mit dem Player-Fenster vergrößert und verkleinert. Wenn `false`, werden die Elemente in ihrer Originalgröße belassen.

```
fscommand("allowscale", "false");
```

- `fullscreen` – zeigt bei `true` den Flash Player über den gesamten Bildschirm im Vollbild an.

```
fscommand("fullscreen", "true");
```



Neulich surften wir bei einer bekannten deutschen Sportinformationsseite vorbei: Wir sahen im Vollbild Autoverbraucherinformationen, die sich nicht selbst beenden und keine Ausschalter haben. `[Alt] + [F4]` hat gewonnen, die Marke in den Augen der Verfasser verloren. Sie sollten die

Funktion also nur einsetzen, wenn Vollbild dem Anwender Vorteile bringt.

- `showmenu` – wichtige und häufig verwendete Option. Wenn der Parameter `false` ist, wird kein Kontextmenü angezeigt. `true` ist die Standardeinstellung und zeigt das normale Kontextmenü.

```
fscommand("showmenu", "false");
```

Wir mögen es als Anwender nicht besonders, wenn uns ein Flash-Film bevormundet und einfach das Kontextmenü, mit dem wir stoppen und spulen könnten, entfernt. Überlegen Sie, wie Ihre Anwender darüber denken.



- `exec` – führt eine beliebige Anwendung auf dem Rechner des Anwenders aus. Als Parameter wird der absolute Pfad angegeben.

```
fscommand("exec", "Programmpfad");
```

- `trapallkeys` – alle Tastenkürzel (im eigenständigen Flash Player) sind Sache des Flash-Films.

Movieclip-Steuerung

Wieder ein sprechender Name für eine Oberkategorie. Diese enthält lauter Funktionen, mit denen Sie mit Movieclips arbeiten können. Die folgende Tabelle bietet einen Überblick.

Funktion	Beispiel	Beschreibung
<code>duplicateMovieClip(Quell-Movieclip, Name, Tiefenebene);</code>	<code>duplicateMovieClip("_root.test", "test2", 2);</code> dupliziert den Clip <code>test</code> , nennt das Duplikat <code>test2</code> und legt es auf Tiefenebene 2.	Dupliziert einen Movieclip. Parameter sind der Ausgangs-Movieclip (die Quelle), der neue Name für den Movieclip und die Tiefenebene, auf der sich der Movieclip befindet. Vorsicht, diese ist unabhängig von den Ebenen in der Zeitleiste.
<code>removeMovieClip(Ziel);</code>	<code>removeMovieClip("_root.test");</code> entfernt den Movieclip <code>test</code> .	Entfernt einen Movieclip

Tabelle 10.4 Funktionen zur Movieclip-Steuerung

Funktion	Beispiel	Beschreibung
<code>setProperty(Ziel, Eigenschaft, Wert);</code>	<pre>setProperty("_root.test", _y, "40");</pre> setzt die Position auf der y-Achse für den Movieclip test auf 40 Pixel.	Setzt die Eigenschaft eines Movieclips auf einen bestimmten Wert
<pre>startDrag(Ziel); startDrag(Ziel, Zentrum); startDrag(Ziel, Zentrum, x1, y1, x2, y2);</pre>	<pre>startDrag("_root.tb", true);</pre> zieht einen Movieclip namens tb und erlaubt Einrasten am Mittelpunkt der Mausposition. (Ein ausführlicheres Beispiel finden Sie im Anschluss an die Tabelle.)	Erlaubt das Ziehen eines Movieclips mit der Maus. Als Pflichtparameter wird der Movieclip angegeben. Optional bestimmen Sie, ob beim Loslassen am Mittelpunkt des Movieclips (<code>true</code>) oder an der Mausposition eingerastet wird (<code>false</code>). Die zweite optionale Einstellung sind vier Koordinaten eines Rechtecks, die den aktiven Bereich des Movieclips angeben, der gezogen werden kann.
<code>stopDrag();</code>	<pre>stopDrag();</pre> hält das Drag-and-Drop wieder an.	Steuert das Aufhören der Ziehbewegung
<code>updateAfterEvent();</code>		Aktualisiert die Bildschirmdarstellung. Muss in einem benutzerdefinierten Ereignis wie <code>onClickEvent</code> oder einer Methode von <code>setInterval</code> aufgerufen werden.

Tabelle 10.4 Funktionen zur Movieclip-Steuerung (Forts.)

Wie versprochen, reichen wir hier noch ein kurzes Beispiel zu Drag-and-Drop nach, da sich diese Möglichkeiten normalerweise nicht so einfach vorstellen lassen.

Für unser Beispiel verwenden wir den bereits bekannten Tennisball. Er soll sich jetzt auf der Bühne um die eigene Achse drehen und per Drag-and-Drop verschiebbar sein.

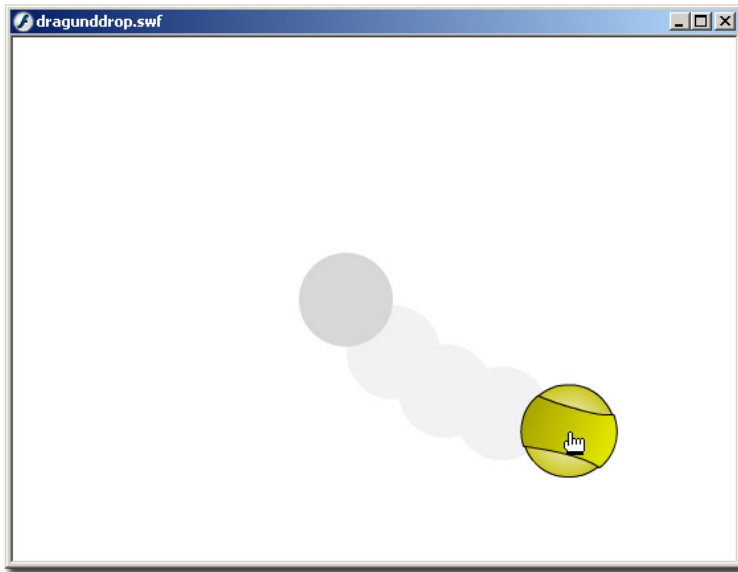


Abbildung 10.12 Drag-and-Drop (Bewegungspfad zur Illustration)

Um das zu realisieren, verwenden wir die Funktionen `startDrag` und `stopDrag`. Sie funktionieren nur mit einem Movieclip, was uns auch schon direkt zur Umsetzung führt. Sie finden die Datei bereits fertig vorbereitet unter `tennisball_dud fla`. Wir haben folgende Schritte durchgeführt, um diesen Stand zu erreichen:

1. Zuerst haben wir den Tennisball in die Datei kopiert.
2. Dann haben wir ihn in ein Movieclip-Symbol umgewandelt.
3. Das Movieclip-Symbol hat ein zweites Schlüsselbild in Bild 10 bekommen. Außerdem wurde es mit einem Bewegungstween versehen, der die Anweisung hat, sich im Uhrzeigersinn zu drehen (siehe Abbildung 10.13).
4. In der Hauptzeitleiste haben wir nun noch auf Bild 20 ein Schlüsselbild eingefügt.
5. Die Instanz des Movieclips wurde auf die Bühne gezogen (achten Sie darauf, alle Bilder zu markieren) und in `tb` umbenannt. Vorsicht: Wenn der Original-Tennisball noch nicht gelöscht ist, hat er noch nicht den Typ Movieclip.



Drehen im Uhrzeigersinn (UZS)

Abbildung 10.13 Drehen im Bewegungstween

Wenn Sie jetzt einmal zwischendurch den Film probeweise abspielen, dreht sich der Tennisball auf der Bühne. Nun aber zum spannenden Teil, der Arbeit mit ActionScript:

1. Als Erstes müssen wir die Animation in einer Endlosschleife abspielen lassen. Wechseln Sie dazu auf das zweite Schlüsselbild in der Hauptzeitleiste (Bild 20) und geben Sie dort ein, dass der Abspielkopf in Bild 1 zurückspringen soll.

```
gotoAndPlay(1);
```

2. Jetzt geht es an den Movieclip. Markieren Sie beide Schlüsselbilder in der Zeitleiste und dann den Movieclip. Wichtig ist nämlich, dass das Drag-and-Drop für alle Bilder des Films gilt.

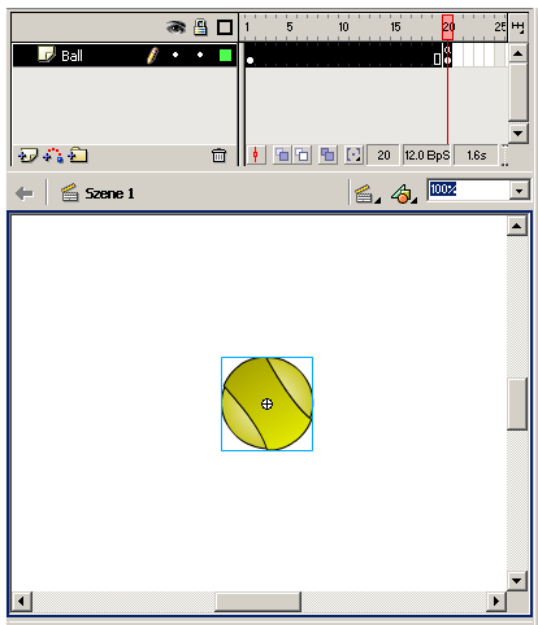


Abbildung 10.14 Die Movieclip-Instanz ist für alle Bilder in der Zeitleiste markiert.

3. Nun können Sie im **Aktionen**-Bedienfeld das Drag-and-Drop-Skript eintragen. Beginnen soll das Drag-and-Drop beim Ereignis `press`, also wenn der Anwender die linke Maustaste drückt. Als Ziel geben wir den Namen der Movieclip-Instanz an, die sich hierarchisch direkt unter der Hauptzeitleiste (`_root`) befindet. Der zweite Parameter gibt an, ob der Movieclip an der angeklickten Stelle (`true`) oder an der Mitte des Movieclips einrastet (`false`).

```
on (press) {  
    startDrag("_root.tb",true);  
}
```

4. So, das Drag ist festgelegt und funktioniert auch schon. Kommen wir also zum Drop. Hier verwenden wir zwei Ereignisse: `release` und `releaseOutside`. Wenn Sie sich jetzt wundern, weil `releaseOutside` doch eigentlich abdeckt, dass man außerhalb des aktiven Movieclips die Maustaste loslässt, haben Sie Recht. Dieses Ereignis brauchen wir, weil bei sehr schnellen Mausbewegungen während des Loslassens der Maustaste ein Movieclip oft noch hängen bleibt, weil das `release`-Ereignis verpasst wurde.

```
on (release, releaseOutside) {  
    stopDrag();  
}
```

Fertig ist das Drag-and-Drop-Erlebnis. Die vollständige Datei finden Sie unter dem Namen **dragunddrop.fla** auf der CD-ROM.

set variable

Im Bereich **Aktionen • Variablen** des **Aktionen**-Bedienfelds befindet sich nur eine Funktion: `set()`. Sie dient dazu, einer Variablen einen Wert zuzuweisen. Sie können sie vor allem dazu verwenden, Variablen dynamisch zu erzeugen.

Eine einfache Variable ist schnell mit einem Wert versehen:

```
var a;  
set("a", 3);  
trace(a);
```

Achten Sie auf die Anführungszeichen um den Variablennamen. Der Wert (zweiter Parameter) erhält nur dann Anführungszeichen, wenn er ein String ist.

Nun wird es etwas interessanter. Wir erstellen aus einem Variablenwert und einem Zähler in einer Schleife neue Variablen und weisen ihnen jeweils den Zählerwert zu.

```
var a = "b";

for (var i = 1; i<5; i++) {
    set(a + i, i);
}
```

In unserem Beispiel wird von 1 bis 4 gezählt. In jedem Durchlauf wird eine Variable gesetzt. Sie hat einen Namen, zusammengesetzt aus `b` (dem Wert der Variablen `a`) und dem Zähler, also 1, 2, 3, 4. Der Wert der Variablen ist der Zähler, also 1, 2, 3, oder 4.

Um das sehen zu können, geben wir die Werte einfach mit einer zweiten Schleife aus:

```
for (var i = 1; i<5; i++) {
    trace(eval("b" + i));
}
```

In Abbildung 10.15 sehen Sie das Ergebnis, nämlich die Zahlen von 1 bis 4.

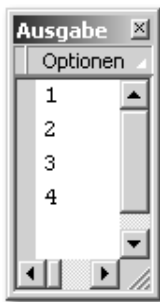


Abbildung 10.15 Die Zahlen von 1 bis 4 werden ausgegeben.



`set` ist eigentlich noch ein Relikt aus Flash 4. Es hieß dort `set variable` und wird auch heute noch im Aktionen-Bedienfeld so bezeichnet. Der Sinn von `set variable` war in Flash 4, mehrere Variablen beispielsweise in einer Schleife dynamisch erstellen zu können. Damit konnte man die Funktionalität eines Arrays simulieren, das es erst seit Flash 5 gibt. `set` wird zwar von Flash nicht als veraltet angesehen, ist aber deswegen immer seltener im Gebrauch.

Drucken

Drucken aus Flash ist natürlich auch möglich. Genauer gesagt gibt es in **Aktionen • Druck** im **Aktionen**-Bedienfeld vier Funktionen dafür, die wir in der folgenden Tabelle zusammengestellt haben.

Funktion	Beispiel	Beschreibung
<code>print(Ziel, Begrenzung);</code>	<code>print(this, "bframe");</code> druckt den aktuellen Movieclip und nimmt für jeden Rahmen die benötigte Größe und skaliert diese auf Blattgröße.	Druckt alle Bilder eines Movieclips oder einer Dokumentebene (angegeben unter Ziel). Begrenzung kann drei Optionen annehmen: <code>bframe</code> druckt jedes Bild in seiner Größe (wird durch die vorhandenen Elemente bestimmt). <code>bmax</code> kombiniert die Abmessung aller Bilder und bildet daraus den größtmöglichen Druckbereich. <code>bmovie</code> orientiert sich an einem Bild mit dem Bezeichner <code>#b</code> und übernimmt dessen Größe.
<code>printNum(Dokumentebene, Begrenzung);</code>	<code>printNum(0, "bmax");</code>	Druckt alle Bilder einer Dokumentebene (Vorsicht: Nicht mit Ebenen in der Zeitleiste verwechseln!)
<code>printAsBitmap(Ziel);</code> <code>printAsBitmap(Ziel, Begrenzung);</code>	<code>printAsBitmap(this, "bframe");</code>	Druckt alle Bilder eines Movieclips oder einer Dokumentebene (angegeben unter Ziel) als Bild. Sollte man verwenden, um Filme mit Bitmap-Bildern und Transparenzen zu drucken
<code>printAsBitmapNum(Dokumentebene);</code> <code>printAsBitmapNum(Dokumentebene, Begrenzung);</code>	<code>printAsBitmapNum(2, "bmovie");</code>	Druckt eine Dokumentebene als Bild. Sollte man verwenden, um Filme mit Bitmap-Bildern und Transparenzen zu drucken.

Tabelle 10.5 Drucken mit ActionScript

Die Druckfunktionen drucken alle Frames eines Movieclips oder einer Dokumentenebene. Wollen Sie, dass nur bestimmte Frames ausgedruckt werden, müssen Sie diese mit `#p` beschriften.

Zum Abschluss wollen wir Ihnen noch zeigen, wie das funktioniert, und beispielhaft eine Print-Schaltfläche in einen Film einbauen. Wenn man sie anklickt, soll ein festgelegtes Bild des Films ausgedruckt werden. Als Ausgangspunkt haben wir uns wieder **tennisball fla** ausgesucht.

In Abbildung 10.16 sehen Sie schon, wie das Beispiel funktionieren soll.

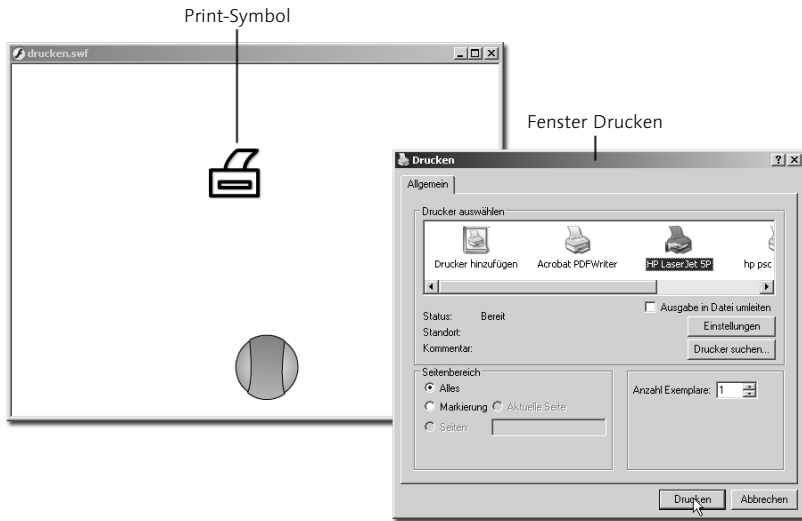


Abbildung 10.16 So soll unser Druck-Beispiel funktionieren.

Die Vorbereitungen sind nicht allzu aufwändig:

1. Wir haben auf einer neuen Ebene ein Print-Zeichen mit Schatten erstellt. Es besteht im Wesentlichen aus zwei Rechtecken und einem von Hand gezeichneten Pfad, der das Papier symbolisieren soll. Sie können aber natürlich auch eine beliebige andere Schaltfläche verwenden, da hier nicht die Optik, sondern die Funktionalität entscheidend ist.



Abbildung 10.17 Das Print-Zeichen – schnell erstellt und sieht nett aus

2. Anschließend haben wir das Print-Zeichen in ein Schaltflächen-Symbol umgewandelt.
3. Der Zustand **Drücken** des Schaltflächensymbols erhält ein neues Schlüsselbild, bei dem die Kontur über den Schatten verschoben wurde, so dass ein optischer Drückefferkt entsteht.



Abbildung 10.18 Der gedrückte Zustand

4. Zum Schluss musste noch der Zustand Aktiv geändert werden, da die Schaltfläche nicht gefüllt ist und der Anwender im Inneren sonst ins Leere drücken würde.



Abbildung 10.19 Im Zustand **Aktiv** ist die Schaltfläche gefüllt.

Wir setzen an dieser Stelle einiges an Wissen über Schaltflächen und ihre Zustände voraus. Sollten Sie hier Schwierigkeiten haben, ist das kein Beinbruch, ein gutes Flash-Buch hilft da weiter, z.B. das aktuelle von Sascha Wolter, ebenfalls erschienen bei Galileo Press.



Die Vorbereitungen in Flash sind erledigt, jetzt geht es an die Skript-Programmierung.

1. Zuerst machen wir den Film wieder zu einer Endlosschleife. Dazu springen wir aus dem letzten Schlüsselbild in das erste Bild des Films.

```
gotoAndPlay(1);
```

2. Dann kommt eine ganz wichtige Einstellung. Wir markieren das Bild, das wir drucken wollen, mit #p. Dazu wechseln wir in das Bild, in dem der Ball ganz unten am Rand der Bühne positioniert ist, das Schlüssel-

bild. Im Eigenschafteninspektor gibt es das Feld **Bildmarkierung**. Hier tragen Sie `#p` ein.



Abbildung 10.20 Die Bildmarkierung eintragen



Vorsicht: Wenn Sie das vergessen, werden alle Bilder gedruckt! Viel Spaß beim Papier Nachlegen oder beim »Verzweifelt-den-Druckauftrag-Stoppen«. Sagen Sie dann aber nicht, wir hätten Sie nicht gewarnt.



Manchmal kann es auch sinnvoll sein, eine komplette Animation auf einzelne Blätter zu drucken. Beispielsweise, wenn man sich über Abläufe klar werden will.

3. Zu guter Letzt kümmern wir uns noch um das Print-Symbol. Hier benötigen wir beim Ereignis `release`, also beim Loslassen der Maustaste, den Druckbefehl. Mit `this` geben wir an, dass der aktuelle Movieclip gedruckt werden kann, mit `bmovie` drucken wird die Bühnengröße auf die Papiergröße skaliert, da wir auch nicht zusätzlich ein Bild mit `#b` als Begrenzungsrahmen angegeben haben.

```
on (release) {  
    print(this, "bmovie");  
}
```



Sie können ein Bild einer Ebene natürlich nicht gleichzeitig mit den Markierungen `#p` und `#b` versehen. Beide Bezeichnungen dürfen außerdem nur einmal pro Zeitleiste vergeben werden.

Die fertige Datei finden Sie auf der CD-ROM unter dem Namen **drucken fla**.



Die verschiedenen Druck-Funktionen gibt es erst seit Flash 5, bzw. seit dem Flash Player 4 r20 (Release 20). Früher hat man zum Drucken die Druckfunktion des Browsers verwendet und diese mit der Funktion `getURL` übergeben. Dies ist heute nur noch sinnvoll, wenn man für Flash

4 produzieren möchte. Ansonsten drucken die Druck-Funktionen von Flash die Vektorgrafiken im Film sauberer, als das der Browser könnte.

Vermischtes

In **Aktionen • Verschiedene Aktionen** im **Aktionen**-Bedienfeld finden sich noch ein paar Funktionen, die sonst nirgends untergebracht werden konnten:

- `trace()` muss eigentlich nicht mehr vorgestellt werden. Es gibt eine Zeichenkette, eine Variable oder einen anderen Ausdruck im Ausgabe-fenster aus. Da `trace` hauptsächlich zum Testen beim Programmieren verwendet wird, müssten Sie eigentlich beim Veröffentlichen des Flash-Films alle `trace`-Anweisungen entfernen. Diese Mühe wird Ihnen von der Option **Trace-Aktionen übergehen** in **Datei • Einstellungen für Veröffentlichungen**, Register **Flash** abgenommen.

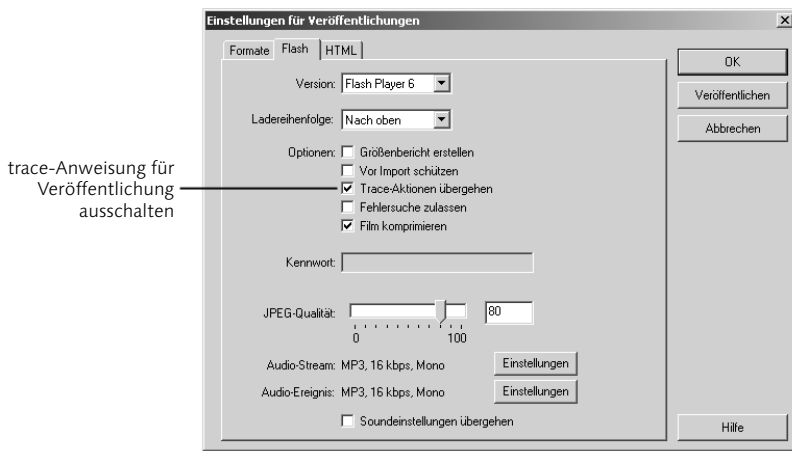


Abbildung 10.21 Trace-Anweisungen für das Veröffentlichen ausschalten

- `setInterval(Funktion, Zeitabstand, Parameter1, Parameter2, ...)` ruft eine Funktion oder Methode immer wieder in festgelegten Zeitabständen auf. Zuerst wird die Funktion angegeben, dann der Zeitabstand in Millisekunden, anschließend optional Parameter für die Funktion. Folgendes Beispiel ruft alle zwei Sekunden (= 2000 Millisekunden) die Funktion `ausgabe` auf. Funktionsnamen werden ohne runde Klammern angegeben. In unserem Beispiel übergibt die Funktion den Parameter 2.

```
function ausgabe(s) {
    trace(s + " Sekunden sind vergangen");
}

var intl = setInterval(ausgabe, 2000, 2);
```

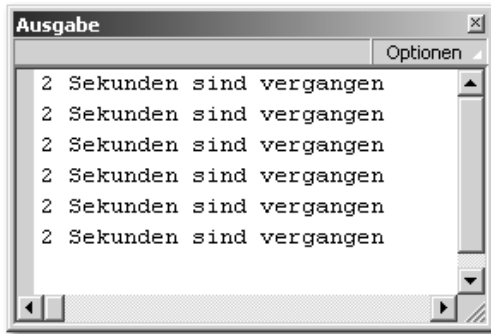


Abbildung 10.22 Der Autor hat zwölf Sekunden seines Lebens mit dem Bewundern einer Ausgabe zugebracht.



Wenn Sie `setInterval` für die Methode eines Objekts anwenden möchten, müssen Sie zuerst das Objekt und dann durch Komma getrennt die Methode angeben. Der Methodenname steht dabei in Anführungszeichen.

```
setInterval(Objekt, "Methode", Zeitabstand, Parameter1,
Parameter2, ...)
```

- `clearInterval(Name)` löscht ein Intervall wieder. Das Intervall aus dem obigen Beispiel würde mit folgender Code-Zeile gelöscht werden:

```
clearInterval(intl);
```



`setInterval` und `clearInterval` sind neu in Flash MX und deswegen in ältere Player-Versionen nicht integriert.

Veraltete Funktionen

Im Aktionen-Bedienfeld gibt es noch eine nette Kategorie, die mit **Nicht mehr verwendet** bezeichnet wird. Hier sind hauptsächlich Funktionen aus Flash 4 und früher versammelt, die aus Gründen der Abwärtskompatibilität erhalten wurden.

In Flash 5 und MX wurden diese Funktionen meist durch Methoden von neuen Objekten ersetzt. Sie sollten diese Funktionen wirklich nur noch

im Notfall, das heißt bei absolutem Zwang zur Abwärtskompatibilität, einsetzen. Das sagen wir nicht, weil wir Macromedia helfen wollen, immer neue Versionen durchzusetzen, sondern weil die Gefahr droht, dass diese Funktionen in zukünftigen Flash-Player-Versionen nicht mehr unterstützt werden. Dann müssten Sie Ihre Anwendung umschreiben, und das ist immer aufwändiger, als gleich auf das neue Pferd zu setzen.

In **Nicht mehr verwendet** gibt es zwei Unterpunkte mit Funktionen:

- Unter **Aktionen** verbergen sich drei Funktionen. `ifFrameLoaded` ist eine Methode des Objekts `Movieclip` geworden und wird in Kapitel 12, `Movieclips`, behandelt. `tellTarget` diente dazu, die Adressierung eines `Movieclips` herauszufinden. Sie wird durch `with` ersetzt. Auch hier hilft Kapitel 12, `Movieclips`, weiter. `toggleHighQuality` steuert das Antialiasing im Flash Player. Seine Funktion geht in der Eigenschaft `_quality` auf, die in Kapitel 11, `Objekte, Eigenschaften und Methoden`, beschrieben wird.

Antialiasing (auch Kantenglättung) ist die sanftere Darstellung von Übergängen in einem Bild. Die Kanten werden durch das Einfügen von Farbtönen in Zwischenstufen weicher gemacht.



- **Funktionen** beherbergt hauptsächlich alte Funktionen für die Arbeit mit Strings. Sie sind mittlerweile im Objekt **String** beheimatet. Außerdem gibt es noch zwei mathematische Funktionen, die jetzt Teil des Objekts **Math** sind. Beide Objekte werden im nächsten Kapitel besprochen. Die Funktionalität gleicht weitgehend den alten Funktionen oder übersteigt sie sogar.

10.2 Arrays

Ein Array ist ein Datenspeicher, der mehrere Werte aufnehmen kann. Jeder Wert erhält dabei eine Indexnummer. Der Index beginnt immer bei 0.

So viel zur theoretischen Definition. Jetzt werden wir gleich einmal ein wenig praktischer. Das folgende Array enthält die sieben Wochentage. Es wird einfach einer Variablen zugewiesen. Diese bestimmt, wie Sie im vorangegangenen Kapitel gelernt haben, den Datentyp automatisch:

```
var tage = ["Montag", "Dienstag", "Mittwoch", "Donnerstag",  
"Freitag", "Samstag", "Sonntag"];
```

Die Werte des Arrays werden mit eckigen Klammern (`[]`) begrenzt. Die einzelnen Werte werden durch Kommata voneinander getrennt. Der

Index wird automatisch bestimmt. In diesem Fall hätte »Montag« den Index 0, »Dienstag« den Index 1, »Mittwoch« den Index 2 und so weiter.



Wie Sie in Kapitel 2, Spracheinführung, schon gesehen haben, gibt es auch in JavaScript Arrays. Die Syntax und das Auslesen gleichen sogar der in ActionScript. Allerdings wird in älteren JavaScript-Versionen die eben vorgestellte vereinfachte Syntax zum Erstellen eines Arrays nicht unterstützt.

10.2.1 Arrays erstellen

Bevor wir Arrays auslesen und uns um die Feinheiten kümmern, möchten wir Ihnen noch eine weitere Möglichkeit zeigen, ein Array zu erstellen, und zwar den Konstruktor `new`.

Ein Konstruktor ist prinzipiell eine Methode, ein neues Objekt zu schaffen (zu instanzieren). In der Praxis sieht der Code nur geringfügig anders aus als bei der Vorgehensweise ohne `new`.

```
var tage = new Array("Montag", "Dienstag", "Mittwoch",  
"Donnerstag", "Freitag", "Samstag", "Sonntag");
```

Statt eckiger Klammern werden runde verwendet und das Schlüsselwort `Array` folgt auf `new`.

10.2.2 Datentypen in Arrays

Die verschiedenen Werte in Arrays können alle völlig unterschiedliche Datentypen haben. Folgendes Array bereitet also keine Probleme:

```
var tage = new Array("Montag", 2, true);
```

Natürlich sollten Sie bei Arrays mit völlig verschiedenen Datentypen nicht die Übersicht verlieren, aber prinzipiell spricht nichts gegen die Verwendung von unterschiedlichen Datentypen in einem Array.

10.2.3 Arrays auslesen

Ein Element aus einem Array lesen Sie einfach mit seinem Index aus. Der Index wird dabei in eckigen Klammern (`[]`) angegeben.

```
var tage = new Array("Montag", "Dienstag", "Mittwoch",  
"Donnerstag", "Freitag", "Samstag", "Sonntag");  
trace(tage[2]);
```

Obiger Code gibt also den Mittwoch aus. Bei der Verwendung eines numerischen Index', wie es bei Arrays der Fall ist, liegt die Idee nahe, das Auslesen zu automatisieren. Das Mittel der Wahl sind Schleifen, denn dort lassen sich Zählervariablen einsetzen.

So weit, so gut. Die Zählervariable entspricht dem Array-Index, also muss sie bei 0 beginnen und immer um eins erhöht werden. Was aber ist die Bedingung, bei der die Schleife abgebrochen wird? Natürlich, wenn es keine Elemente mehr im Array gibt. Dazu muss man nur wissen, wie viele Elemente ein Array hat, und dafür gibt es die Methode `length`.

```
var laenge = tage.length;
```

würde beispielsweise die Länge des Arrays `tage` liefern.

Mit diesen Voraussetzungen ist die Schleife schnell geschrieben:

```
var tage = new Array("Montag", "Dienstag", "Mittwoch",  
"Donnerstag", "Freitag", "Samstag", "Sonntag");
```

```
for (var i = 0; i < tage.length; i++) {  
    trace(tage[i]);  
}
```

Fertig ist die Ausgabe des Arrays. In Abbildung 10.23 sehen Sie das Ergebnis.

Beachten Sie, dass die Länge des Arrays immer um eins höher ist als der Index, der ja bei 0 beginnt. Deswegen ist die Bedingung auch **nicht** `i <= tage.length`. Dann würde der Zähler eins zu weit zählen und als letzten Wert `undefined` liefern.

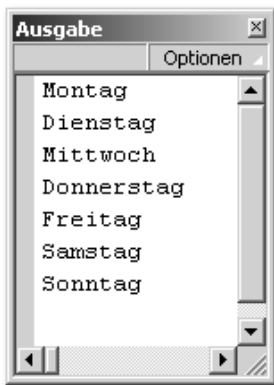


Abbildung 10.23 Der Inhalt des Arrays wurde ausgelesen.

Was in der for-Schleife funktioniert, funktioniert natürlich auch in der while-Schleife.

```
var tage = new Array("Montag", "Dienstag", "Mittwoch",  
"Donnerstag", "Freitag", "Samstag", "Sonntag");  
var i = 0;  
while (i < tage.length) {  
    trace(tage[i]);  
    i++;  
}
```



Die do-while-Schleife verwenden wir hier nicht, da sie bei einem leeren Array einmal durchlaufen würde, obwohl es die Länge 0 hätte. Einmal würde in unserem Beispiel also der Wert `undefined` ausgegeben werden.

Mit Zähler ist das Auslesen eines Arrays ja schon ganz komfortabel. Es gibt mit der for-in-Schleife allerdings auch noch eine Konstruktion, die dafür geschaffen wurde, Arrays (und einige andere Dinge) auszulesen.

```
var tage = new Array("Montag", "Dienstag", "Mittwoch",  
"Donnerstag", "Freitag", "Samstag", "Sonntag");  
for(var i in tage) {  
    trace(tage[i]);  
}
```

Vor dem Schlüsselwort `in` wird eine Variable angegeben, die der Reihe nach alle Indizes des Arrays `tage` aufnimmt, bis das Array komplett durchlaufen worden ist.

Traumhaft? Schon, aber... werfen Sie einmal einen Blick in Abbildung 10.24: Die Werte werden alle in der falschen Reihenfolge ausgegeben.



Abbildung 10.24 Die Wochentage, allerdings in der falschen Reihenfolge

Die for-in-Schleife durchläuft das Array von hinten nach vorne. In vielen Fällen stört das nicht, aber hier ist es nicht so günstig. Man kann schließlich die Woche nicht von hinten anfangen.

Glücklicherweise ist ein Ausweg schnell gefunden. Sie müssen einfach die gelieferten Indexwerte umkehren. Dazu können Sie wieder die Methode `length` verwenden. Subtrahieren Sie von `length` zuerst 1, um sie auf dieselbe Basis zu bringen wie den Index, dann ziehen Sie den in der Schleife gelieferten Index von der bereinigten Länge ab.

```
for(var i in tage) {  
    trace(tage[tage.length-1-i]);  
}
```

Sie glauben uns nicht? In Abbildung 10.25 sehen Sie, dass die Reihenfolge wieder stimmt.

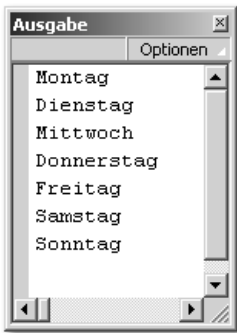


Abbildung 10.25 Die Reihenfolge stimmt wieder.

10.2.4 Bestehende Arrays ändern

Nicht alle Elemente in einem Array müssen einen Wert haben. Sie können sogar ein gänzlich leeres Array definieren. Folgender Code erstellt ein Array mit 20 leeren Elementen. Ein leeres Element hat immer den Wert `undefined`.

```
var leer = new Array(20);
```

Wollen Sie dann einen Wert hinzufügen, weisen Sie einem bestimmten Index einfach eine Zahl zu.

```
leer[15] = "Nicht mehr leer";
```

Obige Zeile weist beispielsweise dem eben noch leeren Array `leer` auf Index-Position 15 eine Zeichenkette zu.

Auch bestehende Arrays lassen sich ändern. Neue Werte überschreiben einfach den Wert, der sich bisher an der Index-Position befand.

```
var tage = new Array("Montag", "Dienstag", "Mittwoch",  
"Donnerstag", "Freitag", "Samstag", "Sonntag");  
tage[3] = "Feiertag";
```

Wissen Sie, welcher Wochentag in diesem Beispiel ein Feiertag ist? Genau, der Donnerstag wurde durch die Zeichenkette »Feiertag« ersetzt.



Wenn Sie mehrere Array-Elemente nach festgelegten Regeln ändern müssen, sind oftmals auch Schleifen das Mittel der Wahl. Folgende Schleife füllt ein Array mit den Zahlen von 1 bis 10.

```
var zahlen = new Array();  
  
for (var i=0; i<10; i++) {  
    zahlen[i] = i+1;  
}
```

10.2.5 In Arrays suchen

Wenn Sie ein Element in einem Array suchen und dessen Index-Nummer herausfinden wollen, gibt es eine einfache Methode. Sie durchlaufen das Array mit einer Schleife und fügen in die Schleife eine Fallunterscheidung mit der Suchbedingung ein. Im folgenden Beispiel suchen wir nach der Position des Donnerstags in unserem Array `tage`.

```
var tage = new Array("Montag", "Dienstag", "Mittwoch",  
"Donnerstag", "Freitag", "Samstag", "Sonntag");  
  
for (var i in tage) {  
    if (tage[i] == "Donnerstag") {  
        trace(i);  
    }  
}
```

In diesem Fall können wir übrigens problemlos eine `for-in`-Schleife verwenden, da die Durchlaufreihenfolge keine Rolle spielt.



Abbildung 10.26 Der Donnerstag hat den Index 3.

Wenn Sie ein nach Größe sortiertes Array haben, können Sie auch eine schnellere Suchmöglichkeit verwenden. Sie prüfen einfach den mittleren Wert des Arrays und stellen fest, ob er größer oder kleiner als der Suchwert ist. Dann halbieren Sie die übrig gebliebene Hälfte und untersuchen hier den mittleren Wert. Diese Methode zeigen wir Ihnen im folgenden Kapitel, wenn wir die Sortierfunktionen für ein Array vorstellen.



10.2.6 Assoziative Arrays

Ein assoziatives Array ist ein Array, dessen Index aus Zeichenketten besteht. Ein Beispiel:

```
var asso = new Array();
asso["vorname"] = "Tobias";
asso["nachname"] = "Hauser";
```

Der Code generiert ein Array, das als Werte die Zeichenketten »Tobias« und »Hauser« erhält. Der Index ist ebenfalls jeweils eine Zeichenkette, nämlich »vorname« und »nachname«. Dieselbe Definition lässt sich auch mit Punkt-Syntax schreiben:

```
var asso = new Array();
asso.vorname = "Tobias";
asso.nachname = "Hauser";
```

Das Aufrufen von Elementen in assoziativen Arrays funktioniert ebenfalls mit beiden Methoden. Für das eben definierte Array gibt also sowohl

```
trace(asso["vorname"]);
```

als auch

```
trace(asso.vorname);
```

»Tobias« aus (der Array-Name `asso` steht übrigens für assoziativ).

Sie können übrigens auch in ein assoziatives Array normale Indexeinträge einfügen:

```
var asso = new Array(2, 3);
asso["vorname"] = "Tobias";
asso["nachname"] = "Hauser";
```

Das Problem ist, dass die Längenangabe mit `length` bei einem Array nur die indizierten Werte, nicht aber die assoziativen berücksichtigt. Obiges Array hätte also eine Länge von 2 statt von 4. Das macht es auch schwie-

rig, assoziative Arrays auszulesen. Die einzige Schleife, mit der das funktioniert, ist also `for-in`.



Assoziative Arrays unterstützen auch keine Sortiermethoden und Ähnliches, wie Sie sie im folgenden Kapitel für indizierte Arrays kennen lernen werden. Das heißt in der Praxis, dass assoziative Arrays Spezialanwendungen vorbehalten bleiben.

10.2.7 Ausblick

Wir sind mit Arrays noch lange nicht am Ende. Es gibt für ein Array-Objekt noch viele Methoden, mit denen man ein Array beispielsweise umkehren oder sortieren kann. Die Methoden werden Sie im folgenden Kapitel kennen lernen.

11 Objekte, Eigenschaften und Methoden

*Alle schlechten Eigenschaften entwickeln sich in der Familie.
Das fängt mit Mord an und geht über Betrug und Trunksucht
bis zum Rauchen.
– Alfred Hitchcock*

Objektorientierung (OO) ist zur Zeit das wichtigste Programmierkonzept für jede Sprache. In diesem Kapitel erklären wir Ihnen zuerst die theoretischen Grundlagen an einem praxisbezogenen Beispiel aus der realen Welt. Erst dann steigen wir in ActionScripts objektorientierte Potenziale ein. Der letzte Teil zeigt dann die vorhandenen Objekte in ActionScript, mit denen sich die meisten Aufgaben in Flash perfekt lösen lassen.

11.1 Begriffe

Klassen, Objekte, Eigenschaften, Methoden, Vererbung – die Schlagworte der Objektorientierung begegnen einem an der Programmierung Interessierten schon fast zwangsläufig. Alle, für die das schon kalter Kaffee ist, sollten im nächsten Abschnitt weiter lesen. Die anderen sind herzlich zu einer Reise ins Innere der Objektorientierung eingeladen.

Ein Objekt stellen Sie sich am besten als etwas sehr Konkretes vor. Ein Hund ist beispielsweise ein Objekt. Er hat bestimmte Eigenschaften, beispielsweise ist sein Fell schwarz und er ist von der Rasse Labrador und er hat Methoden wie `sitz` oder `bellen`. Genauso ist ein Datum ein Objekt. Es hat ebenfalls Eigenschaften wie `Tag`, `Monat` und `Jahr` und Methoden wie `Datum anzeigen`.

Eine Klasse ist ein dem Objekt übergeordneter Begriff, eine Art Gattungssammlung. Beispielsweise entstammen alle Objekte `Hund`, also die einzelnen Hunde, der Klasse `Hund`. Diese Klasse definiert allgemeine Eigenschaften wie `Fell` und `Rasse`, vergibt aber dafür keine Werte. Erst, wenn ein neues Objekt von der Klasse abgeleitet wird, erhält dieses Objekt konkrete Werte für die Eigenschaften. Das Ableiten eines Objekts von einer Klasse nennt man Instanziierung.

Klasse

Objekte

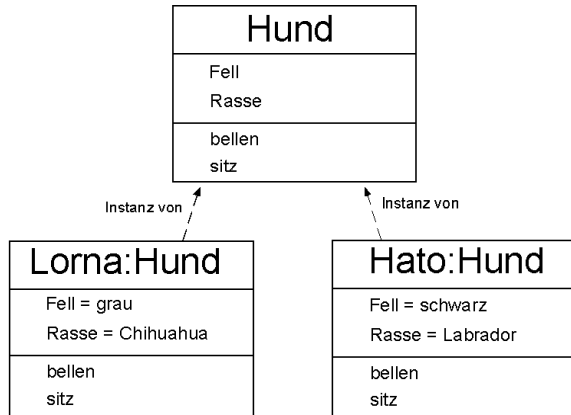


Abbildung 11.1 Lorna und Hato sind Objekte, die von der Klasse Hund abgeleitet wurden.

Abbildung 11.1 zeigt eine Klasse Hund, von der zwei Objekte abgeleitet (instanziiert) wurden, Lorna und Hato. Beide Objekte übernehmen die Eigenschaften und Methoden der Klasse. Die Eigenschaften können bei den Objekten unterschiedliche Werte annehmen. Aber selbst wenn zwei Objekte dieselben Werte haben, sind sie immer noch unterschiedliche, voneinander unabhängige Objekte. Das nennt man Objektintegrität. Zwei schwarze Labradore sind ja auch nicht ein Hund, nur weil sie beide schwarz und eben Labradore sind.



In ActionScript wird der Begriff Klasse als übergeordneter »Objektcontainer« oftmals nicht mehr angegeben. So werden die vordefinierten Klassen als Objekte bezeichnet. Auch bei Eigendefinitionen erzeugt man in ActionScript ein Objekt und erstellt dann Instanzen von diesem Objekt. Diese (mögliche) Sprachregelung hat allerdings keine weiteren Auswirkungen auf die Arbeit, außer dass man aufpassen muss, was gemeint ist: das übergeordnete Objekt oder eine Instanz davon. Wir verwenden hier allerdings die saubere Trennung der Begriffe Klassen und Objekte.

11.1.1 Eigenschaften und Methoden

Ein Objekt hat beliebig viele Methoden und Eigenschaften. Natürlich gibt es auch Objekte, die gar keine Methoden und/oder Eigenschaften haben.

Eine Eigenschaft kann auch als Attribut bezeichnet werden. Die Eigenschaft erhält einen Wert, der von jedem beliebigen Datentyp sein kann. Das Alter wäre beispielsweise eine Eigenschaft für einen Hund, die mit einer Zahl angegeben würde. Eine Eigenschaft kann auch eine Bedingung

enthalten. Für die Klasse Hund wäre das beispielsweise, dass die Eigenschaft Beine (normalerweise ☺) gleich 4 sein muss. Hier spricht man auch von einer Zusicherung.

Methoden sind Anweisungen. Methoden funktionieren ähnlich wie Funktionen. Gerade bei den in ActionScript vorhandenen Objekten und deren Methoden ist eine Unterscheidung zwischen Methode und Funktion nur schwer zu treffen. Auch bei Funktionen können Parameter übergeben werden und Methoden können, müssen aber keine Rückgabewerte haben.

Wir differenzieren hier der Praktikabilität halber so, dass Methoden beim Aufruf immer ein Objekt vorangestellt werden muss, wohingegen Funktionen ohne Objekt aufgerufen werden können (globale Funktionen).

11.1.2 Vererbung

Als Vererbung bezeichnet man den Vorgang, wenn eine Klasse Eigenschaften und Methoden einer anderen Klasse erbt. Die erbende Klasse ist dabei eine Spezialisierung der vererbenden (Generalisierungs-/Spezialisierungsprinzip).

Der Klasse Hund könnte man beispielsweise noch die Klassen Labrador und Chihuahua unterordnen. Der Hund Lorna wäre dann ein Objekt der Klasse Chihuahua, Hato ein Objekt von Labrador. Die beiden Klassen würden beispielsweise die Eigenschaft Fell und die Methode sitz von der übergeordneten Klasse Hund erben.

In Abbildung 11.2 sehen Sie außerdem, dass die Klasse Chihuahua eine Eigenschaft Charakter hat, die der Klasse Labrador fehlt. Das Objekt übernimmt dementsprechend alle Eigenschaften und Methoden der Klasse, von der es instanziiert ist, und aller Klassen, von der diese erbt.

Bei der Klasse Labrador haben wir dasselbe auch noch für die Methode apportieren illustriert.

Klasse

Spezialisierte
Klassen

Objekte

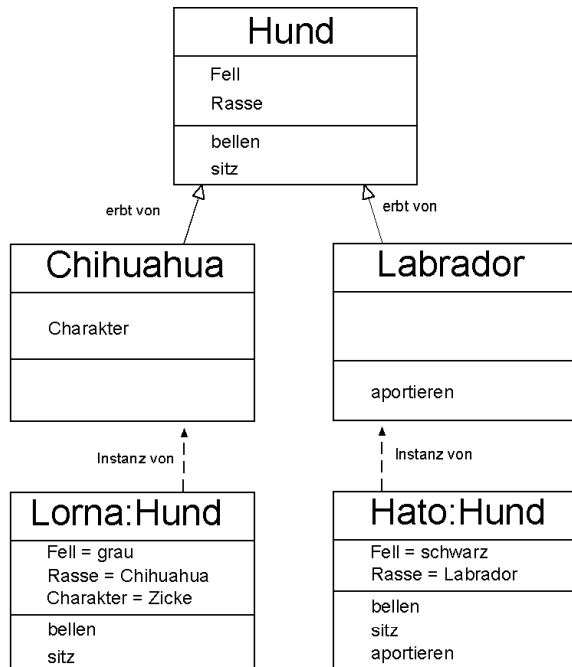


Abbildung 11.2 Vererbung

11.1.3 Polymorphismus

Als Polymorphismus bezeichnet man es, wenn eine Methode (oder Operation) in verschiedenen Klassen unterschiedliche Dinge tut. Ein Beispiel wäre die Methode `fass`. Beim `Hund` führt sie zu einem Angriffsverhalten, bei einem Objekt der Klasse `Baby` dagegen zu einem Greifreflex. Dennoch heißen beide Methoden gleich. Unterschieden werden sie nur, indem sie entweder einem Objekt `Hund` oder einem Objekt `Baby` zugeordnet werden.

11.2 Eigene Objekte, Eigenschaften und Methoden

Nachdem Sie schon einiges über Objektorientierung gehört haben oder bereits wussten, zeigen wir Ihnen, wie Sie in Flash eigene Objekte erstellen und diesen Eigenschaften und Methoden zuweisen.

11.2.1 Klassen

In Flash erstellen Sie Klassen mit dem Schlüsselwort `function`. Ein eigenes Schlüsselwort zur Klassen-Definition, wie Sie es vielleicht aus anderen Sprachen kennen (meist `class`), gibt es in Flash nicht.

Bevor Sie zu Recht fragen: Der Unterschied zu einer normalen Funktion ist natürlich, dass Objekte mit dem Konstruktor `new` instanziiert werden, statt nur mit dem Funktionsnamen aufgerufen zu werden.

Die Syntax zum Erstellen einer Klasse ist sehr einfach:

```
function Name () {  
    this.Eigenschaft = Wert;  
    //Weitere Eigenschaften und Methoden  
}
```

Eigenschaften und Methoden werden innerhalb der Funktion definiert. Wenn wir die Klasse Hund als Beispiel verwenden, die wir bei der Erklärung der Objektorientierung bereits eingesetzt haben, wird allerdings deutlich, dass obige Syntax nur zum Teil sinnvoll ist:

```
function Hund () {  
    Fell = "Schwarz";  
    Rasse = "Labrador";  
}
```

Wir müssten im Prinzip schon konkrete Werte angeben. Das ist aber nicht Sinn der Objektorientierung, da ja aus einer Klasse beliebig viele Objekte abgeleitet werden müssten, die aber alle unterschiedliche Eigenschaften haben. Deswegen übergibt man Werte von Eigenschaften auch gerne als Parameter:

```
function Name (Parameter1) {  
    this.Eigenschaft = Parameter1;  
    //Weitere Eigenschaften und Methoden  
}
```

In unserem Beispiel würde das so aussehen:

```
function Hund (fell, rasse) {  
    this.Fell = fell;  
    this.Rasse = rasse;  
}
```




Das Schlüsselwort `this` vor der Eigenschaft besagt, dass die Eigenschaft zu der hier definierten Klasse gehört, und ist erforderlich. Würde es fehlen, wäre nicht klar, zu welcher Klasse oder welchem Objekt die Eigenschaft gehört.

11.2.2 Der Konstruktor `new`

Um ein Objekt zu instanziiieren, benötigen Sie den Konstruktor `new`. Danach folgen der Objektname und optional noch Parameterangaben.

```
var obj = new Name(Parameter1);
```

Für unser Beispiel mit den Hunden würde das heißen:

```
var Lorna = new Hund("Grau", "Chihuahua");
```

Mit dem Konstruktor `new` erstellen Sie übrigens auch Objekte von vorgefertigten Klassen in Flash, die instanziiert werden müssen. Ein Beispiel wäre das Datums-Objekt.

```
var datum = new Date();
```



Um ein neues Objekt zu erstellen, ist es nicht unbedingt erforderlich, dass die Klasse definiert wurde. Es ist aber sinnvoll, da Sie das Objekt sonst nicht mehrmals verwenden können. Alle Eigenschaften und Methoden müssten jedes Mal neu definiert werden.

11.2.3 Eigenschaften

Wie Eigenschaften definiert werden, haben Sie ja schon gesehen. Wichtig ist vor allem das Schlüsselwort `this`, das die direkte Referenz auf das Objekt angibt.

Folgendes Beispiel definiert die Klasse `Hund` und instanziiert dann mit `new` in der Variablen `Lorna` ein Objekt der Klasse. Das Objekt hat natürlich den Datentyp `Object`.

```
function Hund (fell, rasse) {  
    this.Fell = fell;  
    this.Rasse = rasse;  
}  
  
var Lorna = new Hund("Grau", "Chihuahua");
```

Anschließend kann mit dem Namen des Objekts und (durch einen Punkt abgetrennt) mit dem Namen der Eigenschaft auf die Eigenschaft zugegriffen werden.

```
trace(Lorna.Rasse + ", " + Lorna.Fell);
```

In Abbildung 11.3 sehen Sie die Ausgabe der beiden Eigenschaften.

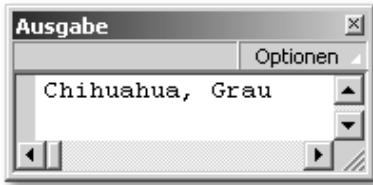


Abbildung 11.3 Die korrekten Werte werden ausgegeben.

Eigenschaften müssen denselben Namenskonventionen folgen wie Variablen, also beispielsweise keine Buchstaben am Anfang, keine Sonderzeichen, Punkte und Ähnliches. Alle Konventionen können Sie in Kapitel 9, ActionScript: Grundlagen im Abschnitt Namenskonventionen nachlesen.



Das Setzen einer Eigenschaft ist genauso einfach wie das Auslesen. Sie benötigen nur das Objekt und die durch einen Punkt abgetrennte Eigenschaft und weisen ihr dann mit dem Zuweisungsoperator (=) einen Wert zu.

Wandeln wir das Beispiel ein wenig ab. Beim Instanzieren des neuen Objekts `Hato` wird nur die Fellfarbe als Parameter übergeben. Die Eigenschaft `Rasse` bleibt noch leer.

```
function Hund (fell, rasse) {  
    this.Fell = fell;  
    this.Rasse = rasse;  
}  
  
var Hato = new Hund("Schwarz");
```

Sie wird dann einfach in der nächsten Zeile festgelegt. Dazu muss nur zuerst das Objekt, dann nach dem Punkt die Eigenschaft angegeben und ihr ein Wert zugewiesen werden.

```
Hato.Rasse = "Labrador";
```

Die folgende Ausgabe gibt wieder Rasse und Fell korrekt an (siehe Abbildung 11.4).

```
trace(Hato.Rasse + ", " + Hato.Fell);
```

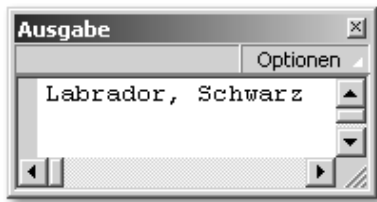


Abbildung 11.4 Die Rasse wurde nachträglich definiert.

Nach demselben Prinzip ändern Sie auch den Wert einer Eigenschaft. Sie weisen ihn einfach neu zu. Die folgende Zeile ändert Hatos Fellfarbe beispielsweise in Dunkelgrau. Bei einer Ausgabe wird für die Eigenschaft dann dieser neue Wert angezeigt.

```
Hato.Fell = "Dunkelgrau";
trace(Hato.Fell);
```

11.2.4 Methoden

Eine Methode wird in ActionScript innerhalb eines Objekts als Funktion definiert. Über die Verwandtschaft von Funktionen und Methoden haben wir ja schon berichtet; in der Praxis sieht das Ganze so aus:

```
function Name () {
    this.Eigenschaft = Wert;
    this.methode = Funktion();
}
```

Die Methode selbst wird in einer Funktion in folgender Form gespeichert:

```
function Funktion () {
    Anweisungen;
}
```

Auch hierfür verwenden wir wieder unser Beispiel mit den Hunden. Wir wollen eine Methode `bell` einbauen. Wie bellt ein Hund in Flash? Ihrer Fantasie sind keine Grenzen gesetzt: Starten Sie einen Soundtrack, geben Sie eine Sprechblase aus oder begnügen Sie sich wie wir für dieses Beispiel mit einer einfachen Textausgabe.

1. Zuerst erstellen wir die Klasse. Die Eigenschaften bleiben unverändert, angefügt wird die Methode `Bell`. Beachten Sie, dass auch hier das Schlüsselwort `this` angegeben werden muss, damit die Methode dem Objekt korrekt zugeordnet wird.

```
function Hund (fell, rasse) {
    this.Fell = fell;
    this.Rasse = rasse;
    this.Bellen = bellen();
}
```

2. Die Methode ruft die Funktion `bellen()` auf. Diese Funktion ist sehr einfach, sie gibt die Zeichenkette »Wuff« aus.

```
function bellen () {
    trace("Wuff");
}
```

3. Nun muss noch ein Objekt instanziiert werden. Lorna ist wieder in Aktion ...

```
var Lorna = new Hund("Grau", "Chihuahua");
```

4. ... und muss sich bemerkbar machen. Die Methode wird einfach mit der Syntax `Objekt.Methode()` aufgerufen. Beachten Sie die runden Klammern hinter der Methode, die angeben, dass es sich um eine Methode handelt.

```
Lorna.Bellen();
```

Das Objekt muss der Methode natürlich vorangestellt werden, da sonst unklar ist, woher die Methode kommt. Sie würde dann unter Umständen wie eine normale Funktion behandelt.

Man kann natürlich auch an eine Methode in einer Funktion Parameter übergeben. Dazu lassen sich sogar Parameter für die Klasse verwenden. Folgender Code unterscheidet bei den Hunden zwischen zwei Rassen und gibt für jede Rasse ein anderes Gebell aus.

```
function Hund (fell, rasse) {
    this.Fell = fell;
    this.Rasse = rasse;
    this.Bellen = bellen(rasse);
}

function bellen (rasse) {
    if (rasse == "Chihuahua") {
        trace("Wuff");
    } else if (rasse == "Labrador") {
        trace("Wau");
    }
}
```

```

}

var Lorna = new Hund("Grau", "Chihuahua");
Lorna.Bellen();
var Hato = new Hund("Schwarz", "Labrador");
Hato.Bellen();

```



Abbildung 11.5 Lorna und Hato bellen unterschiedlich.

Wir wollen Ihnen jetzt noch zeigen, wie Sie ein Objekt mit Methoden versehen, wenn vorher keine Klasse definiert wurde.

1. Dazu erstellen wir ein neues Objekt mit dem Namen `Wauz` (armer Hund, bei dem Namen J).

```
var Wauz = new Object();
```

2. Anschließend fügen wir die Methode/Funktion `bellen` ein.

```
function bellen () {
    trace("Wuff");
}
```

3. Diese Funktion weisen wir dann als Methode `Bellen` dem Objekt `Wauz` zu und rufen sie auf.

```
Wauz.Bellen = bellen;
Wauz.Bellen();
```

Diese Methode hat den Vorteil, dass Sie nicht umständlich eine allgemeine Klasse erzeugen müssen, sondern mit einem Objekt beginnen können. Bei mehreren Objekten wird sie allerdings umständlicher, da für jedes Objekt die Methode neu definiert werden muss. Bei einer Klasse steht sie dagegen allen instanziierten Objekten zur Verfügung.



In ActionScript spricht übrigens nichts dagegen, dass Sie einem Objekt einer Klasse nachträglich weitere Eigenschaften und Methoden hinzufügen. In der reinen objektorientierten Programmierung wäre das allerdings

eine Todsünde. In ActionScript funktioniert dies hauptsächlich, weil es keine strikte Trennung zwischen Methoden, Objekten und Funktionen gibt. Folgendes Beispiel gibt also auch »Wuff« aus:

```
function Hund (fell, rasse) {
    this.Fell = fell;
    this.Rasse = rasse;
}

var Wauz = new Hund("Braun", "Straßenköter");

function bellen () {
    trace("Wuff");
}

Wauz.Bellen = bellen();
```

11.2.5 Alle Eigenschaften auslesen

Oftmals will man überprüfen, welche Eigenschaften ein Objekt bereithält und welche Werte sie haben. Dies funktioniert am einfachsten mit einer for-in-Schleife, die überprüft, welche Elemente in einem Objekt enthalten sind.

Wir legen wieder unser einfaches Beispiel mit den Hunden zugrunde und instanzieren ein Objekt der Klasse Hund.

```
function Hund (fell, rasse) {
    this.Fell = fell;
    this.Rasse = rasse;
}

var Lorna = new Hund("Grau", "Chihuahua");
```

Die Schleife übernimmt alle Elemente des Objekts in eine Variable. Die Elemente sind die Namen der Eigenschaften. Wir geben die Namen aus. Daneben schreiben wir allerdings noch die Werte. Der Zugriff auf die Werte erfolgt in eckigen Klammern wie bei einem Array.

```
for (var aus1 in Lorna) {
    trace(aus1 + ": " + Lorna[aus1]);
}
```

Wie Abbildung 11.6 zeigt, funktioniert dies reibungslos.

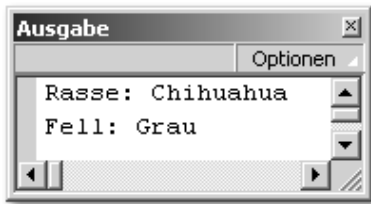


Abbildung 11.6 Die Eigenschaften und Werte wurden ausgelesen.



Methoden eines Objekts erhalten Sie auf dieselbe Weise. Wenn Sie allerdings die Werte der Methoden auslesen, werden diese gleich ausgeführt. Das müssen Sie natürlich bei der Gestaltung der for-in-Schleife berücksichtigen.

11.2.6 Vererbung

Nehmen wir an, Sie haben eine einfache Klasse und bereits ein oder mehrere Objekt(e) der Klasse, so wie in dem folgenden Beispiel mit den Hunden:

```
function Hund(fell, rasse) {  
    this.Fell = fell;  
    this.Rasse = rasse;  
}  
var Lorna = new Hund("Grau", "Chihuahua");
```

Wenn Sie nun zur Klasse, also zum Konstruktor, noch eine neue Methode oder Eigenschaft hinzufügen möchten, liegt es nahe, einfach zu versuchen, außerhalb des Konstruktors noch eine Eigenschaft oder Methode festzulegen.

```
Hund.Beine = 4;
```

Das Problem ist, dass die Objekte diese Eigenschaft oder Methode nicht erben, sie ist also Klassen-intern und sitzt dort fest. Wir wollen aber, dass die Eigenschaft oder Methode auch für die Objekte, also beispielsweise für Lorna, gilt.

Der Schlüssel zur Vererbung in ActionScript ist `prototype`. Jede Klasse (Klassenkonstruktor mit `function`) hat das nicht sichtbare Unterobjekt `prototype`. Wenn Sie diesem Objekt Methoden und Eigenschaften zuweisen, werden sie automatisch an die Objekte vererbt.

Und so funktioniert es in der Praxis:

1. Sie haben eine Klasse.

```
function Hund(fell, rasse) {  
    this.Fell = fell;  
    this.Rasse = rasse;  
}
```

2. Von der Klasse instanziiieren Sie ein Objekt.

```
var Lorna = new Hund("Grau", "Chihuahua");
```

3. Anschließend weisen Sie dem Unterobjekt `prototype` der Klasse eine neue Eigenschaft und eine neue Methode zu.

```
Hund.prototype.Beine = 4;  
Hund.prototype.Bellen = bellen;  
function bellen() {  
    trace("Wuff");  
}
```

4. Eigenschaft und Methode werden vererbt, wie Sie unschwer testen können:

```
Lorna.Bellen();  
trace(Lorna.Beine);
```



Abbildung 11.7 Die neue Methode und die neue Eigenschaft werden vererbt.

Wenn Sie Eigenschaften mit `prototype` vererben, haben diese für alle Objekte die gleichen Werte. Weisen Sie einer vererbten Eigenschaft für ein Objekt einen neuen Wert zu, wird stattdessen eine neue Eigenschaft gleichen Namens nur für das Objekt erstellt.



Vererbte Eigenschaften und Methoden auslesen

Die mit `prototype` vererbten Eigenschaften und Methoden lassen sich auch wieder auslesen. Dazu gibt es für ein Objekt die Eigenschaft `__proto__`, die alle vererbten Eigenschaften und Methoden enthält. Das Auslesen funktioniert mit einer `for-in`-Schleife.

1. Zuerst erstellen wir die Klasse und instanzieren ein Objekt.

```
function Hund(fell, rasse) {  
    this.Fell = fell;  
    this.Rasse = rasse;  
}  
var Lorna = new Hund("Grau", "Chihuahua");
```

2. Dann folgen die vererbte Eigenschaft und die vererbte Methode.

```
Hund.prototype.Beine = 4;  
Hund.prototype.Bellen = bellen;  
function bellen() {  
    trace("Wuff");  
}
```

3. Den Abschluss bildet die Schleife, die die Namen der vererbten Eigenschaft und Methode ausgibt:

```
for (var ele in Lorna.__proto__) {  
    trace(ele);  
}
```

Den Wert der Eigenschaft könnten Sie nun mit `Lorna.__proto__[ele]` aufrufen. Bei der Methode ergäbe dies allerdings nur den Typ.



Natürlich werden die vererbten Funktionen und Eigenschaften mit allen anderen Eigenschaften und Funktionen eines Objekts ausgegeben, wenn auf das Objekt direkt zugegriffen wird. Für unser Beispiel ginge das so:

```
for (var ele in Lorna) {  
    trace(ele + Lorna[ele]);  
}
```

Mehrfachvererbung

Wir wollen nun in der Vererbung aber noch einen Schritt weitergehen und Klassen von anderen Klassen Methoden und Eigenschaften erben lassen.

1. Zuerst erstellen wir eine Klasse `Hund`, die die Methode `Bellen` enthält.

```
function Hund() {  
    this.Bellen = bellen;  
}
```

2. Die Funktion für die Methode `Bellen` kennen Sie bereits:

```
function bellen() {  
    trace("Wuff");  
}
```

3. Eine zweite Klasse ist spezieller als `Hund`. Sie heißt `Labrador` und hat die Eigenschaft `Charakter`.

```
function Labrador() {  
    this.Charakter = "Kinderfreund";  
}
```

4. Nun wollen wir, dass die Klasse `Labrador` von `Hund` erbt. Dazu machen wir das Unterobjekt `prototype` von `Labrador` einfach mit dem Konstruktor `new` zu einem Objekt von `Hund`. Damit erbt die Klasse `Labrador` alle Methoden und Eigenschaften der Klasse `Hund`.

```
Labrador.prototype = new Hund();
```

5. Wir testen dies, indem wir ein neues Objekt von `Labrador` instanzieren und sowohl die Eigenschaft von `Labrador` ausgeben als auch die Methode von `Hund` über das `Labrador`-Objekt aufrufen.

```
var Hato = new Labrador();  
trace(Hato.Charakter);  
Hato.Bellen();
```

Vererbung ist ein mächtiges, aber auch gefährliches Werkzeug. Es ist schnell einmal passiert, dass man unsinnig vererbt. Wenn man dann beispielsweise zur übergeordneten Klasse noch Eigenschaften und Methoden hinzufügt, merkt man auf einmal, dass diese nicht zu den Klassen passen, die erben.



11.2.7 Mehrfachvererbung mit Parametern

Wollten Sie bei der Mehrfachvererbung Parameter mit übergeben, war das bis Flash 5 nicht so einfach möglich. Sie mussten zuerst eine Methode mit der übergeordneten Klasse als Funktion definieren und anschließend diese Methode aufrufen und im Aufruf den Parameter übergeben.

Das folgende Beispiel definiert mit folgender Zeile die übergeordnete Klasse als Methode:

```
this.klasse = Hund;
```

Anschließend wird die Methode aufgerufen und der Parameter, der natürlich bei der Konstruktion des Objekts angegeben werden muss, wird übergeben.

```
this.klasse(fell);

function bellen() {
    trace("Wuff");
}

function Hund(fell) {
    this.Fell = fell;
    this.Bellen = bellen;
}

function Labrador(fell) {
    this.Charakter = "Kinderfreund";
    this.klasse = Hund;
    this.klasse(fell);
}

Labrador.prototype = new Hund();

var Hato = new Labrador("Schwarz");

trace(Hato.Fell);
```



Abbildung 11.8 Die Fellfarbe wurde übergeben.

Flash MX hält eine etwas einfachere Lösung bereit. Mit `super` können in einer Klasse Parameter an die übergeordnete Klasse übergeben werden. Die Vererbung selbst muss zwar nach wie vor mit `Labrador.prototype = new Hund();` angegeben werden, aber man erspart sich bei der Parameterübergabe den Umweg über eine neue Methode.

```

function bellen() {
    trace("Wuff");
}
function Hund(fell) {
    this.Fell = fell;
    this.Bellen = bellen;
}
function Labrador(fell) {
    this.Charakter = "Kinderfreund";
    super(fell);
}
Labrador.prototype = new Hund();

var Hato = new Labrador("Schwarz");

trace(Hato.Fell);

```

Wenn Sie in einer erbenenden Klasse innerhalb einer Methode `super.Methode()` eingeben, wird die gleichnamige Methode aus der vererbenden Klasse aufgerufen. Parameter können optional angegeben werden.



11.2.8 Polymorphie

Polymorphie (oder auch Polymorphismus) begegnet einem in ActionScript, wenn man beispielsweise für zwei Klassen, die von derselben Klasse erben, jeweils gleichnamige Methoden mit unterschiedlichen Funktionsweisen definiert.

Ein Beispiel wäre, dass man für die Klassen Chihuahua und Labrador unterschiedliche Methoden für das Bellen verwendet, da beide unterschiedliche Töne ausstoßen. Beide Methoden heißen aber dennoch Bellen.

```

function Chihuahua() {
}
function Labrador() {
}
Chihuahua.prototype.Bellen = function () {
    trace("Wuff");
};
Labrador.prototype.Bellen = function () {
    trace("Wau");
};

```

```
);
```

```
var Larna = new Chihuahua();  
Larna.Bellen();
```

```
var Hato = new Labrador();  
Hato.Bellen();
```

In Abbildung 11.1 sehen Sie sehr schön, dass beide – eigentlich gleichnamigen – Methoden unterschiedliche Ausgaben produzieren.



Abbildung 11.9 Unterschiedliche Funktionen für das Bellen

11.3 Vorgefertigte Objekte

Man kann die in Flash bereits vorhandenen Objekte und Klassen grob in zwei Bereiche einteilen: in diejenigen, die noch als Objekt instanziiert werden müssen, und in diejenigen, deren Methoden und Eigenschaften direkt zur Verfügung stehen.

Wenn Sie einmal Ihren Blick über das **Aktionen**-Bedienfeld schweifen lassen, sehen Sie als Erstes die Hauptkategorie **Objekte**. Sie haben gut gewählt, hier befinden sich tatsächlich die Objekte von ActionScript.

Unser erstes Thema in diesem Abschnitt sind aber nicht die Objekte, sondern die Eigenschaften, die sich in der Oberkategorie **Eigenschaften** befinden. Dann folgen natürlich die Objekte.

11.3.1 Eigenschaften

Alle Eigenschaften in der Kategorie **Eigenschaften** des **Aktionen**-Bedienfelds beziehen sich auf Movieclips. Alle Eigenschaften? Zwei Ausreißer sind auch vertreten. Für die Eigenschaften von Movieclips verweisen wir Sie auf Kapitel 12, Movieclips und Ereignisse. Die zwei Ausreißer wollen wir allerdings hier erwähnen. Zum Schluss werfen wir noch einen Blick auf ein paar veraltete Eigenschaften.

Vordefinierte Eigenschaften erkennt man daran, dass sie mit einem Unterstrich () beginnen.



Qualität des Films

`_quality` steuert die Wiedergabequalität des Films. Folgende Werte sind möglich:

- ▶ **LOW** – niedrige Qualität. Kein Antialiasing (Kantenglättung durch Einfügen von Farben mit Zwischenwerten),
- ▶ **MEDIUM** – mittlere Qualität. Antialiasing wird verwendet, allerdings nur mit einem 2 * 2-Raster. Das Raster gibt an, auf welcher Fläche Zwischenwerte gesucht werden. Je größer die Fläche, desto mehr Zwischenwerte,
- ▶ **HIGH** – hohe Qualität. Dies ist die Standardeinstellung. Antialiasing mit 4 * 4-Raster. Bitmaps werden bei statischen Flash-Filmen, also nicht per ActionScript neu geladenen Filmen, geglättet, das heißt mit einer Art Kantenglättung versehen,
- ▶ **BEST** – bestmögliche Qualität. Antialiasing mit 4 * 4-Raster. Bitmaps werden immer geglättet.

Die Entscheidung zwischen Qualität und Geschwindigkeit ist im Allgemeinen ein Kompromiss. Die Standardeinstellung **HIGH** bietet eine gute Lösung, aber auch **MEDIUM** kann sinnvoll sein. Vor allem bei text-/typolastigen Seiten ist unter Umständen ein niedrigeres Antialiasing vorteilhaft.



`_quality` ist eine globale Eigenschaft, das heißt Sie können sie einfach aufrufen und direkt ansprechen. Sie benötigen auch kein Objekt davor. Folgende Zeile gibt also die aktuelle Qualität des Flash-Films aus:

```
trace(_quality);
```



Abbildung 1110 Die Qualität unseres Films

Um die Qualität zu ändern, müssen Sie den richtigen Wert als Zeichenkette einfach nur der Eigenschaft zuweisen:

```
_quality = "LOW";
```

Mit obiger Zeile ändern Sie die Qualität des Films beispielsweise in niedrig.

Allein mit dieser Eigenschaft können Sie auch eine Schaltfläche erstellen, die zwischen verschiedenen Qualitätsstufen wechselt. Damit lassen Sie dem Anwender die Wahl.

Vorlage für dieses Beispiel ist die Datei **tennisball2 fla**, die Sie auf der CD-ROM finden. In dieser Datei sehen Sie unseren animierten Tennisball und eine Schrift, da man an der Schrift die Qualitätsunterschiede sehr deutlich sieht.

1. Als Vorbereitung in Flash müssen wir jetzt nur noch die Schaltfläche hinzufügen. Wir haben uns hier für ein großes Q entschieden und legen es auf eine neue Ebene.
2. Das Q muss eine richtig große Schriftart haben und möglichst in Fettdruck dargestellt werden, damit es als Schaltfläche auch wahrgenommen wird. Wir wählen Arial, 50 pt und als Farbe ein dunkles Blau (RGB: (0, 0, 102); hexa: #000066). Es wird in der rechten oberen Ecke der Bühne platziert.

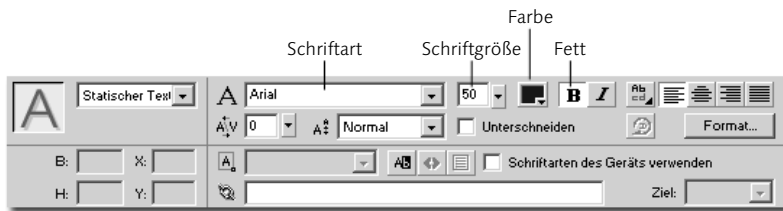


Abbildung 11.11 Einstellungen im Eigenschaftenspektor

3. Wenn Sie Text für eine Schaltfläche verwenden, müssen Sie immer die Entscheidung treffen, wann Sie den Text mit **Teilen** (Menübefehl **Modifizieren • Teilen**; oder: **Strg** + **B**) oder Kontextmenü) in ein normales Objekt umwandeln. Sie sollten es in diesem Fall spätestens dann tun, wenn Sie den Bereich innerhalb des Qs als aktiven Teil der Schaltfläche definieren wollen.
4. Wir teilen das Q gleich, kopieren es und legen eines der beiden mit einem hellen Grau unter das andere Q.
5. Anschließend konvertieren wir beide zusammen in ein Schaltflächen-Symbol.

6. Für den **Drücken**-Zustand erstellen wir ein Schlüsselbild und rücken das obere über das untere Q. Am besten nehmen Sie hierzu die Pfeiltasten.
7. Für den Zustand **Aktiv** erstellen wir ebenfalls ein Schlüsselbild und füllen das Q noch, damit auch der Bereich im Q zur Schaltfläche gehört.

Jetzt kommt ActionScript zum Einsatz.

8. Wir benötigen für das Q-Symbol ein Ereignis. `release` ist hier das richtige.
9. In diesem Ereignis überprüfen wir mit der Eigenschaft `_quality`, ob die Qualität des Films gerade `HIGH`, also hoch, ist. Wenn ja, wird sie in `LOW` geändert.
10. In einer zweiten Anweisung testen wir mit `else if`, ob die Qualität gerade `LOW`, also niedrig, ist. Wenn ja, wird auf `HIGH` umgeschaltet.

```
on (release) {
    if (_quality == "HIGH") {
        _quality = "LOW";
    } else if (_quality == "LOW") {
        _quality = "HIGH";
    }
}
```

Mit dieser einfachen Fallunterscheidung haben Sie einen simplen Schalter erzeugt. Natürlich könnte er auch die Qualität in Stufen regeln. Für diesen Fall wäre eine `switch-case`-Anweisung gut geeignet, die alle möglichen Zustände von `_quality` überprüft und jeweils um eine Stufe nach oben oder unten schaltet.

Die fertige Datei finden Sie auf der CD-ROM unter dem Namen **tennisball_quality fla**.

`_quality` gibt es erst seit Flash 5. Für Flash 4 hieß die Alternative, die aus Gründen der Abwärtskompatibilität immer noch funktioniert, `_highquality`. Sie kann drei Werte annehmen: 0 (kein Antialiasing), 1 (Antialiasing), 2 (optimale Qualität mit Antialiasing und Glättung). Vor Flash 4 konnte das Antialiasing mit der ebenfalls noch gültigen, aber als veraltet eingestuft Funktion `toggleHighQuality()` realisiert werden. Sie wird einfach aufgerufen und schaltet je nach aktuellem Zustand Antialiasing ein oder aus.



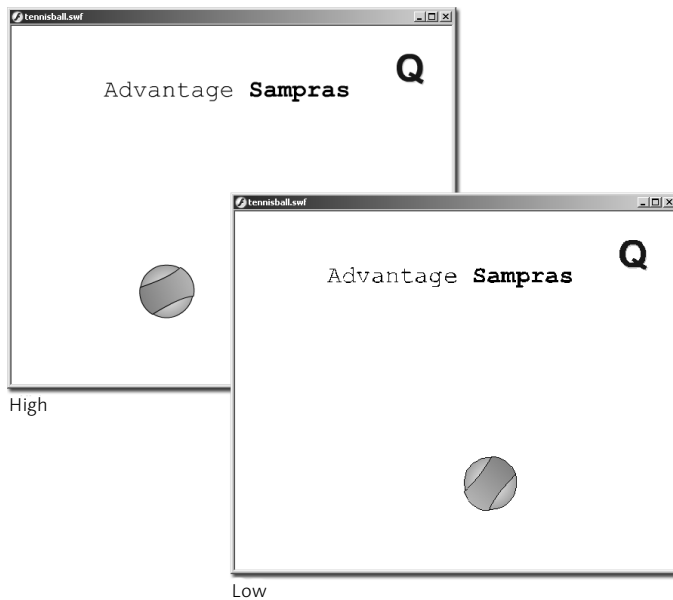


Abbildung 11.12 Qualitätsunterschiede (Hintergrund: HIGH; Vordergrund: LOW)

_soundbuftime

Wenn Sie einen Sound als Streaming-Sound in Flash laden, wird der Sound abgespielt, während er noch geladen wird. Das ist die Definition von Streaming. Das ist natürlich nicht sehr sinnvoll, wenn am Anfang noch überhaupt nichts vom Sound vorhanden ist. Dann kann der Flash-Player den Sound kaum schnell genug nachladen.

Deswegen bestimmen Sie mit der Eigenschaft `_soundbuftime` eine Zeitspanne in Sekunden, in der der Sound vorgeladen wird. Die Eigenschaft gilt für alle Streaming-Sounds in einem Flash-Film. Standardeinstellung sind fünf Sekunden.

`_soundbuftime` lässt sich ganz einfach auslesen:

```
trace(_soundbuftime);
```



Abbildung 11.13 So viele Sekunden wird mit dem Abspielen gewartet.

Gesetzt wird `_soundbuftime` ebenfalls wie jede andere Eigenschaft, nur dass kein Objekt nötig ist, da es sich um eine globale Eigenschaft handelt.

```
_soundbuftime = 4;
```

11.3.2 Allgemeines zu Objekten

Im **Aktionen**-Bedienfeld finden Sie alle Objekte von ActionScript. Sie sind in die vier Kategorien **Core**, **Film**, **Client/Server** und **Authoring** unterteilt. Ein einzelnes Objekt enthält den Konstruktor und zusätzlich noch Unterordner mit Methoden, Eigenschaften und Ereignissen.

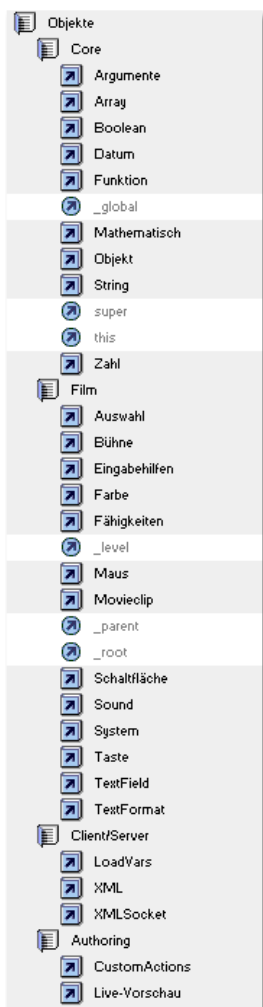


Abbildung 11.14 Die Objekte von ActionScript

Im Laufe dieses Kapitels zeigen wir Ihnen verschiedene ausgewählte Objekte und ihre Anwendungsmöglichkeiten. Das vielleicht wichtigste Objekt `Movieclip` lernen Sie in Kapitel 12, Movieclips, kennen. Wir haben in unserer Auswahl keinen Wert auf Vollständigkeit gelegt, sondern auf Praxisrelevanz. Da Sie fast alle interessanten Dinge in ActionScript mit Objekten realisieren, wäre es auch vermessen, alles auf einmal abdecken zu wollen. Wenn Sie aber erst einmal den Umgang mit einigen grundlegenden Objekten beherrschen, werden Sie schnell eine Vorstellung davon bekommen, wie andere Objekte funktionieren.



Der übergeordnete Datentyp für Objekte ist `Object`. Alle Objekte haben diesen Datentyp.

11.3.3 String

Das String-Objekt enthält, der Name sagt es schon, Methoden, um mit Zeichenketten zu arbeiten. Sie finden es im **Aktionen**-Bedienfeld unter **Objekte • Core • String**. Ein einfacher Operator zum Verbinden von Zeichenketten ist das `+`-Symbol. Sie kennen es bereits. Es gibt allerdings noch viele weitere Methoden. Einige wollen wir Ihnen anhand eines Beispiels zeigen, die anderen finden Sie in einer Tabelle am Ende dieses Abschnitts.

Unsere Beispielanwendung ist sehr einfach aufgebaut. Wir wollen dem Anwender ermöglichen, seinen Namen einzugeben. Der Name wird dann in einem ersten Schritt in Vor- und Nachname getrennt.

Die Vorbereitungen in Flash sind dementsprechend schnell getroffen. In Abbildung 11.15 sehen Sie die benötigten Elemente.



Wenn Sie nicht selbst nachbasteln möchten, finden Sie die fertige Flash-Datei auf der CD-ROM unter dem Namen **eingabemaske fla**.

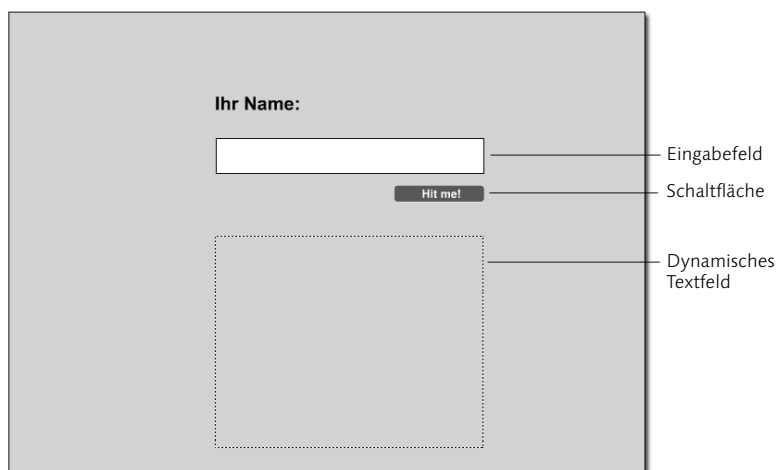


Abbildung 1115 Eingabe- und Ausgabemaske in Flash

Und so werden die einzelnen Elemente des Beispiels erstellt:

- Die Bühne hat eine Größe von 550 * 400 Pixeln und eine cremefarbene Hintergrundfarbe (RGB: (255, 204, 153); hexa: #FFCC99).
- Ein Eingabefeld mit der Variablen `name`. Als weitere Einstellungen lassen wir **Rahmen um Text zeigen** aktiviert und belassen das Feld einzellig. Schriftart ist Arial, die Größe beträgt 24 Punkt.

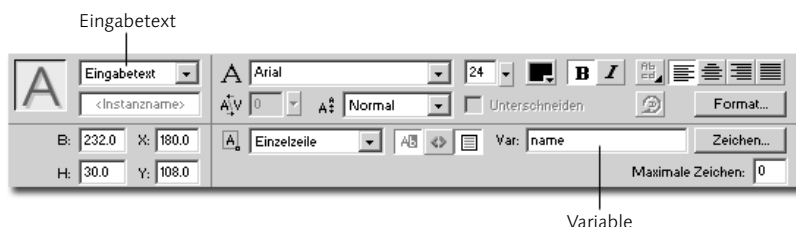


Abbildung 1116 Der Eigenschafteninspektor für das Eingabefeld

- Ein dynamisches Textfeld für die Ausgabe des Textes Es ist **Mehrzeilig**, der Inhalt darf nicht **Auswählbar** sein und als Variable vergeben wir `ausgabe`. Als Schriftart wählen wir Arial in der Größe 16 Punkt.

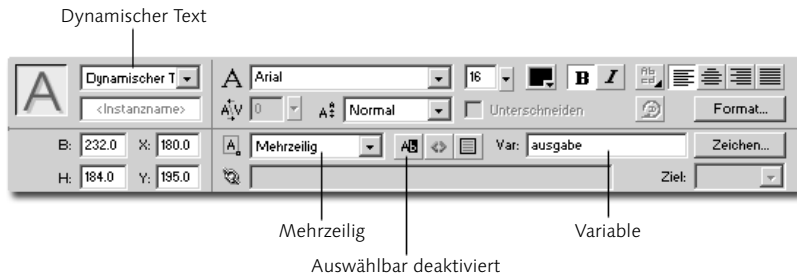


Abbildung 11.17 Der Eigenschafteninspektor für das dynamische Textfeld

- Zwischen den beiden Textfeldern befindet sich eine Schaltfläche, die angeklickt werden kann. Sie besteht aus einem grünen Rechteck (RGB: (47, 128, 70); hexa: # 2F8046) mit abgerundeten Ecken (**Eckradius** 3) und weißem Rand. Die Schaltfläche enthält den statischen Text »Hit me!« und wird in ein Schaltflächensymbol umgewandelt. Für den Zustand **Darüber** erhält sie die Hintergrundfarbe der Bühne als Textfarbe. Für den Zustand **Drücken** wird die Farbe der Schaltfläche zu einem leuchtenden Rot (RGB: (254, 84, 12); hexa: #FE540C).

Nun zur Programmlogik. Sie wird natürlich für die Schaltfläche angelegt. Wenn der Anwender die Schaltfläche loslässt (Ereignis: `release`), soll der Name aufgeteilt und in zwei Zeilen des dynamischen Textfelds geschrieben werden.

1. Der Name, der aus dem Eingabefeld kommt, ist ein String. Insofern sind alle Methoden des String-Objekts anwendbar. Jetzt wird es also interessant. Die Methode, die wir hier wählen, heißt `split(Trennzeichen)`. Sie teilt eine Zeichenkette anhand des als Parameter angegebenen Trennzeichens. Wir verwenden hier also ein einfaches Leerzeichen.

```
var namen = name.split(" ");
```

2. Die mit `split()` getrennten Zeichenketten werden der Reihe nach in das eben erstellte Array `namen` abgelegt. Aus einer Zeichenkette ist also ein Array mit mehreren Zeichenketten geworden. Wir können nun per Index auf die Zeichenketten zugreifen und diese getrennt ausgeben.

```
ausgabe = "Vorname: " + namen[0] + "\n";
ausgabe += "Nachname: " + namen[1];
```

Im Folgenden sehen Sie noch einmal den vollständigen Code:

```
on (release) {  
    var namen = name.split(" ");  
    ausgabe = "Vorname: " + namen[0] + "\n";  
    ausgabe += "Nachname: " + namen[1];  
}
```

In Abbildung 11.18 sehen Sie, dass der Name sauber aufgeteilt wird. Das fertige Beispiel finden Sie unter **text_split.fla** auf der CD-ROM.



Abbildung 11.18 Der Name wird geteilt.

Nun sind wir mit dieser Aufteilung noch nicht wirklich zufrieden. Wir wollen auch den Anwendern gerecht werden, die mehrere Vornamen haben. Hierfür benötigt unsere Schaltfläche ein wenig mehr Programmlogik.

1. Zuerst wird der Name wieder geteilt.

```
var namen = name.split(" ");
```

2. Anschließend geben wir zunächst nur den Vornamen aus. Wir gehen davon aus, dass ihm immer die erste Zeichenkette entspricht. Im Array `namen` hat er also die Position 0.

```
ausgabe = "Vorname: "+namen[0]+"\\n";
```

3. Dann prüfen wir mit einer if-Anweisung, ob das Array `namen` größer als 2 ist, also mehr als zwei Elemente hat.

```
if (namen.length>2)
```



Hier sehen Sie ein Beispiel für eine Eigenschaft in ActionScript, die bei unterschiedlichen Objekten Unterschiedliches liefert. Die Eigenschaft `length` gibt es sowohl für Arrays als auch für Strings. Bei Arrays liefert sie die Anzahl der Elemente, bei Strings die Länge in Zeichen.

4. Sind mehr als zwei Elemente vorhanden, werden alle Elemente nach dem ersten und vor dem letzten mit einer for-Schleife durchlaufen und der Reihe nach ausgegeben. Anschließend wird noch der Nachname ausgegeben.

```
for (var i = 1; i<namen.length-1; i++) {  
    ausgabe += "Weiterer Vorname: "+namen[i] +"\\n";  
}  
ausgabe += "Nachname: "+namen[namen.length-1];
```



Eine for-in-Schleife würde nicht zum gewünschten Ergebnis führen, wenn mehr als ein Vorname eingegeben wird, da in diesem Fall die Ausgabereihenfolge nicht stimmen würde.

5. Sind nur zwei Elemente vorhanden, wird gleich der Nachname ausgegeben. Dann ist seine Index-Position ja bekannt, nämlich 1. Das heißt Sie müssen sie nicht wie vorher umständlich mit `namen.length-1` herausfinden. Das würde allerdings auch funktionieren.

```
ausgabe += "Nachname: "+namen[1];
```

Das ist bereits alles. Unser Skript ist wesentlich flexibler geworden. Die fertige Datei finden Sie auf der CD-ROM unter dem Namen **text_split2 fla**. Im Folgenden ist der Code abgedruckt, um Ihnen noch einmal das Gesamtbild vor Augen zu führen:

```
on (release) {  
    var namen = name.split(" ");  
    ausgabe = "Vorname: "+namen[0]+"\\n";  
    if (namen.length>2) {  
        for (var i = 1; i<namen.length-1; i++) {  
            ausgabe += "Weiterer Vorname: "+namen[i] +"\\n";  
        }  
    }  
}
```

```

        ausgabe += "Nachname: "+namen[namen.length-1];
    } else {
        ausgabe += "Nachname: "+namen[1];
    }
}

```



Abbildung 1119 Mehrere Vornamen werden ausgegeben.

Verbessern kann man nie genug. Deswegen folgt jetzt unser dritter Streich. Wir wollen Vornamen gleich abkürzen und mit Anfangsbuchstaben und Punkt schreiben. Außerdem unterscheiden wir Spitznamen von Vornamen. Spitznamen sollen vom Anwender immer in Anführungszeichen eingegeben werden.

1. Den Anfang kennen Sie schon: erst teilen, dann Vornamen ausgeben.

```

var namen = name.split(" ");
ausgabe = "Vorname: "+namen[0]+"\\n";

```

2. Dann wird wieder unterschieden, ob mehr als Vor- und Nachname vorhanden ist. Wenn nicht, wird, wie oben gesehen, der Nachname ausgegeben, wenn ja, folgt im nächsten Schritt eine etwas komplexere for-Schleife.

```

if (namen.length>2) {

```


3. Die for-Schleife durchläuft alle Elemente zwischen dem ersten und dem letzten des Arrays `namen`.

```
for (var i = 1; i < namen.length - 1; i++)
```

4. Am Anfang der Schleife wird die Länge des gerade durchlaufenen Strings in einer Variablen namens `laenge` gespeichert. Dies dient dazu, den Code übersichtlicher zu machen und nicht immer `namen[i].length` schreiben zu müssen.

```
var laenge = namen[i].length;
```



Wenn Sie Werte und Abfragen immer wieder benötigen, sollten Sie sie in einer Variablen speichern. Sie müssen bei Änderungen dann nur einmal den Variablenwert ändern und haben kürzeren, übersichtlicheren Code.

5. Innerhalb der for-Schleife unterscheidet eine if-Anweisung, ob es sich um einen Spitznamen oder ein Kürzel handelt. Um einen Spitznamen handelt es sich, wenn die jeweilige Zeichenkette mit einem Anführungszeichen beginnt und endet. Um das zu testen, verwenden wir die Methoden `indexOf` und `lastIndexOf`. `indexOf(Suchstring)` liefert die Position eines gesuchten Zeichens in einem String. Wenn ein Anführungszeichen die Position 0 hat, steht es am Anfang. `lastIndexOf(Suchstring)` funktioniert ähnlich, nur dass es von hinten zu suchen beginnt. Es wird also überprüft, ob das Zeichen auf der letzten Position (Länge des Strings minus 1) ein Anführungszeichen ist.

```
if (namen[i].indexOf("\"") == 0 && namen[i].lastIndexOf("\"")  
    == laenge - 1) {
```



Sie haben in diesem Beispiel sicher bemerkt, dass die Indexpositionen bei Strings genauso wie die Elemente bei Arrays mit 0 beginnen, wohingegen die Länge immer die absolute Zahl der Buchstaben liefert. Achten Sie immer darauf, hier liegt eine häufige Fehlerquelle.

6. Gibt es Anführungszeichen, wird der Text innerhalb der Anführungszeichen ausgegeben. Dazu dient die Methode `substring(Start-Index, End-Index)`. Sie liefert den Teil eines Strings zwischen der Start-Index-Position und der End-Index-Position zurück. Wir vergeben als Start Position 1 und als Ende die vorletzte Position. Damit schließen wir die Anführungszeichen aus.

```
ausgabe += "Spitzname: "+namen[i].substring(1, laenge-1)+"\n";
```

Bei der Eingabe von "Air" würde also Air ausgegeben werden.

7. Handelt es sich nicht um einen Spitznamen, tritt die else-Anweisung auf den Plan. Dann wird nur der erste Buchstabe ausgegeben und ein Punkt angehängt. Den ersten Buchstaben erhalten wir mit der Methode `charAt(Position)`. Sie liefert das Zeichen für eine beliebige Index-Position in einem String, für die erste ist das die 0.

```
else {
    ausgabe += "Kürzel: "+namen[i].charAt(0)+".\n";
}
```

Fertig ist das Beispiel. Im Folgenden sehen Sie den vollständigen Code.

```
on (release) {
    var namen = name.split(" ");
    ausgabe = "Vorname: "+namen[0]+"\\n";
    if (namen.length>2) {
        for (var i = 1; i<namen.length-1; i++) {
            var laenge = namen[i].length;
            if (namen[i].indexOf("\\") == 0 &&
                namen[i].lastIndexOf("\\") == laenge -1) {
                ausgabe += "Spitzname: "+namen[i].substring(1,
laenge-1)+"\\n";
            } else {
                ausgabe += "Kürzel: "+namen[i].charAt(0)+".\n";
            }
        }
        ausgabe += "Nachname: "+namen[namen.length-1];
    } else {
        ausgabe += "Nachname: "+namen[1];
    }
}
```

Nun können Sie mit beliebigen Namen testen. Es werden beispielsweise alle Namensbestandteile des bekanntesten Basketballers der Welt perfekt getrennt (siehe Abbildung 11.20).



Abbildung 11.20 Nun werden sogar Spitznamen erkannt.

Die fertige Datei finden Sie unter dem Namen **namenssplitter.fla** auf der CD-ROM zum Buch.



Man könnte bei diesem Beispiel übrigens noch wesentlich mehr Informationen verarbeiten. Beispielsweise vernachlässigen wir bisher Titel wie »Dr.« oder Ähnliches und »von« als Namenszusatz kommt ebenfalls zu kurz. Je mehr man allerdings berücksichtigt, desto komplizierter wird die Programmlogik.

Sie haben im vorigen Beispiel schon einige Methoden des String-Objekts kennen gelernt. In der folgenden Tabelle finden Sie eine Übersicht.

Methode	Beispiel (var j = "Jordan")	Beschreibung
charAt(Position);	var e = j.charAt(3); liefert r	Liefert das Zeichen an der angegebenen Position. Index beginnt immer mit 0. Ist kein Zeichen mehr vorhanden, wird ein leerer String geliefert.
charCodeAt(Position)	var e = j.charCodeAt(3); liefert 100	Liefert den Zahlencode für ein Zeichen. Der Zahlencode bezeichnet die Position im Latin-1 Zeichensatz.

Tabelle 11.1 Die wichtigsten Methoden des String-Objekts

Methoden	Beispiel (var j = "Jordan")	Beschreibung
concat(String); concat(String1, String2, ...);	var m = "AllStar"; var e = j.Concat(" ",m); liefert "Jordan AllStar"	Verbindet zwei oder mehr Strings miteinander. Funktioniert also wie der +-Operator
fromCharCode(Code); fromCharCode(Code1, Code2, ...);	var e = fromCharCode(100); liefert r	Wandelt einen Zahlen-code in ein Zeichen um. Dient bspw. dazu, von charCodeAt produzierte Strings umzuwandeln
indexOf(Zeichen) ; indexOf(Zeichen, Start-Index) ;	j.indexOf("d", 2); liefert 3	Liefert die erste Position von Zeichen. Kann optional erst bei einer bestimmten Index-Position gestartet werden
lastIndexOf(Zeichen- kette, Start-Index) ;	j.lastIndexOf("d", 4); liefert 3	Durchsucht von hinten nach vorne und liefert die Position
slice(Start-Index); slice(Start-Index, End-Index);	var e = j.slice(1,3); liefert or	Extrahiert aus einem String die Zeichenkette ab Start-Index. Optional kann ein End-Index angegeben werden. Die Indexwerte können auch negativ sein. Dann wird vom Ende des Strings aus gezählt.
split(Trennzeichen); split(Trennzeichen, Max.Elemente);	Siehe Beispiel oben	Teilt einen String anhand von Trennzeichen in mehrere Strings und übergibt diese einem Array. Optional lässt sich die maximale Zahl an Elementen angeben, die das Array haben darf. Die übrigen Elemente werden beseitigt.
substr(Start-Index); substr(Start-Index, Länge);	var e = j.substr(1,2); liefert or	Extrahiert aus einem String die Zeichenkette von Start-Index ab. Optional kann eine Länge angegeben werden.

Tabelle 111 Die wichtigsten Methoden des String-Objekts (Forts.)

Methode	Beispiel (var j = "Jordan")	Beschreibung
substring(Start-Index, End-Index);	var e = j.substring(1,3); liefert or	Extrahiert aus einem String die Zeichenkette von Start-Index bis End-Index. Negative Werte sind nicht erlaubt und werden auf 0 gesetzt.
toLowerCase();	var e = j.toLowerCase(); liefert jordan	Wandelt einen String in Kleinbuchstaben um
toUpperCase();	var e = j.toUpperCase(); liefert JORDAN	Verwandelt einen String in Großbuchstaben

Tabelle 111 Die wichtigsten Methoden des String-Objekts (Forts.)



Sie können auch verschiedene Methoden kombinieren. Beispielsweise lässt sich eine Zeichenkette aus einem String mit `substring` heraustrennen und mit `toLowerCase` in Kleinbuchstaben umwandeln.

11.3.4 Array

Über Arrays haben Sie im vorangegangenen Kapitel schon einiges gelesen. Deswegen beschränken wir uns hier darauf, Ihnen in einer Tabelle die Methoden des zugehörigen Objekts `Array` zu zeigen. Das Objekt finden Sie im Aktionen-Bedienfeld unter **Objekte • Core • Array**.

Methode	Beispiel (var a = new Array("Dienstag", "Mittwoch", "Donnerstag"))	Beschreibung
concat(Array); concat(Array1, Array2, ...);	var a2 = new Array("Freitag"); var na = a.concat(a2); liefert Array mit Dienstag, Mittwoch, Donnerstag und Freitag.	Fügt Elemente aus einem Array hinten an ein Array an. Dabei wird allerdings kein Array verändert, sondern ein komplett neues Array geschaffen.
delete(Element);	delete(a[0]); löscht Dienstag	Löscht ein Element eines Arrays. Kann auch Variablen etc. löschen.

Tabelle 112 Die Methoden des Objekts `Array`

Methode	Beispiel (var a = new Array("Dienstag", "Mittwoch", "Donnerstag"))	Beschreibung
<code>join(Trennzeichen);</code>	<code>var s = a.join("");</code> liefert einen String mit den verbundenen Elementen (siehe Abbildung 11.21)	Verbindet alle Elemente eines Arrays zu einem String und fügt ein angegebenes Trennzeichen zwischen den Elementen ein
<code>pop()</code>	<code>var e = a.pop();</code> liefert Donnerstag; Array enthält noch Dienstag, Mittwoch	Entfernt das letzte Element eines Arrays und liefert dessen Wert. Sie können auch Elemente aus einem Array entfernen, indem sie die Eigenschaft <code>length</code> heruntersetzen.
<code>push(Element1, Element2, ...);</code>	<code>a.push("Freitag", "Samstag");</code>	Fügt neue Elemente hinten an ein Array an. Werden keine Elemente angegeben, wird ein leeres Element angefügt. Die Methode gibt gleichzeitig die neue Array-Länge aus.
<code>reverse();</code>	<code>a.reverse();</code> liefert Donnerstag, Mittwoch, Dienstag	Kehrt die Reihenfolge eines Arrays um. Ist in Verbindung mit einer <code>for-in</code> -Schleife nützlich, um Elemente in der richtigen Reihenfolge auszugeben
<code>shift();</code>	<code>var e = a.shift();</code> liefert Dienstag; Array enthält noch Mittwoch, Donnerstag	Entfernt das vorderste Element eines Arrays und übergibt es einer Variablen
<code>slice(Start-Index, End-Index);</code>	<code>var na = a.slice(1,3);</code> neuer Array enthält Mittwoch, Donnerstag	Übergibt die Elemente von Start-Index bis End-Index (exklusive!) an ein neues Array.

Tabelle 11.2 Die Methoden des Objekts `Array` (Forts.)

Methode	Beispiel (var a = new Array("Dienstag", "Mittwoch", "Donnerstag"))	Beschreibung
<code>sort();</code> <code>sort(Vergleich);</code>	<code>a.sort()</code> liefert Dienstag, Donnerstag, Mittwoch	Sortiert die Elemente innerhalb eines Arrays. Ohne Vergleich werden die Elemente nach der Position im Latin-1-Zeichensatz von klein nach groß sortiert (Großbuchstaben kleiner als Kleinbuchstaben, sonst alphabetisch). Vergleich kann eine beliebige Funktion sein
<code>sortOn(Feld);</code>		In Flash MX kann ein Array aus Elementen mit mehreren Feldern bestehen. Diese Funktion wählt eines der Felder als Sortierkriterium.
<code>splice(Start-Index);</code> <code>splice(Start-Index, Anzahl);</code> <code>splice(Start-Index, Anzahl, Element1, Element2, ...);</code>	<code>a.splice(2, 1, "Feiertag");</code> liefert Dienstag, Mittwoch, Feiertag	Entfernt Elemente ab Start-Index. Es kann angegeben werden, wie viele Elemente entfernt werden sollen. Wenn dies unterbleibt, werden alle Elemente bis zum Ende entfernt. Die Elemente können optional auch durch neue Elemente ersetzt werden.
<code>toString();</code>	<code>var e = a.toString();</code> liefert String »Dienstag,Mittwoch,Donnerstag«	Wandelt Array in String um und trennt durch Kommata. Alternative zu <code>join</code>
<code>unshift(Element1, Element2, ...);</code>	<code>a.unshift("Montag");</code>	Fügt Elemente vorn an ein Array an und verschiebt die bestehenden Elemente nach hinten. Werden keine Elemente angegeben, wird ein leeres Element angefügt. Die Methode gibt gleichzeitig die neue Array-Länge aus.

Tabelle 11.2 Die Methoden des Objekts Array (Forts.)

Wenn Sie die Inhalte eines Arrays beispielsweise in einer Datei speichern möchten, können Sie sie mit `join` recht einfach mit Trennzeichen als String ausgeben (siehe Abbildung 11.21). Die Funktion `split(Trennzeichen)` für einen String kann diese dann wieder trennen. Folgender Code verwandelt also ein Array in einen String und wieder zurück. Als Test werden der String und das erste Element des Arrays ausgegeben.



```
var a = new Array("Dienstag", "Mittwoch", "Donnerstag");

var s = a.join("");
trace(s);

var a = s.split("");
trace(a[0]);
```

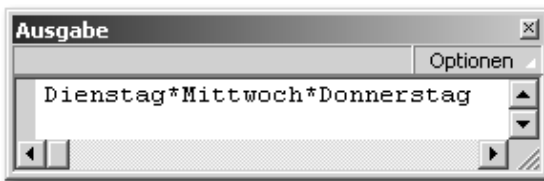


Abbildung 11.21 Mit `join` verbundene Elemente

11.3.5 Andere Datentypen

Arrays und Strings haben eigene Objekte. Was liegt also näher, als auch bei Wahrheitswerten und Zahlen ein eigenes Objekt zu vermuten? Und so ist es tatsächlich.

Für Wahrheitswerte heißt das Objekt **Boolean** und befindet sich im **Aktionen**-Bedienfeld unter **Objekte • Core • Boolean**. Ein neuer Wahrheitswert wird einfach mit dem Konstruktor `new` instanziiert. In Klammern wird der Wert übergeben. Das kann `true` oder `false` beziehungsweise die jeweilige numerische Entsprechung 1 oder 0 sein.

Für Wahrheitswerte gibt es zwei Methoden:

- ▶ `toString` wandelt den Wahrheitswert in eine Zeichenkette (»true« oder »false«) um.
- ▶ `valueOf` liefert den Wert des Boolean, allerdings nicht als Zeichenkette, sondern als Wahrheitswert.

Im Folgenden verwenden wir beide Methoden.


```
var b = new Boolean(false);
trace(b.toString());
trace(b.valueOf());
```

In Abbildung 11.22 sehen Sie, dass beide Methoden die gleiche Ausgabe produzieren. Allerdings ist das nur vordergründig so. Sie müssen beachten, dass es sich bei dem Wert von `toString` auch wirklich um eine Zeichenkette und nicht mehr um einen Wahrheitswert handelt.



Abbildung 11.22 `toString` und `valueOf` haben vordergründig das gleiche Ergebnis geliefert.

Das Objekt `Zahl` befindet sich im **Aktionen**-Bedienfeld auch unter **Objekte • Core**. Und nicht nur das, es besitzt auch dieselben Methoden wie `Boolean`, nämlich `toString`, um eine Zahl in eine Zeichenkette zu verwandeln, und `valueOf`, um den Wert der Zahl zurückzuliefern.

11.3.6 Datum

Das Datums-Objekt ist ein mächtiger Geselle und als solcher dennoch mit ein wenig Vorsicht zu genießen. So ist einer der Autoren neulich auf der Homepage der amerikanischen Profi-Basketballliga NBA vorbeigesurft (<http://www.nba.com>), und was musste er sehen? Einen Flash-Counter, der ein negatives Datum anzeigt. Wie die Playoffs in einer negativen Zeitspanne starten sollen, ist uns zwar ein Rätsel, dennoch halten wir die Seite der NBA ansonsten für ein Meisterstück modernen Webdesigns.

Bevor wir uns in die Tiefen der Datumswerte stürzen, fangen wir erst einmal einfach an.

Ein neues Datum instanzieren Sie mit folgender Zeile:

```
var datum = new Date();
```

Dieses Datumsobjekt übernimmt die Systemzeit des Rechners, auf dem der Flash-Player läuft. Aus diesem Datumsobjekt können Sie nun mit verschiedenen Methoden Teile des Datums auslesen.



Abbildung 11.23 Nanu, der Counter tut nicht ganz, was er soll ...

Wir verwenden zuerst einmal drei Methoden, um die aktuelle Zeit als Digitaluhr in unseren Flash-Film zu übernehmen. Für dieses Beispiel bedarf es wieder einiger Vorbereitungen in Flash. Sie finden die fertige Datei allerdings auch auf der CD-ROM unter dem Namen **uhr.fla**, weswegen wir uns hier kurz fassen.

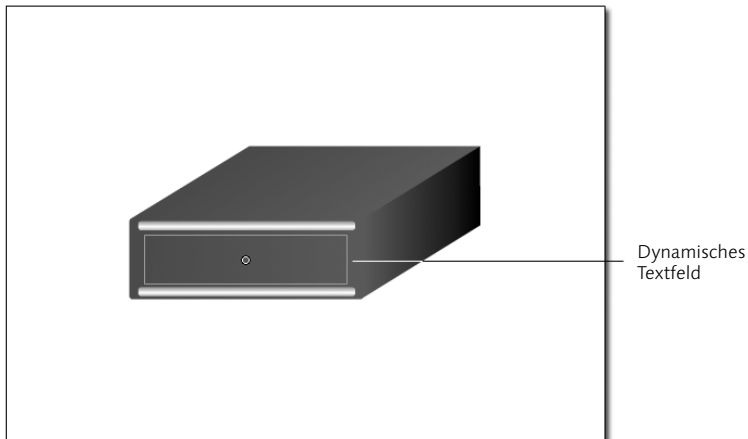


Abbildung 11.24 Ein digitaler Wecker

Die Uhr besteht aus folgenden Elementen:

- aus drei Rechtecken, die so transformiert wurden, dass daraus ein dreidimensionaler Kasten wurde. Jedes Rechteck hat einen Verlauf von dunklem Lila (RGB (101, 84, 112); hexa: #655470) zu jeweils dunkleren Farbwerten. Wir haben hier an jeder Wand ein wenig variiert, um einen ausgewogenen Effekt zu erzielen,

- ▶ aus zwei Rechtecken mit abgerundeten Kanten und einem Verlauf mit der Farbkombination Schwarz, Weiß, Schwarz. Diese Rechtecke bilden die Wülste über und unter der digitalen Anzeige,
- ▶ aus dem Kernstück, einem dynamischen Textfeld mit der Variablen `uhr`. Es ist einzeilig und nicht auswählbar.

Jetzt beginnen wir mit der ActionScript-Programmierung.

1. Die erste Frage ist, wie wir es bewerkstelligen, dass die Uhrzeit der Uhr laufend aktualisiert wird. Wir haben uns hier für einen in Flash MX neuen Weg entschieden, die Funktion `setInterval`. Sie ruft in festgelegten Abständen eine Funktion immer wieder auf. Eine alternative Methode werden Sie im Anschluss kennen lernen.
2. Wenn wir `setInterval` verwenden, müssen wir das Skript zum Anzeigen der Uhrzeit natürlich in eine Funktion legen. Wir nennen diese Funktion `zeit` und führen Sie gleich zu Anfang einmal aus, da `setInterval` erst nach dem Ende des ersten Intervalls tätig wird. Dann rufen wir die Funktion in `setInterval` jede Sekunde (alle 1000 Millisekunden) auf.

```
zeit();
setInterval(zeit, 1000);
```

3. Die Funktion selbst beginnt mit der Instanziierung eines neuen Datums-Objekts.

```
var datum = new Date();
```

4. Als Erstes lesen wir die Stunden aus. Dazu dient die Methode `getHours`.

```
var stunde = datum.getHours();
```

5. `getHours` hat allerdings für unsere Zwecke eine Schwäche. Es zeigt einstellige Uhrzeiten auch nur einstellig an. Deswegen fügen wir noch eine Fallunterscheidung hinzu, die einstelligen Uhrzeiten eine 0 voranstellt.

```
if (stunde<10) {
    stunde = "0"+stunde;
}
```



Beim Datums-Objekt sind öfter solche kleinen Anpassungen nötig. So werden beispielsweise die Wochentage in Zahlen von 0 bis 6 wiedergegeben. Aber dazu später mehr.

6. Nach den Stunden sind die Minuten dran. Dazu dient die Methode `getMinutes`. Auch hier fügen wir eine Fallunterscheidung für einstellige Minuten ein.

```
var minute = datum.getMinutes();
if (minute<10) {
    minute = "0"+minute;
}
```

7. Die Sekunden folgen nach demselben Muster mit der Methode `getSeconds`.

```
var sekunde = datum.getSeconds();
if (sekunde<10) {
    sekunde = "0"+sekunde;
}
```

8. Zum Schluss werden alle drei Uhrzeitangaben in der Ausgabe für das dynamische Textfeld zusammengefasst.

```
uhr = stunde+":"+minute+":"+sekunde;
```

Die fertige Uhr finden Sie auf der CD-ROM unter **digiuhr fla**. Im Folgenden sehen Sie den Code noch einmal im Überblick. In Abbildung 11.25 können Sie das Ergebnis bewundern.

```
zeit();
setInterval(zeit, 1000);
function zeit() {
    var datum = new Date();
    var stunde = datum.getHours();
    if (stunde<10) {
        stunde = "0"+stunde;
    }
    var minute = datum.getMinutes();
    if (minute<10) {
        minute = "0"+minute;
    }
    var sekunde = datum.getSeconds();
    if (sekunde<10) {
        sekunde = "0"+sekunde;
    }
    uhr = stunde+":"+minute+":"+sekunde;
}
```

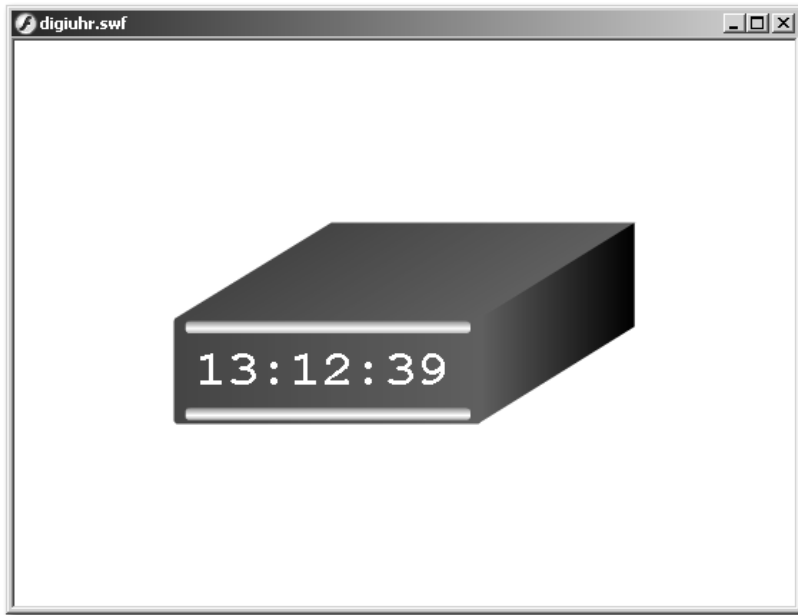


Abbildung 11.25 Die aktuelle Uhrzeit

Unsere Uhr funktioniert recht gut, hat aber einen kleinen Schönheitsfehler: Manchmal werden zwei Sekunden übersprungen. Können Sie sich denken, woran das liegt? Der Intervallabstand beträgt genau eine Sekunde, der Interpreter braucht allerdings für die Funktion auch ein wenig Zeit. Das ist zwar nicht viel, aber genug, um kleine Zeitsprünge zu produzieren. Bevor Sie jetzt versuchen, die Intervallzeit auf einen kürzeren Wert auszutariieren, zeigen wir Ihnen eine andere Variante, die ohne `setInterval` auskommt und deshalb auch mit Flash 5 kompatibel ist.

1. Für diese Lösung wandeln wir Uhr und dynamisches Textfeld in ein Movieclip-Symbol um.
2. Für die Instanz des Movieclip-Symbols weisen wir ein ActionScript zu und verwenden `onClipEvent` mit dem Ereignis `enterFrame`. Das heißt, sobald das Bild betreten wird, wird das Datum ausgegeben. Wie oft geschieht das? Bei einem Movieclip mit einem Bild so oft, wie die Abspielrate des Films gesetzt wurde. Es reicht aber auf jeden Fall, um die Uhrzeit immer aktuell zu halten.

```
onClipEvent (enterFrame) {
```

Das Skript selbst ist ansonsten genauso aufgebaut, wie es oben beschrieben wurde.

```
onClipEvent (enterFrame) {  
    var datum = new Date();  
    var stunde = datum.getHours();  
    if (stunde<10) {  
        stunde = "0"+stunde;  
    }  
    var minute = datum.getMinutes();  
    if (minute<10) {  
        minute = "0"+minute;  
    }  
    var sekunde = datum.getSeconds();  
    if (sekunde<10) {  
        sekunde = "0"+sekunde;  
    }  
    uhr = stunde+": "+minute+": "+sekunde;  
}
```

Die fertige Datei mit dem Movieclip finden Sie unter **digiuhr_movie fla** auf der CD-ROM.

Bevor wir Ihnen in einer großen Tabelle die wichtigsten Methoden des Datums-Objekts näher bringen, noch einige kleine Hinweise zur Arbeit mit dem Datum:

- ▶ Jahreszahlen werden ab 2000 immer vierstellig angegeben. Ein- und zweistellige Jahreszahlen beziehen sich immer auf das 20. Jahrhundert. 6 bezeichnet also 1906, 98 bezeichnet 1998.
- ▶ Monate werden von 0 (Januar) bis 11 (Dezember) angegeben. Sie müssen also eine Anpassung vornehmen, wenn Sie den Monat als Zahl ausgeben.
- ▶ Wochentage werden von 0 (Sonntag) bis 6 (Samstag) angegeben.

Wenn Sie einen Datumswert, beispielsweise den Wochentag oder den Monat, der von `getDay` oder `getMonth` geliefert wird, in ein anderes Format umwandeln möchten, verwenden Sie am besten eine switch-case-Anweisung.



Sie können natürlich auch selbst ein Datum setzen. Die Reihenfolge der Elemente ist festgelegt:

```
datum = new Date(Jahr, Monat, Tag, Stunde, Minute, Sekunde, Mil-
lisekunde);
```

Wollen Sie nur ein einzelnes Element des Datums neu setzen, verwenden Sie einfach das Gegenstück zur get-Methode, eine set-Methode. Für `getDay` wäre das Gegenstück beispielsweise `setDay(Tag)`.

Folgendes Beispiel setzt ein neues Datum und ändert dann die Stunde:

```
var datum = new Date(2002, 10, 15, 12, 30);
trace(datum);
datum.setHours(15);
trace(datum);
```

Ein Blick auf die Ausgabe des Beispiels in Abbildung 11.26 zeigt, dass das Datum mit `setHours` um einige Stunden verändert wurde. Bemerkenswert ist außerdem das Ausgabeformat des Datums-Objekts. Es gibt auch noch die Abweichung von der Greenwich-Standard-Zeit an. Leider ist diese Datenausgabe meist nicht brauchbar. Stattdessen nimmt man einzelne Daten und formatiert diese, wie im obigen Beispiel für die Uhrzeit schon geschehen.

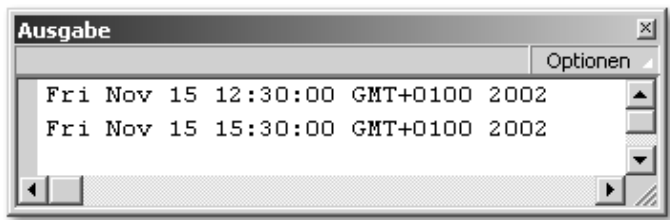


Abbildung 11.26 Zwei Daten

Die folgende Tabelle gibt einen Überblick über die wichtigsten Methoden des Datums-Objekts.

Methode	Beschreibung
<code>getDate();</code>	Ergibt den Tag im Monat. Mögliche Werte reichen von 1 bis 31. Vorsicht, Verwechslungsgefahr mit <code>getDay()</code>
<code>setDate(Tag);</code>	Setzt den Tag im Monat von 1 bis 31
<code>getDay();</code>	Wochentag von 0 (Sonntag) bis 6 (Samstag). Muss meist mit einer switch-case-Anweisung in den Namen des Tages umgewandelt werden

Tabelle 11.3 Wichtige Methoden des Datums-Objekts

Methode	Beschreibung
<code>getFullYear()</code> ;	Liefert die vierstellige Jahreszahl
<code>setFullYear(Jahr)</code> ;	Setzt die Jahreszahl vierstellig. Hat sie weniger Stellen, wird sie so belassen. <code>setFullYear(32)</code> ergibt also 32.
<code>getYear()</code>	Liefert das Jahr seit 1900. Alle Werte aus dem 20. Jahrhundert sind also zweistellig. 2002 ist beispielsweise dreistellig, nämlich 102.
<code>setYear(Jahr)</code> ;	Setzt das Jahr ins Verhältnis zu 1900. Aber Vorsicht, eine Jahreszahl von 103 wird von <code>getFullYear()</code> nicht mehr in 2003 umgewandelt.
<code>getMonth()</code> ;	Liefert den Monat von 0 (Januar) bis 11 (Dezember). Angepasst werden die Werte entweder mit der Addition von 1 oder mit der Umsetzung in Monatsnamen, am besten mit switch case.
<code>setMonth(Monat)</code> ;	Setzt den aktuellen Monat mit 0 (Januar) bis 11 (Dezember)
<code>getHours()</code> ;	Liefert die aktuelle Stunde
<code>setHours(Stunde)</code> ;	Setzt eine Stunde zwischen 0 und 23. Müssen Sie Stunden in ein anderes Format wie 8 am und 8 pm umwandeln, benötigen Sie eine Fallunterscheidung.
<code>getMinutes()</code> ;	Liefert die aktuelle Minutenzahl
<code>setMinutes(Minute)</code> ;	Setzt die aktuellen Minuten
<code>getSeconds()</code> ;	Liefert die aktuelle Sekundenzahl
<code>setSeconds()</code> ;	Setzt die aktuelle Sekundenzahl
<code>getMilliseconds()</code> ;	Liefert die aktuelle Millisekunde
<code>setMilliseconds()</code> ;	Setzt die aktuelle Millisekunde
<code>getTime()</code> ;	Liefert die Zahl der Millisekunden seit dem 1.1.1970 um 0 Uhr. Mit dieser Funktion kann mit Daten gerechnet werden (siehe unten).
<code>setTime(Millisekunden)</code> ;	Setzt das Datum in Millisekunden zur Basis vom 1.1.1970 um 0 Uhr
<code>toString()</code> ;	Erzeugt aus dem Datums-Objekt eine Zeichenkette der Form, wie Sie sie in Abbildung 11.4 sehen.
<code>valueOf()</code> ;	Liefert die Zahl der Millisekunden seit dem 1.1.1970 um 0 Uhr. Wie <code>getTime()</code>

Tabelle 11.3 Wichtige Methoden des Datums-Objekts (Forts.)



Die UTC-Zeit entspricht der Greenwich Mean Time (GMT). Jede Methode zur Gewinnung eines Datums gibt es auch mit dem Zusatz UTC. Damit wird die Standardzeit statt der Systemzeit verwendet. `getUTCHours` liefert beispielsweise die Stunden in Standardzeit. Man verwendet diese Zeit, wenn man einen einheitlichen Standard auf allen Rechnern der Anwender erreichen möchte.

Zum Abschluss dieses Abschnitts zeigen wir Ihnen noch, wie Sie mit Daten rechnen können. Nehmen wir ein Beispiel. Sie möchten gerne wissen, wie viel Zeit vom 3. Mai 2002 bis zum 4. August 2002 in Tagen vergeht.

- Dafür brauchen Sie zuerst einmal zwei Datums-Objekte.

```
var datum1 = new Date(2002, 7, 4);  
var datum2 = new Date(2002, 4, 3);
```

- Anschließend ziehen Sie einfach die zwei Daten voneinander ab. Die Datenwerte werden dabei automatisch in Millisekunden vom 1.1.1970 um 0 Uhr verwendet, da nur dieses Format sich zum Rechnen eignet.

```
var e = datum1 - datum2;
```



Die Datumsumrechnung in Millisekunden kann auch mit der Methode `getTime` von Hand erfolgen.

- Der Rückgabewert ist ein Wert in Millisekunden. Diesen müssen Sie nun noch in Tage umrechnen. Wir teilen dafür den Wert einfach durch 1000 zur Umrechnung in Sekunden, durch 60 zur Umrechnung in Minuten, noch einmal durch 60 für die Stunden und durch 24 für die Tage.

```
e = e / (1000 * 60 * 60 * 24);  
trace(e);
```

In Abbildung 11.27 sehen Sie die Anzahl der Tage.



Abbildung 11.27 So viele Tage sind es ...

11.3.7 Mathematische Methoden und Konstanten

Ihhh, Mathe!

Sie sind Designer? Sie hassen Mathematik und haben sie schon immer gehasst. Keine Angst, auch für den bekennenden Nichtmathematiker finden sich in diesem Abschnitt interessante Methoden. Sie müssen ja nicht in die mathematischen Tiefen von ActionScript eintauchen. Nützlich ist es allerdings manchmal schon. So sind Vektoranimationen eigentlich nichts anderes als geometrische Figuren, die natürlich mit den Grundfunktionen der Geometrie hervorragend beschrieben werden können.

Konstanten

Beginnen wollen wir allerdings nicht mit den mathematischen Methoden, sondern mit den Konstanten in ActionScript. Konstanten sind festgelegte Werte, die eine bestimmte Bedeutung haben.

Konstanten werden immer mit vorangestelltem Math-Objekt aufgerufen. `Math.SQRT2` liefert beispielsweise die Wurzel von 2.

Sie finden übrigens alle mathematischen Konstanten auch im **Aktionen-**Bedienfeld unter **Objekte • Core • Math • Konstanten**.



Konstante	Wert (in Flash)	Beschreibung
E	2.71828182845905	Eulersche Zahl. Die Basis des natürlichen Logarithmus'
LN2	0.693147180559945	Der natürliche Logarithmus von 2
LN10	2.71828182845905	Der natürliche Logarithmus von 10
LOG2E	1.44269504088896	Logarithmus von 2
LOG10E	0.434294481903252	Logarithmus von 10
PI	3.14159265358979	Kreiszahl PI zur Berechnung des Durchmessers
SQRT2	1.4142135623731	Wurzel aus 2
SQRT1_2	0.707106781186548	Wurzel aus ½. Kehrwert der Wurzel aus 2

Tabelle 11.4 Mathematische Konstanten

Methoden

Die Methoden des `Math`-Objekts sind sehr vielfältig und mächtig. Im Folgenden zeigen wir Ihnen, wie Sie mit ein paar dieser Methoden arbeiten. Am Ende des Abschnitts finden Sie eine Übersicht über alle Methoden.

Wir beginnen mit einem einfachen Beispiel. Wir ziehen mit der Methode `sqrt` die Wurzel aus der Zahl 9 und potenzieren das Ergebnis wieder mit 2, machen also die Wurzelziehung rückgängig.

```
var e = Math.sqrt(9);  
e = Math.pow(e, 2);  
trace(e);
```

Das Ergebnis ist 9. Sie sehen auch, dass Flash MX absolut exakt rechnet. Wo in Flash 4 noch Näherungswerte ausgegeben werden, wird hier die zweimalige Umrechnung sauber durchgeführt.

Eine weitere wichtige Anwendung des `Math`-Objekts ist die Bildung von Zufallszahlen. Zufallszahlen benötigt man in der Programmierung recht häufig.

- Die mathematische Methode `random` erzeugt eine Zufallszahl zwischen 0 und 1.

```
var z = Math.random();
```

- Eine solche Zufallszahl ist allerdings nicht so praktisch. Man kann sie nun auf einen bestimmten Wertebereich bringen. Wenn wir sie beispielsweise mit 10 multiplizieren, erhalten wir Zufallszahlen zwischen 0 und 10.

```
z *= 10;
```

- Jetzt können wir das Ganze noch mit `round` runden.

```
var e = Math.round(z);
```

So haben wir also auf einfache Art und Weise zehn Zufallswerte erhalten. In Abbildung 11.28 sehen Sie die ursprünglichen Zufallszahlen und das Ergebnis nach der Umwandlung an ein paar Beispielen.

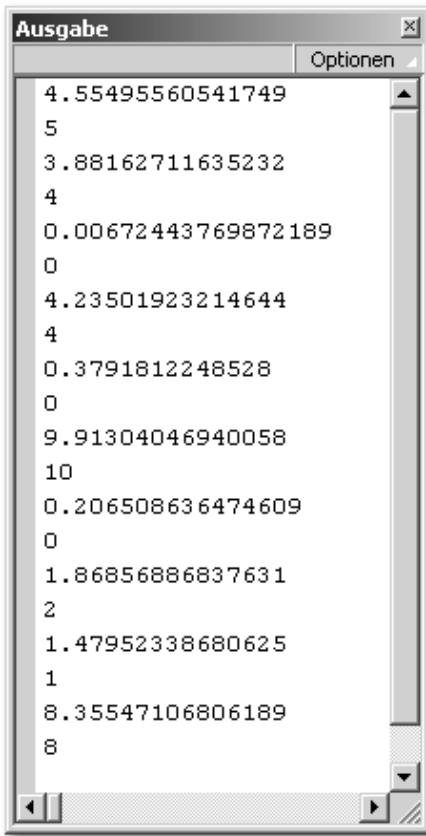


Abbildung 11.28 Zufallszahlen vor und nach der Umwandlung

Es müssen nicht immer 10 Zufallszahlen sein. Die Methode funktioniert natürlich auch für jede beliebige andere Anzahl. Multiplizieren Sie statt mit 10 einfach mit einer anderen Zahl. Benötigen Sie Werte, die nicht bei 0 beginnen, addieren Sie einen beliebigen Wert.

Folgender Code liefert beispielsweise fünf Zufallszahlen zwischen 10 und 15:

```
var z = Math.random();  
z *= 5;  
var e = Math.round(z) + 10;
```

Ein letztes Anwendungsgebiet mathematischer Funktionen, auf das wir Ihre Aufmerksamkeit lenken möchten, ist die Geometrie. Mit Sinus, Kosi-

nus und Konsorten haben Sie ein Set trigonometrischer Funktionen und die Kreiszahl π steht als Konstante zur Verfügung.

Bevor Sie die trigonometrischen Funktionen aber nutzen können, müssen wir Ihnen noch das Konzept des Bogenmaßes erklären, denn die Werte für $\cos$, $\sin$, $\tan$ werden im Bogenmaß angegeben. acos , asin , atan liefern Werte im Bogenmaß zurück.

Das Bogenmaß ist die Länge des Kreisbogens, den der Winkel aus dem Einheitskreis mit Gesamtumfang $2 * \pi$ herauschneidet. In Abbildung 11.29 sehen Sie das Bogenmaß für einen 90° -Winkel.

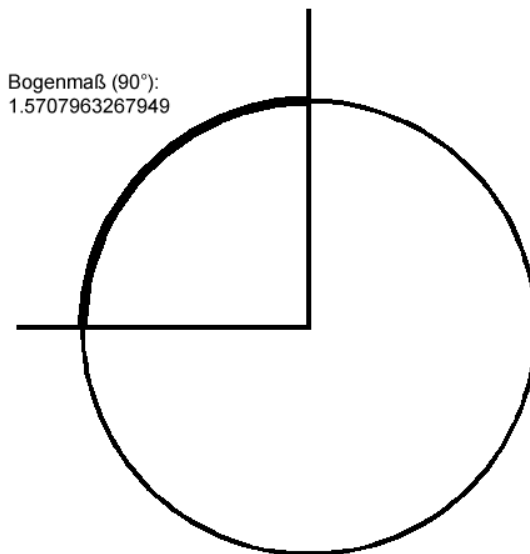


Abbildung 11.29 Das Bogenmaß für einen 90° -Winkel

Die Umwandlung in ein Bogenmaß ist sehr einfach. Man multipliziert den Winkel mit π und teilt durch 180 (360° ist der Winkel eines Kreises, $2 * \pi$ der Umfang).

```
var bm = Math.PI * Winkel / 180;
```

Die Rückumwandlung geschieht mit dem Umkehrbruch:

```
var w = 180 * Bogenmaß / Math.PI;
```

Wir haben dies für ein einfaches Beispiel umgesetzt und rechnen einen 90° -Winkel in Bogenmaß um und wandeln dieses danach wieder in einen Winkel um.

```
var b = Math.PI * 90 / 180;  
trace(b);  
var w = 180 * 1.5707963267949 / Math.PI;  
trace(w);
```

Wenn Sie das Ergebnis in Abbildung 11.30 betrachten, sehen Sie, dass sich bei der Rückumwandlung ein kleiner Rundungsfehler einschleicht. Benötigen Sie keine genaueren Winkel, sollten Sie diese in einem solchen Fall mit der Methode `round` runden.

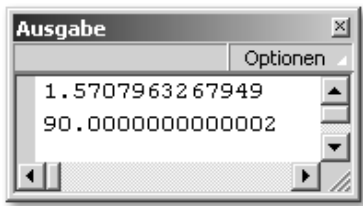


Abbildung 11.30 Bogenmaß und Winkel

Sie haben in diesem Abschnitt bereits einige mathematische Methoden kennen gelernt. In der folgenden Tabelle werden diese und einige andere noch einmal zusammengefasst.

Methode	Beispiel	Beschreibung
<code>abs(Zahl);</code>	<code>Math.abs(-2);</code> ergibt 2	Betrag einer Zahl
<code>ceil(Zahl);</code>	<code>Math.ceil(2.57);</code> ergibt 3 <code>Math.ceil(-2.57);</code> ergibt -2	Erhöht auf die nächste ganze Zahl
<code>exp(Zahl);</code>	<code>Math.exp(2);</code> ergibt 7.38905609893065	Exponentialfunktion zur Basis e (Eulersche Zahl, siehe Konstante <code>Math.E</code>). Also e^x , wobei x der angegebenen Zahl entspricht
<code>floor(Zahl);</code>	<code>Math.floor(2.57);</code> ergibt 2 <code>Math.floor(-2.57);</code> ergibt -3	Ergibt die nächstkleinere Zahl

Tabelle 11.5 Mathematische Methoden

Methode	Beispiel	Beschreibung
<code>log(Zahl);</code>	<code>Math.log(2);</code> ergibt 0.693147180559945	Logarithmus
<code>max(Zahl1, Zahl2);</code>	<code>Math.max(2,3);</code> liefert 3	Liefert die größere Zahl von beiden
<code>min(Zahl1, Zahl2);</code>	<code>Math.min(2,3);</code> liefert 2	Liefert die kleinere Zahl von beiden
<code>pow(Basis, Zahl);</code>	<code>Math.pow(2,3);</code> liefert 8	Zahl zur Basis potenzieren
<code>random();</code>		Zufallszahl zwischen 0 und 1
<code>round(Zahl);</code>	<code>Math.round(2.4);</code> ergibt 2 <code>Math.round(2.6);</code> ergibt 3	Rundet eine Zahl auf die nächste ganze Zahl auf oder ab
<code>sqrt(x);</code>	<code>Math.sqrt(9);</code> Liefert 3	Quadratwurzel
<code>sin(Winkel);</code>	<code>Math.sin(Math.PI/2);</code> ergibt 1	Sinus
<code>cos(Winkel);</code>	<code>Math.cos(Math.PI);</code> ergibt -1	Kosinus
<code>tan(Winkel);</code>	<code>Math.tan(0);</code> ergibt 0	Tangens
<code>asin(Zahl);</code>	<code>Math.asin(0);</code> ergibt 0	Arkussinus. Zahl muss zwischen -1 und 1 liegen. Liefert ein Bogenmaß
<code>acos(Zahl);</code>	<code>Math.acos(1);</code> ergibt 1	Arkuskosinus. Zahl muss zwischen -1 und 1 liegen. Liefert ein Bogenmaß
<code>atan(Winkel);</code>	<code>Math.atan(90);</code> ergibt 1.55968567289729	Arkustangens
<code>atan2(x-Koor., y-Koord);</code>	<code>Math.atan2(100,100);</code> ergibt 0.785398163397448	Winkel eines Punktes relativ zum Ursprung des Koordinatensystems. Zurückgegeben im Bogenmaß

Tabelle 11.5 Mathematische Methoden (Forts.)

12 Movieclips

Botschaften soll man per Telex verschicken, aber nicht in einen Film verpacken.

– Jerry Lewis

Für die ActionScript-Programmierung sind Movieclips eines der wichtigsten Elemente in Flash. Der Grund dafür liegt auf der Hand: Movieclips lassen sich mit ActionScript perfekt steuern und erzeugen.

Ein weiteres Thema dieses Kapitels sind besondere Ereignisse, wie beispielsweise das Drücken bestimmter Tasten.

12.1 Movieclips

Movieclips heißen so, seit sie in Flash MX (noch?) nicht in Filmsequenzen übersetzt wurden. Diese kleine Namensänderung ändert nichts an der Bedeutung.

Ein Movieclip hat, wie in Kapitel 8, Basics, bereits erwähnt wurde, eine eigene Zeitleiste. Wie müssen Sie sich das vorstellen? Streng hierarchisch. Der Hauptfilm hat eine Hauptzeitleiste. Unterhalb dieser Zeitleiste kann es beliebig viele Movieclips mit eigenen Zeitleisten geben. Jeder dieser Movieclips kann weitere Movieclips aufnehmen, so dass eine hierarchische Struktur entsteht.

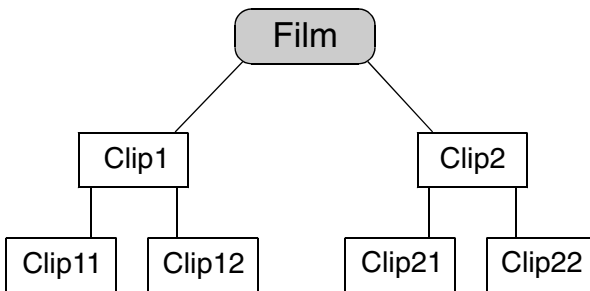


Abbildung 12.1 Eine hierarchische Struktur aus verschiedenen Movieclips

Der Einfachheit halber lassen wir hier absichtlich die Möglichkeit weg, mit mehreren Filmen auf unterschiedlichen Levels (Stufen, Ebenen) zu arbeiten. Mehr dazu finden Sie im Abschnitt »Externe Filme«.



Movieclips können Sie für alles einsetzen: für Navigationselemente, Animationen oder einfach nur für ActionScript-Code, den Sie auslagern möchten.



Gerade die so genannten unsichtbaren Movieclips werden häufig eingesetzt. Sie sollten sie nur auf eine eigene Skript-Ebene legen, damit Sie die unsichtbaren Movieclips auch wieder finden, ohne unter den übrigen Elementen ein Chaos zu veranstalten.

Movieclips werden in Flash als Symbol erzeugt oder aus bestehenden Elementen in ein Symbol konvertiert. Die Movieclip-Symbole befinden sich gesammelt in der Bibliothek (Sie können sie einblenden mit: **Fenster • Bibliothek** oder mit **[F11]**).

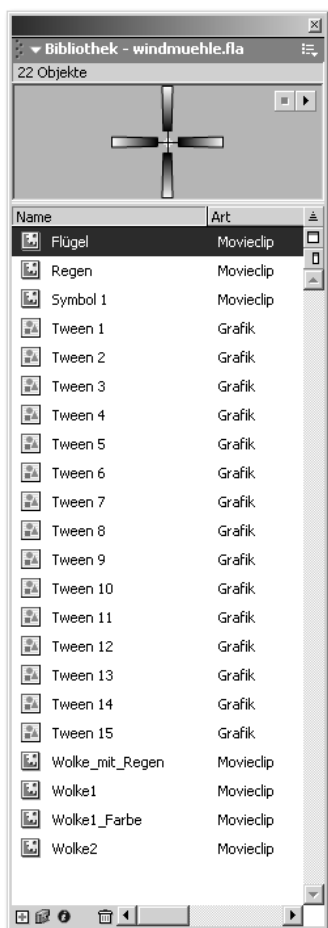


Abbildung 12.2 Movieclips in der Bibliothek

Um eine Instanz eines Movieclips zu erstellen, ziehen Sie ihn einfach auf die Bühne. Im Eigenschafteninspektor vergeben Sie dann einen Instanznamen. Über diesen Namen können Sie auf den Movieclip zugreifen und ebenfalls alle Movieclip-Eigenschaften und -Methoden nutzen.

Die Namensgebung für Instanzen folgt den Regeln für Variablen. Sie können diese im Abschnitt »Namenskonventionen« in Kapitel 9, Action-Script-Grundlagen, nachlesen.

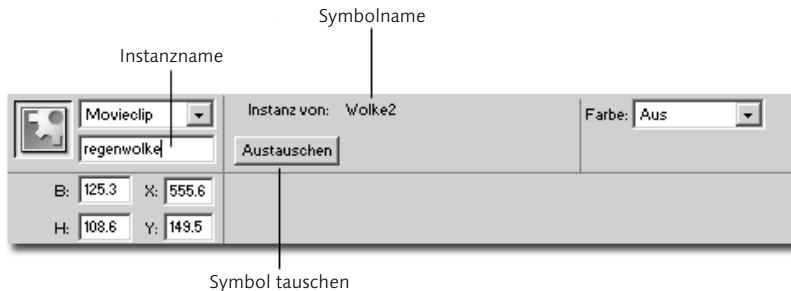


Abbildung 12.3 Instanznamen eintragen

Sie können Movieclip-Instanzen auch per Eigenschafteninspektor in eine Schaltflächen- oder Grafikinstanz verwandeln. In diesem Fall funktionieren allerdings Skripte innerhalb der Zeitleiste des Movieclip-Symbols nicht mehr und auch die übrigen Vorteile gehen verloren. Das Symbol in der Bibliothek bleibt allerdings ein Movieclip-Symbol.

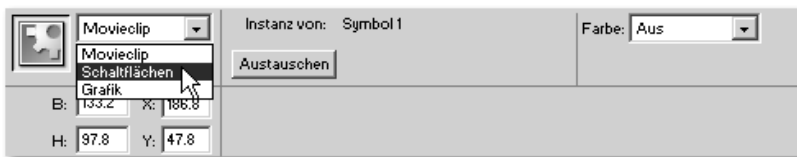


Abbildung 12.4 Die Art der Instanz lässt sich ändern.

Es gibt eine mehr oder weniger akademische Diskussion darüber, ob Movieclips auch Objekte darstellen. Dagegen spricht, dass Movieclips keine eigene Konstruktorfunktion haben, sich also nicht mit `new` instanzieren lassen. Ein weiteres Argument ist, dass Movieclips keine Klasse zugrunde liegt, sondern der eigene Datentyp **Movieclip**. Dieser Datentyp verhält sich auf der anderen Seite wie ein Objekt. Er hat Eigenschaften und Methoden, die sich mit derselben Syntax ansprechen lassen, die für Objekte gilt.

Wir scheren uns hier nicht um diese Diskussion, sondern sehen Movieclips einfach als eine Art Objekt, dem zwar manches fehlt, aber nichts, was Sie für Ihre Arbeit vermissen würden.

12.1.1 Ein Beispiel

Bevor wir uns eingehend mit Movieclips im Allgemeinen und mit verschachtelten Movieclips im Besonderen beschäftigen, möchten wir Ihnen ein Beispiel vorstellen, das wir in diesem Kapitel noch öfter verwenden werden. Die Beispieldatei finden Sie unter dem Namen **windmuehle fla** auf der CD-ROM.

Es handelt sich um eine Windmühle, die sich dreht. Gleichzeitig fliegen zwei Wolken durch die Gegend. Die eine ändert zufallsgesteuert ihre Farbe, die andere lässt es regnen.

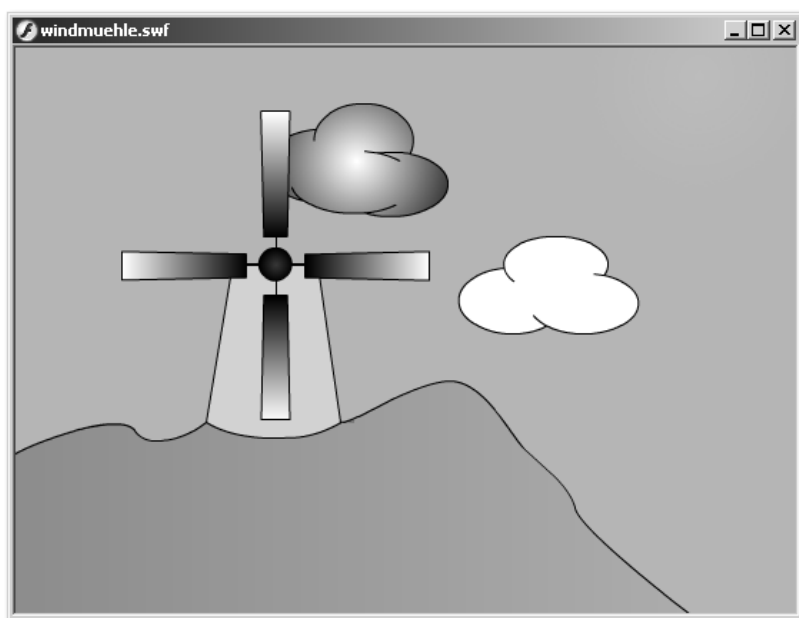


Abbildung 12.5 Windmühle und Wolken

Wir erklären kurz die einzelnen Elemente der Flash-Datei, bevor wir auf die in dieser Datei verschachtelten Movieclips eingehen. Die Elemente sind in verschiedene Ebenen unterteilt. In der Hauptzeitleiste hat der Film nur ein Bild, das mit `gotoAndPlay(1)` im Skript-Kanal immer wieder aufgerufen wird. Die Bewegungen werden alle in Movieclips ausgelagert.

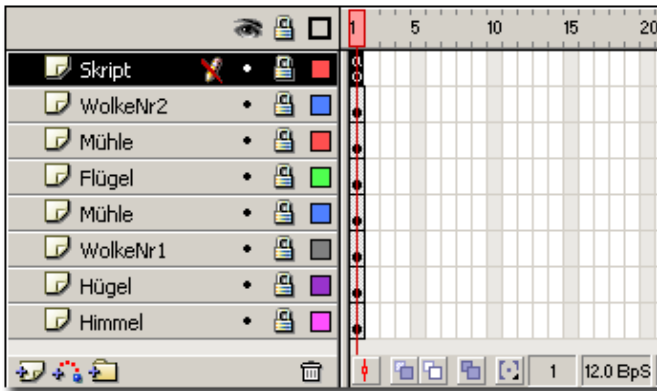


Abbildung 12.6 Die Ebenen in der Zeitleiste

Um die einzelnen Elemente des Films kennen zu lernen, entsperren Sie einfach immer die jeweilige Ebene, mit der Sie sich näher beschäftigen möchten.



Nun aber zu den einzelnen Vektorgrafik-Elementen. Eine Übersicht darüber, was es alles gibt, erhalten Sie schon durch einen Blick auf die verschiedenen Ebenen in der Zeitleiste (siehe Abbildung 12.6). Die folgende Aufstellung ist nicht als komplette Anleitung zum Erstellen des Flash-Films gedacht, sondern soll hauptsächlich einen Überblick über die verschiedenen Elemente und einen Einblick in die Möglichkeiten von Flash geben:

- Die Hintergrundfarbe des gesamten Clips ist ein helles Blau (RGB: (102, 204, 255); hexa: #66CCFF).
- In der Ebene **Himmel** haben wir ein Rechteck aufgezogen und dieses mit einem radialen Verlauf gefüllt. Die Farbe in der Mitte ist Gelb (RGB: (227, 227, 53); hexa: #E3E335) und soll die Sonne symbolisieren. Die Farbe, in die die Sonne verläuft, ist das Hintergrundblau. Den Verlauf haben wir mit dem Füllungstransformationswerkzeug angepasst, mit dem Frei-transformieren-Werkzeug haben wir das Rechteck an die Position bugsiert, an der wir es haben wollten.
- Der Hügel wurde mit dem Stiftwerkzeug gezeichnet und enthält einen linearen Verlauf zweier Grünwerte (#497A4F und #BBE9BA). Hier sind Ihrer Fantasie aber keine Grenzen gesetzt.

- Die erste Wolke haben wir außerhalb der Bühne platziert. Sie besteht aus mehreren weißen Kreisen mit schwarzem Rand. Die schwarzen Linien wurden teilweise wegradiiert und wieder weiß gefüllt, um die Figur der Wolke zu definieren. Wir haben die Wolke dann in ein Movieclip-Symbol konvertiert und sie mit einem Bewegungstween versehen. Der Instanzname ist `wolke`. Aber hierzu später mehr.

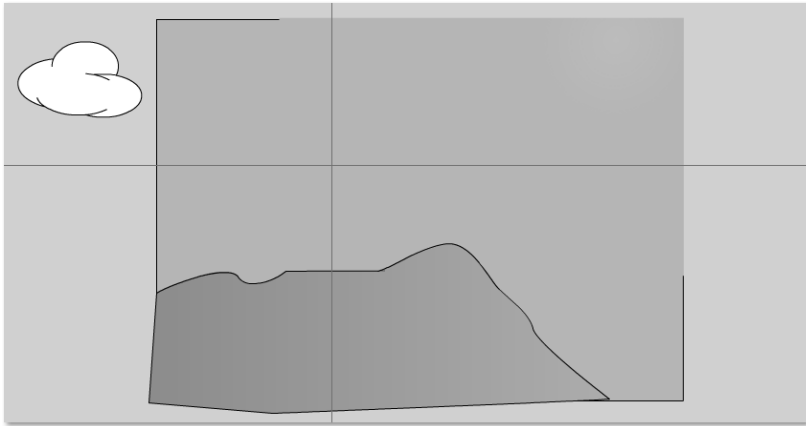


Abbildung 12.7 Die Wolke befindet sich außerhalb der Bühne.

- Die Mühle wurde ebenfalls mit einem Pfad gemalt und mit Beige gefüllt (RGB: (255, 204, 153), hexa: #FFCC99). Die obere schwarze Kante wurde entfernt.
- Ein Flügel besteht aus einem etwas verzerrten Rechteck mit linearem Verlauf von Weiß nach Schwarz und einer Linie. Durch Drehung und Anpassung der Füllung können Sie aus einem Flügel ganz einfach vier erzeugen.
Der Flügel wird dann in ein Movieclip-Symbol konvertiert und erhält den Instanznamen `fluegel`.
- Nun kommt noch ein Punkt mit einem radialen Verlauf von Blau nach Schwarz auf die zweite Ebene **Mühle**. Er bildet den Mittelpunkt der Flügel.

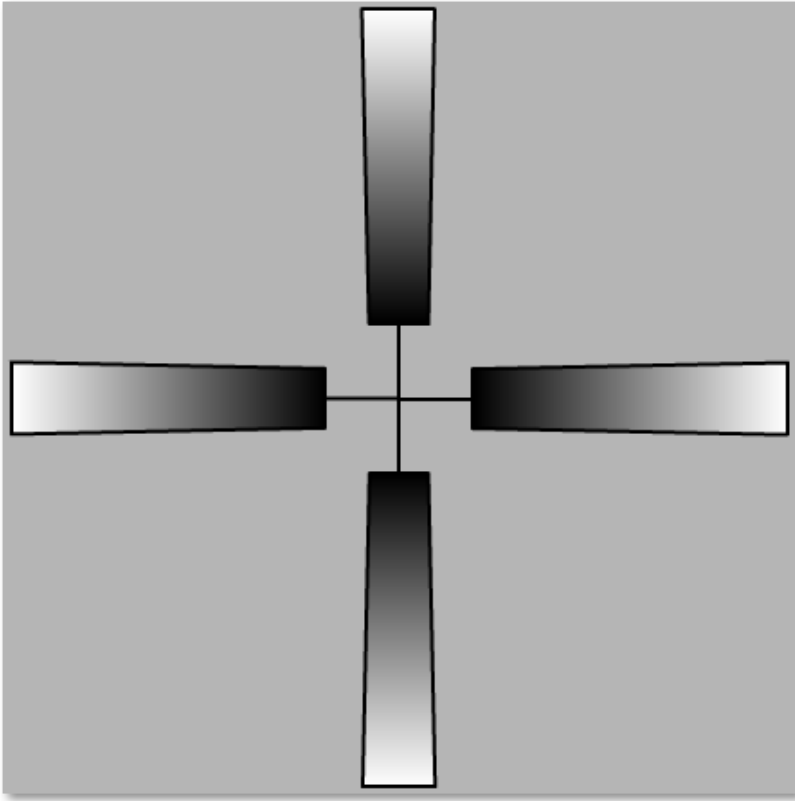


Abbildung 12.8 Die vier Flügel

Sie sollten hier mit Hilfslinien arbeiten. Sie erleichtern die Orientierung.

- ▶ Die zweite Wolke wird ähnlich erstellt wie die erste, nämlich aus weißen Kreisen mit schwarzem Rand. Dann radieren Sie in der Mitte die Konturen aus und füllen die leeren Stellen mit Weiß. Die zweite Wolke wird ebenfalls außerhalb des Films, aber auf der rechten Seite platziert. Die zweite Wolke ist ebenfalls ein Movieclip-Symbol, das die Animation enthält. Es hat den Instanznamen `regenwolke`.
- ▶ Als Letztes gibt es noch eine Skriptebene. Sie enthält im Moment nur den Befehl `gotoAndPlay(1)`, damit der Film in einer Endlosschleife abläuft.



Aus welchen Elementen der Flash-Film besteht, ist Ihnen nun klar. Jetzt geht wir daran, die verschiedenen Movieclips zu untersuchen. Wenn Sie aufgepasst haben – wovon wir ausgehen ☺ –, haben Sie gesehen, dass wir unter dem Hauptfilm drei Movieclips verwenden:

- einen für die erste Wolke. Der Instanzname ist `wolke`. Das zugehörige Symbol heißt `Wolke1`. Es enthält die Bewegung der Wolke.
- Der zweite Movieclip enthält die Flügel. Sein Instanzname ist `fluegel`. Das zugehörige Symbol heißt `Flügel1`.
- Der dritte Movieclip enthält die zweite Wolke. Seine Instanz heißt `regenwolke`. Das zugehörige Symbol ist `Wolke2`.



Den Symbolnamen sehen Sie im Eigenschafteninspektor, wenn Sie die Instanz anklicken (siehe Abbildung 12.8).

Diese drei Movieclips bilden aber nur die oberste Ebene der Movieclips. Die beiden Wolken haben jeweils noch untergeordnete Movieclips. Die erste Wolke enthält eine Instanz `farbe`. Sie stammt von dem Symbol `Wolke1_Farbe`. Dieser Movieclip enthält die Bilder und den Code, der die Farbe der Wolke zufallsgesteuert ändert.

Die zweite Wolke hat eine Instanz `mitregen`. Das dahinter liegende Symbol ist `Wolke_mit_Regen`. Es enthält die Wolke mit dem Regen. Der Regen enthält allerdings auch noch einmal eine Animation, nämlich die Bewegung von oben nach unten. Deswegen gibt es für ihn einen eigenen Movieclip. Der Instanzname dazu ist `regen`, das zugehörige Symbol `Regen`.

In Abbildung 12.9 sehen Sie die Hierarchie der Movieclips in unserem Beispiel, aufgezeigt mit den Instanznamen. In dieser Hierarchie werden wir in den folgenden Abschnitten noch öfter herumspielen.

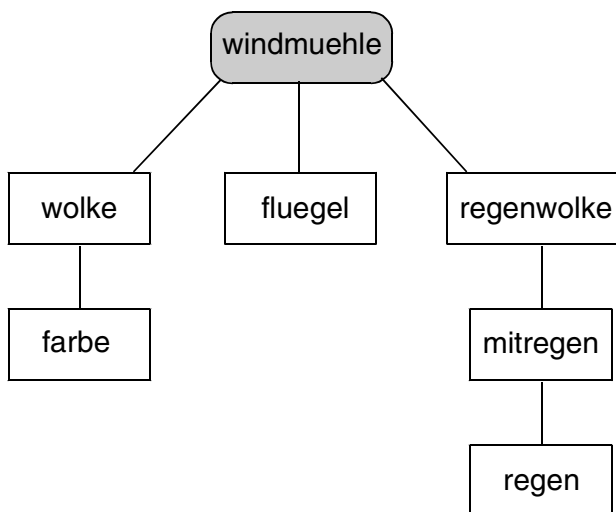


Abbildung 12.9 Die Hierarchie der Movieclips von `windmuehle.fl`

Natürlich könnte man das obige Beispiel auch anders realisieren. Die Methode mit Movieclips erlaubt allerdings eine einfache Verschachtelung und nicht zuletzt einen einfachen Zugriff mit ActionScript.



12.1.2 Zugriff

Nehmen wir an, Sie haben mehrere, ineinander verschachtelte Movieclips. Dann ist natürlich die Frage zu klären, wie Sie mit ActionScript von einem Movieclip auf einen anderen zugreifen. Wie Sie in den vorangegangenen Kapiteln gelesen haben, sind Variablen und Funktionen – wenn sie nicht gerade global definiert wurden – immer nur für eine Zeitleiste gültig.

Es gibt zwei Möglichkeiten, auf einen Movieclip zuzugreifen: relativ oder absolut. Relativ bedeutet relativ, gesehen von der aktuellen Position aus. Absolut bedeutet mit einem festgeschriebenen Pfad vom Ursprung aus.

Wenn Sie schon viel im Web gearbeitet haben, kennen Sie sicher das Prinzip von relativen und absoluten Links. Ein absoluter Link enthält immer den Domainnamen und den kompletten Pfad (z.B. **<http://www.galileocomputing.de/katalog/buecher/gp/themaID-7>**). Ein relativer Link dagegen wechselt beispielsweise nur vom aktuellen in einen übergeordneten Ordner (**[../index.htm](#)**).



Wie sieht das in der Praxis aus? Rufen Sie sich bitte noch einmal die hierarchische Struktur von Movieclips in Erinnerung (Visualisierung siehe Abbildung 12.1). Nehmen wir nun an, Sie befinden sich in der Hauptzeitleiste. Um auf einen Movieclip zuzugreifen, benötigen Sie nur seinen Instanznamen:

```
Clip1
```

Wollen Sie auf einen Movieclip zugreifen, der noch eine Ebene tiefer liegt, geben Sie die Movieclip-Hierarchie durch Punkte getrennt an.

```
Clip1.Clip11
```

greift beispielsweise auf den Movieclip mit dem Instanznamen `Clip11` zu, der hierarchisch unter `Clip1` angeordnet ist.

Was aber, wenn Sie sich nicht in der Hauptzeitleiste befinden oder absolut verlinken wollen? Dazu benötigen Sie Möglichkeiten, den aktuellen, einen übergeordneten oder den Haupt-Movieclip aufzurufen. Drei Schlüsselworte helfen hier weiter:

- `this` referenziert immer auf den aktuellen Movieclip. Innerhalb einer Movieclip-Instanz benötigen Sie das Schlüsselwort `allerdings` nicht. Hier reicht es, eine Eigenschaft oder Methode direkt aufzurufen.
- `_root` liefert den Haupt-Movieclip.
- `_parent` liefert den um eine Hierarchieebene höher liegenden Movieclip.

In Abbildung 12.10 finden Sie illustriert, wie `_parent` und `_root` dazu verwendet werden können, auf höher gelegene Ebenen zu verweisen.



Verwechseln Sie die Hierarchieebenen der Movieclips nicht mit den Ebenen in der Zeitleiste. Eine zweite Verwechslungsgefahr besteht mit den verschiedenen Ebenen (Levels, Stufen), die entstehen, wenn mehrere Filme geladen werden. Noch einmal oberlehrerhaft: Die Hierarchie der Movieclips ist von diesen beiden anderen Ebenen-Arten vollkommen unabhängig.

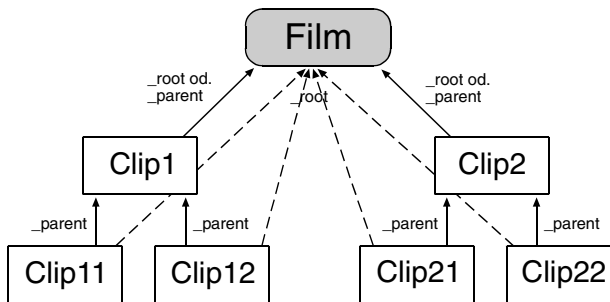


Abbildung 12.10 Zugriff auf höher gelegene Movieclips

Das Rüstzeug ist vorhanden, lassen Sie uns nun ein paar Aufgaben lösen.

- Ausgangspunkt ist `Clip11`, gewechselt werden soll zu `Clip12`.

Relativ:

```
_parent.Clip12
```

Absolut:

```
_root.Clip1.Clip12
```

- Ausgangspunkt ist wieder Clip11, aufgerufen werden soll Clip22.

Relativ:

```
_parent._parent.Clip2.Clip22
```

Absolut:

```
_root.Clip2.Clip22
```

Sie merken schon, absolute Links arbeiten immer mit `_root`, also einem Verweis auf den Haupt-Movieclip. Das kennen Sie auch von anderen absoluten Links, dort entspricht der Haupt-Movieclip einem Laufwerk (c:) oder einer Domain (www.galileo-press.de).

Absolute Referenzierung auf Movieclips sollten Sie eher selten wählen, da bei einer Strukturänderung die Links nicht mehr stimmen. Das passiert schon nicht? Leider doch, in der Praxis kann es beispielsweise sein, dass Sie einen Film erstellen und diesen dann in einen anderen laden, und schon funktioniert die absolute Verlinkung nicht mehr.

In Flash 4 folgte der Zugriff auf Movieclips noch einem anderen Adressierungsschema, nämlich einer Notation mit Schrägstrichen (/). Diese Notation wird noch von manchen Eigenschaften verwendet. Eine Umwandlung ist mit der Funktion `eval` möglich.



Am Beispiel erklärt

Bisher erfolgte das Linken an eher unspezifischen Beispielen. Wir wollen jetzt die Datei **windmuehle fla** verwenden, um Ihnen den Zugriff auf Movieclip-Instanzen in der Praxis zu zeigen.

Ziel soll es sein, den Regen etwas sanfter (durchsichtiger) zu machen.

1. Um das Ziel zu erreichen, verwenden wir die Eigenschaft `_alpha`, die für jeden Movieclip zur Verfügung steht. Eine Übersicht aller Eigenschaften von Movieclips finden Sie im Abschnitt »Eigenschaften« in Tabelle 12.1.
2. Entsperren Sie die Ebene **Skript**.
3. Sie müssen nun den Alphawert des Movieclips mit dem Regen herunterregeln. Würden Sie einen übergeordneten Movieclip ändern, wäre auch die Wolke durchsichtiger. Fügen Sie im **Aktionen**-Bedienfeld also folgenden Code hinzu:

```
_root.regenwolke.mitregen.regen._alpha=20;
```

Alternativ können Sie auch relativ verlinken:

```
regenwolke.mitregen.regen._alpha=20;
```



Sollten Sie Schwierigkeiten haben, die Adressierung nachzuvollziehen, betrachten Sie noch einmal Abbildung 12.9 mit der Hierarchie von **windmuehle fla**. Wir empfehlen Ihnen, auch für eigene Filme die Hierarchie immer im Kopf zu haben. Allerdings finden Sie im Zieleditor (siehe hierzu den folgenden Abschnitt) auch eine gute Hilfestellung.

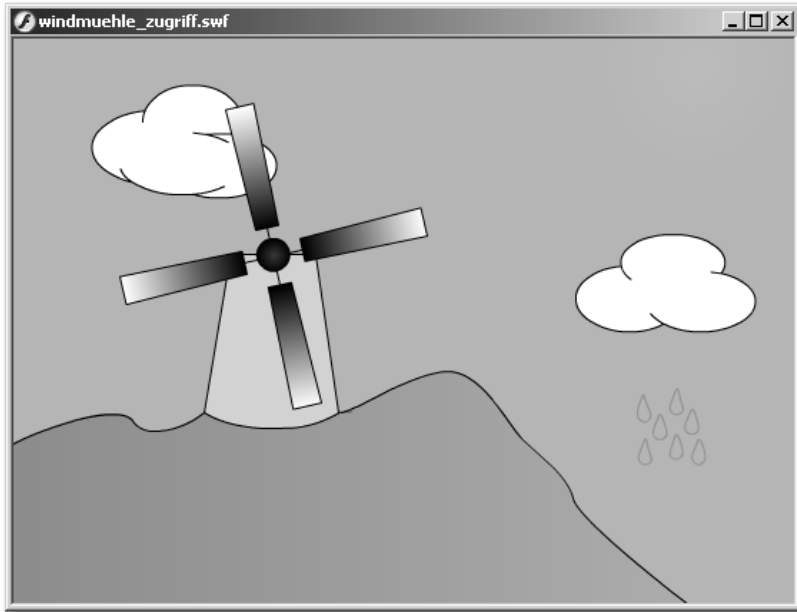


Abbildung 12.11 Der Regen ist wesentlich schwächer zu sehen.

Das war ja schon ganz erfolgreich (siehe Abbildung 12.11). Nun wollen wir noch die Windmühle immer dann anhalten, wenn die erste Wolke ihre Farbe wechselt.

Um das zu erreichen, müssen wir in den Movieclip mit dem Farbwechsel gehen. Das Symbol heißt `Wolke1_Farbe`. Dort finden Sie Folgendes vor:

- Sie sehen in der Zeitleiste des Symbols drei Schlüsselbilder. Das erste Schlüsselbild enthält ActionScript-Code, der ein Zeitintervall festlegt.

```
stop();  
var z;  
var int;
```

```

z = Math.random()*10;
z = (Math.round(z) * 1000)+1;
int = setInterval(springen, z);
function springen() {
    gotoAndPlay(2);
}

```

Der Code erstellt zuerst eine Zufallszahl, die so umgewandelt wird, dass sie zehn Werte zwischen 0001 und 10001 ergibt. Diese zehn Werte werden als Zufallszahl für eine `setInterval`-Funktion verwendet. Wenn das Intervall zu Ende ist, wird mit der Funktion `springen` zum zweiten Bild des Movieclips gewechselt.

- Das zweite Bild enthält die Wolke in einer anderen Farbe. So wird also der Farbwechsel in zufallsbestimmten Zeitintervallen gesteuert. Im zweiten Bild wird außerdem das Intervall wieder gelöscht, damit es später wieder neu belegt werden kann.

```
clearInterval(int);
```

- Das letzte Schlüsselbild enthält die Funktion `gotoAndPlay(1)`, mit der wieder auf das erste Bild zurückgesprungen wird. Die Wolke ist dann wieder weiß und das nächste Intervall wird wieder mit einer Zufallszahl gesteuert.

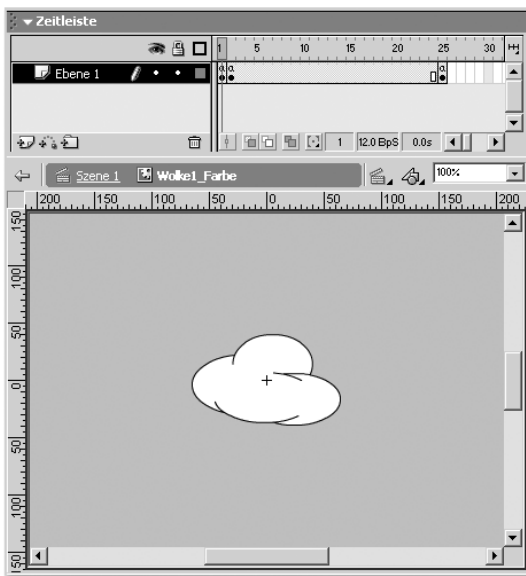


Abbildung 12.12 Das Symbol und seine Zeitleiste mit drei Schlüsselbildern

In diesen Code wollen wir nun die Steuerungsanweisung einfügen, dass die Mühlenflügel anhalten sollen, wenn die Farbe wechselt.

1. Wechseln Sie dazu zuerst auf das zweite Bild, denn hier findet der Farbwechsel statt. Mit der Movieclip-Funktion `Stop` halten Sie die Flügel an. Der Abspielkopf im Movieclip `fluegel` bewegt sich nicht mehr weiter.

```
_root.fluegel.Stop();
```

Wollen Sie den Movieclip relativ referenzieren, verwenden Sie folgenden Code:

```
_parent._parent.fluegel.Stop();
```

2. Wenn Sie den Film testen (`[Strg] + [↵]`), werden die Flügel angehalten, es wird aber nicht weitergespielt, wenn die Wolke wieder weiß wird. Hierfür benötigen Sie noch eine Zeile Code im ersten Schlüsselbild des Symbols.

```
_root.fluegel.Play();
```

erlaubt dem Movieclip `fluegel` wieder, den Abspielkopf weiterzubewegen. Das Ganze funktioniert natürlich auch relativ:

```
_parent._parent.fluegel.Play();
```



Sie finden die fertige Datei auf der CD-ROM zum Buch unter dem Namen **windmuehle_zugriff fla**. Experimentieren Sie noch ein bisschen mit verschiedenen Möglichkeiten. Beispielsweise ließe sich auch die Regenwolke anhalten, wenn gerade Regen fällt. Mehr Möglichkeiten zum Einstellen und Ändern für Movieclips finden Sie in den Abschnitten »Eigenschaften« und »Methoden«.

Zieleeditor

Für alle, denen das Eingeben der Pfade von Hand zu kompliziert ist, hält das **Aktionen**-Bedienfeld ein eigenes Dialogfeld **Zielpfad einfügen** bereit (auch **Zieleeditor**). Sie finden das Symbol in der Mitte des **Aktionen**-Bedienfelds.

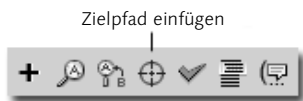


Abbildung 12.13 Das Dialogfenster **Zielpfad einfügen**

Wenn wir das Dialogfeld aus dem ersten Schlüsselbild des Haupt-Movieclips von **windmuehle.fla** aufrufen, erhalten wir alle Movieclips, die unter dem Haupt-Movieclip liegen.

Mit +-Symbolen wird angezeigt, dass ein Movieclip noch untergeordnete Movieclips hat. Sie können diese ausklappen.

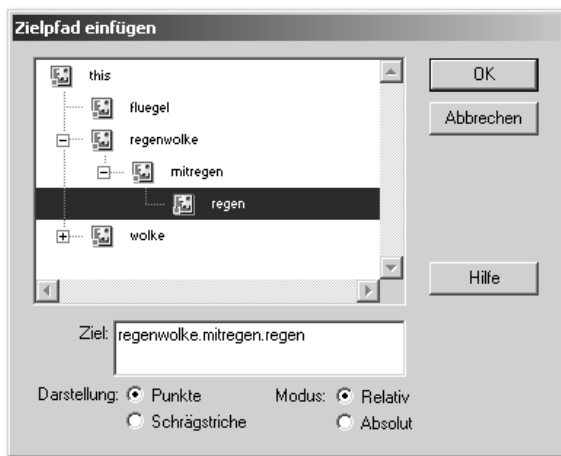


Abbildung 12.14 Relativer Pfad

Im Feld **Ziel** sehen Sie den Pfad des momentan aktivierten Movieclips. Der **Modus** bestimmt, ob der Pfad relativ oder absolut angegeben wird.

In Abbildung 12.15 sehen Sie den absoluten Pfad für den Movieclip **regen**.

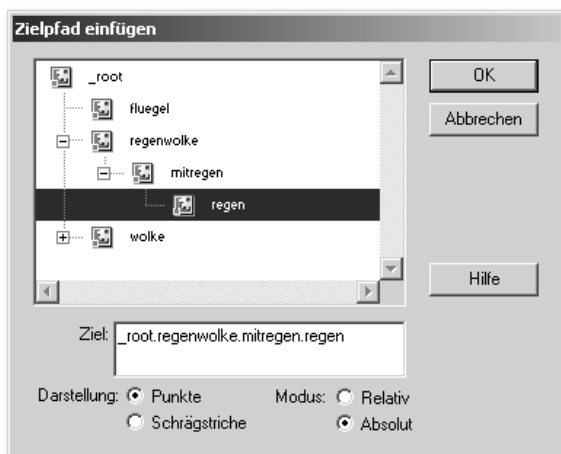


Abbildung 12.15 Absoluter Pfad

Die **Darstellung** erlaubt Ihnen, die alte Darstellung aus Flash 4 mit Schrägstrichen zu übernehmen. Das ist beispielsweise nützlich, wenn Sie Flash 4-kompatible Pfade anlegen möchten.



Abbildung 12.16 Darstellung mit Schrägstrichen



Wenn Sie sich in einem hierarchisch niedriger stehenden Movieclip befinden, können Sie im Dialogfeld **Zielpfad einfügen** mit einer relativen Verlinkung nicht auf darüber liegende Movieclips zugreifen. Dies funktioniert nur mit absoluter Verlinkung. Der Zieleditor kennt also `_parent` nicht. In Abbildung 12.17 sehen Sie dies für die Movieclip-Instanz `farbe`.



Abbildung 12.17 Die übergeordneten Movieclips stehen nicht zur Verfügung.

12.1.3 Mit ActionScript erstellen

Die Instanz eines MovieClip-Symbols von Hand zu erstellen ist schön und gut, überlegen Sie aber, welche Möglichkeiten Sie haben, wenn Sie Instanzen mithilfe von einfachen Programmierbefehlen erzeugen und

verändern können. Es gibt in ActionScript drei wichtige Methoden, Instanzen zu erzeugen:

- ▶ Sie können mit der Methode `duplicateMovieClip` eine bestehende Instanz duplizieren. Dann lässt sich auch das Duplikat erneut duplizieren.
- ▶ `createEmptyMovieClip` erstellt einen neuen, leeren Movieclip.
- ▶ Mit `attachMovie` lässt sich eine Instanz aus einem Symbol erstellen, allerdings nur, wenn das Symbol auch in die swf-Datei exportiert wurde.

Instanz duplizieren

Wenn Sie bestehende Instanzen haben, können Sie diese einfach mit `duplicateMovieClip` kopieren und unter einem anderen Instanznamen einfügen. Die duplizierte Instanz wird dabei an der gleichen Position eingefügt wie das Original.

Wir wollen Ihnen die Funktionsweise kurz an einem Beispiel erläutern. Öffnen Sie dazu die Datei **windmuehle.fla**. In dieser Datei gibt es eine Menge netter Sachen zu duplizieren. Zuerst wollen wir uns einmal an dem Flügel der Windmühle versuchen.

1. Aktivieren Sie das erste Schlüsselbild der Ebene **Skript** in der Hauptzeitleiste.
2. Duplizieren Sie den Movieclip `fluegel` mit `duplicateMovieClip`. Bei `duplicateMovieClip` als Methode eines Movieclips müssen Sie zwei Parameter angeben, den Namen der neuen Instanz und die Position (Tiefe) des Movieclips im Ebenenstapel. Dieses Konzept bedarf einer Erklärung, die Sie gleich im Anschluss an dieses Beispiel finden werden.

```
fluegel.duplicateMovieClip("fluegel_kopie", 0);
```

Die Adressierung des Movieclips erfolgt, wie im vorangegangenen Abschnitt, »Zugriff«, erklärt wurde, entweder relativ oder absolut.



3. Zum Schluss drehen wir die neue Instanz noch mit der Movieclip-Eigenschaft `_rotation` um 45°. Beachtenswert ist, dass diese Eigenschaft im Gegensatz zu einigen mathematischen Funktionen kein Bogenmaß, sondern die Gradzahl als Wert übernimmt.

```
fluegel_kopie._rotation = 45;
```

Nun laufen also zwei Flügel der Mühle um 45° versetzt. Sieht lustig aus, oder (siehe Abbildung 12.18)? Und stellen Sie sich vor, was Sie in Flash alles

hätten tun müssen: eine neue Instanz des Symbols auf die Bühne ziehen, ausrichten und drehen. In ActionScript waren es nur zwei Zeilen Code.

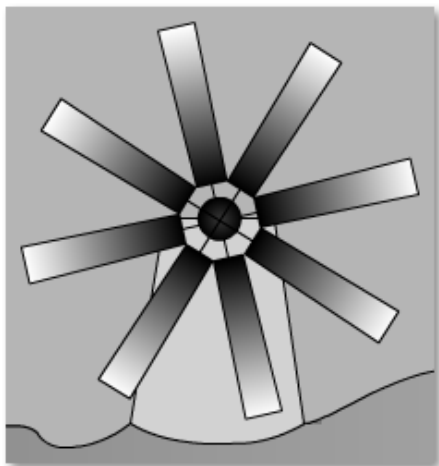


Abbildung 12.18 Die Mühle mit zwei Flügeln

Wir haben Ihnen die Erklärung des Konzepts der Tiefe (Position) einer mit ActionScript eingefügten Instanz im Ebenenstapel versprochen. Wenn Sie den Film testen (**Strg** + **↵**) und der zweiten Wolke beim Fliegen zusehen, was fällt Ihnen da auf?

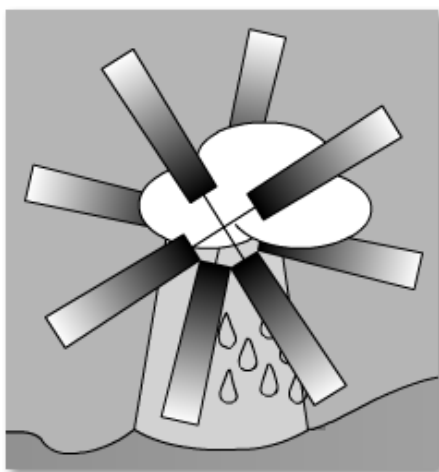


Abbildung 12.19 Der zweite Flügel liegt über der Wolke.

Der zweite Flügel liegt über der Wolke (siehe Abbildung 12.19). Das lässt sofort einen Schluss zu: Die neu duplizierte Instanz kommt nicht auf dieselbe Ebene in der Zeitleiste wie das Original. Es geht noch weiter: Eine duplizierte Ebene kommt nie auf eine Ebene der Zeitleiste, sondern liegt immer darüber.

Die Positionsangabe, bzw. die Tiefe im Ebenenstapel, bezieht sich auf den Ebenenstapel aller per ActionScript eingefügten Ebenen. Dort ist Position 0 die unterste Ebene. Sie liegt aber immer noch über den Ebenen der Zeitleiste. Abbildung 12.20 visualisiert dieses Konzept.

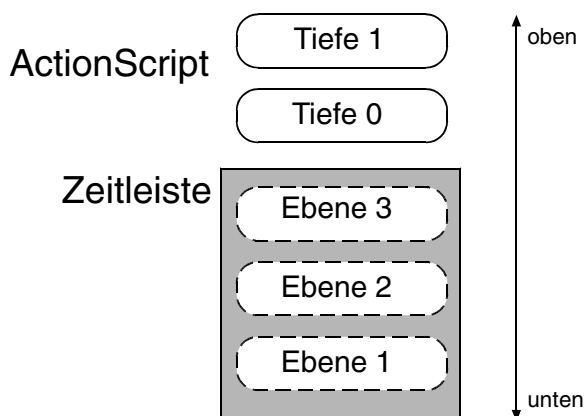


Abbildung 12.20 Das Ebenenkonzept

Per ActionScript erzeugte Ebenen haben noch eine Besonderheit: Es lassen sich nicht mehrere Movieclip-Instanzen auf einer Ebene (Tiefe) einfügen.

In Flash 4 gab es das Konzept der bestimmbaren Tiefe noch nicht. Hier wurden duplizierte Instanzen einfach fest übereinander gelegt. Die Tiefe ließ sich nicht manipulieren.



Beim Duplizieren werden von der Original-Instanz alle Einstellungen, auch per ActionScript vorgenommene, mit übernommen. Auch Ereignisse in den Movieclips werden mit dupliziert. Das Duplikat lässt sich dann allerdings unabhängig vom Original weiterbehandeln, wie wir an der Rotation im Beispiel schon gesehen haben.

Aber nicht nur das Duplikat ist vom Original unabhängig. Auch nachträgliche Änderungen im Original wirken sich nicht auf das Duplikat aus.

Wenn wir also beispielsweise den Original-Flügel etwas vergrößern, **nachdem** wir das Duplikat erzeugt haben, wird das Duplikat nicht ange-tastet.

```
fluegel.duplicateMovieClip("fluegel_kopie", 0);  
fluegel_kopie._rotation = 45;  
fluegel._xscale = 120;  
fluegel._yscale = 120;
```

Im obigen Beispiel haben wir das Original `fluegel` dupliziert, danach das Duplikat gedreht und zum Schluss das Original horizontal (`_xscale`) und vertikal (`_yscale`) auf 120% vergrößert. Die beiden Änderungen der Movieclip-Instanzen geschehen unabhängig voneinander. Zwischen Original und Duplikat besteht also nach der Erzeugung des Duplikats keine Verbindung mehr.

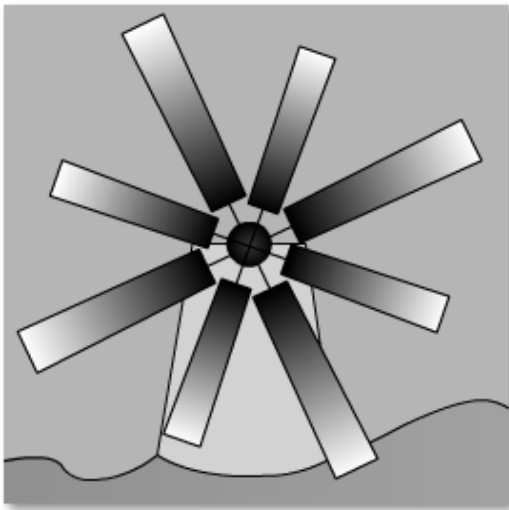


Abbildung 12.21 Nur das Original wurde skaliert.

Nun wollen wir aber noch einmal unser Beispiel aufgreifen und nach dem Duplizieren des Flügels noch die erste Wolke ein zweites Mal einblenden.

1. Auch dies lässt sich im ersten Schlüsselbild des Hauptfilms am besten erledigen. Aber jetzt verwenden wir eine andere Schreibweise von `duplicateMovieClip`, nämlich die als globale Funktion. In dieser Form wird kein Movieclip-Objekt vorangestellt, sondern es werden Name und Pfad des Originals als erster Parameter übergeben. Der zweite Parameter ist der Name und Pfad der neuen Instanz. Sie kann jedem

beliebigen Movieclip hierarchisch untergeordnet werden. Wir ordnen sie der Hauptbühne unter. Als Tiefe geben wir 1 an, da auf 0 schon der Flügel liegt.

```
duplicateMovieClip("wolke", "wolke_kopie", 1);
```

2. Nun versetzen wir die Wolke noch ein wenig, indem wir neue Koordinaten angeben. Die Koordinaten werden vom Koordinatensystem des übergeordneten Movieclips abgeleitet. In diesem Fall ist das die Bühne. Unsere duplizierte Wolke startet also bei den Koordinaten $x = 0$ und $y = 100$, gemessen vom Ursprung in der linken oberen Ecke der Bühne.

```
wolke_kopie._x = 0;  
wolke_kopie._y = 100;
```

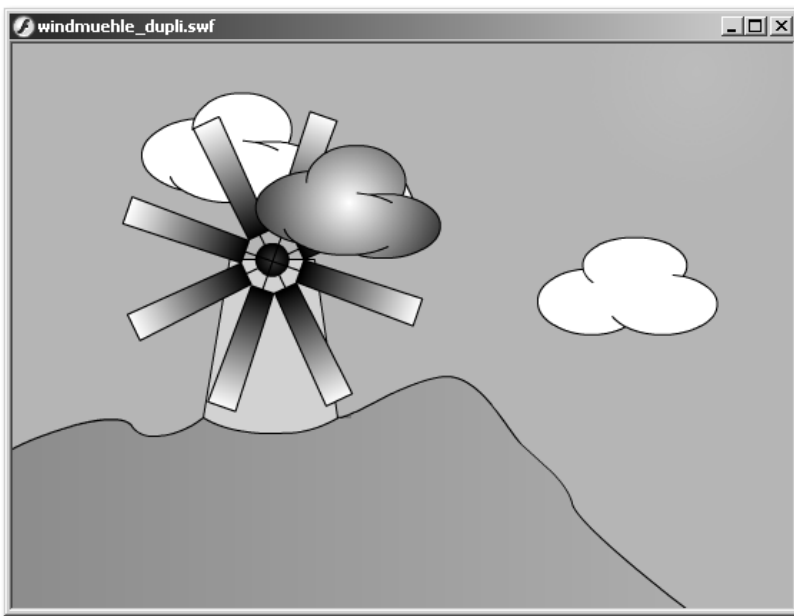


Abbildung 12.22 Das fertige Beispiel

Das fertige Beispiel finden Sie auf der CD-ROM unter dem Namen **windmuehle_dupli fla**.

Noch ein Wort zur Arbeitstechnik: Um mit `duplicateMovieClip` effizient arbeiten zu können, benötigt man bereits bestehende Instanzen von Elementen. Von vielen Elementen soll es aber zumindest zu Anfang oder unabhängig von ActionScript gar keine Instanzen auf der Bühne geben. Die Lösung für dieses Problem ist, Elemente außen um die Bühne herum

anzuordnen. Diese können Sie dann jederzeit duplizieren und zum Einsatz bringen. Diese Technik wird nicht mehr so oft verwendet, seit mit der Methode auch aus Symbolen Instanzen erzeugt werden können (siehe den Abschnitt »Aus Symbol wird Instanz«).

Leere Movieclips erzeugen

In Flash MX kommt eine neue Funktion hinzu: `createEmptyMovieclip()`. Sie erzeugt, wie der Name unschwer erkennen lässt, einen leeren Movieclip. In diesen Movieclip können Sie dann mit den Zeichnungsmethoden neue Elemente einfügen.

Die Zeichnungsmethoden finden Sie im **Aktionen**-Bedienfeld (siehe Abbildung 12.23) unter **Objekte • Film • Movieclip • Zeichnungsmethoden**. Mit ihnen können Sie einfache Formen wie Linien und Kurven zeichnen und gezeichnete Elemente mit einer Farbe oder einem Verlauf füllen.

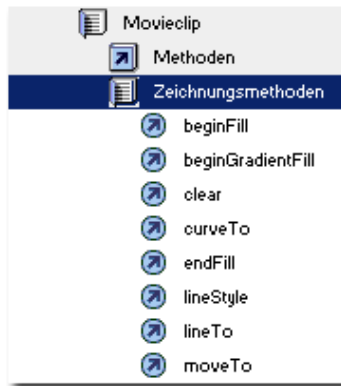


Abbildung 12.23 Die Zeichnungsmethoden im **Aktionen**-Bedienfeld

Wir wollen Ihnen an einem Beispiel zeigen, was Sie mit `createEmptyMovieclip()` machen können. Ausgangspunkt ist wieder **windmuehle fla**. Ziel ist es, nachdem der Flügel der Mühle dupliziert wurde, über die Mitte der beiden Flügel noch einmal ein Rechteck mit einem Farbverlauf von Blau nach Schwarz zu legen. Dieses Rechteck ist dann sozusagen die Halterung des Flügels.

1. Der gesamte Code wandert in das erste Schlüsselbild der Hauptzeitleiste in die Ebene **Skript**. Wir erwähnen den Ort, an dem sich der Code befindet, immer wieder, weil der Startpunkt für die Pfade der Movieclips von Bedeutung ist.

2. Das Duplizieren des Flügels geht mit `duplicateMovieClip()` ganz einfach von der Hand, vor allem, wenn Sie den vorhergehenden Abschnitt gelesen haben.

```
fluegel.duplicateMovieClip("fluegel_kopie", 0);
```

3. Anschließend wird der duplizierte Flügel um 45° gedreht. Ein positiver Wert dreht immer im Uhrzeigersinn.

```
fluegel_kopie._rotation = 45;
```

4. Nun müssen wir einen leeren Movieclip erzeugen. Dazu rufen wir `createEmptyMovieClip()` auf. Mit `this` wird der Haupt-Movieclip adressiert, da wir uns dort befinden. Der leere Movieclip hat den Instanznamen `quadrat` und wird an den Haupt-Movieclip angehängt. Seine Tiefe im Ebenenstapel beträgt 1, er liegt also höher als der duplizierte Flügel.

```
this.createEmptyMovieClip("quadrat", 1);
```

5. Zuletzt geht es ans Zeichnen. Sie lernen hier mehrere neue Dinge kennen. Zum einen die `with`-Anweisung: Sie erlaubt Ihnen, auf alle Eigenschaften und Methoden eines Objekts zuzugreifen, ohne das Objekt dauernd anzugeben. Sie schreiben das Objekt einmal in runde Klammern und verwenden dann nur noch Methoden und Eigenschaften dieses Objekts. Mehr dazu erfahren Sie im Abschnitt »with«.

```
with (quadrat) {  
    //Methoden und Anweisungen für den Movieclip quadrat  
}
```

6. Als Nächstes lernen Sie, wie Sie zeichnen können. Der Verlauf wird mit der Zeichnungsmethode `beginGradientFill()` erstellt. Diese Methode benötigt mehrere Parameter, die wir vorher als Arrays definiert haben.

```
var farben = [0x0000FF, 0x000000];  
var alpha = [100, 100];  
var farbauftrag = [0, 0xFF];  
var matrix = { matrixType:"box", x:170, y:140, w:28, h:28, r:0 };  
beginGradientFill("radial", farben, alpha, farbauftrag, matrix);
```

Folgende Parameter können Sie festlegen:

- Der erste Parameter bestimmt die Art des Verlaufs als Zeichenkette: entweder `radial` (kreisförmig) oder `linear`.
- Der zweite Parameter legt die Farben des Verlaufs fest. Wir haben Blau (hexa: `#0000FF` entspricht `0x0000FF`) und Schwarz (hexa: `0x000000`) gewählt.
- Der dritte Parameter legt für alle Farben die Transparenz, also die Alphawerte, fest. Es müssen genau so viele Transparenzwerte angegeben werden, wie vorher Farben festgelegt wurden.
- Der vierte Parameter gibt an, wo die jeweilige Farbe einen Farbauftrag von 100 % hat.
- Die Matrix ist der fünfte Parameter. Sie gibt die Form und eventuelle Transformationen des Verlaufs an. Wenn Sie für `matrixType` die – einzig mögliche – Option `box` verwenden, wird ein Rechteck definiert mit den Startkoordinaten (links oben) `x` und `y`, der Breite `w` und der Höhe `h`. `r` gibt eine eventuelle Krümmung des Rechtecks im Bogenmaß an.



Die Matrix kann auch ohne `matrixType` auskommen. Dann besteht Sie aus neun Werten, die eine 3×3 Felder große Matrix bilden. Mehr dazu finden Sie in der Online-Hilfe, wenn Sie nach `matrixType` suchen.

7. Anschließend müssen wir das Quadrat zeichnen. Dazu bewegen wir zuerst mit der Methode `moveto(x, y)` den Zeichenstift auf die Startposition (hier die rechte obere Ecke). Dann zeichnen wir nach und nach mit `lineto(x, y)` die vier Punkte.

```
moveto(170, 140);
lineto(170, 168);
lineto(198, 168);
lineto(198, 140);
lineto(170, 140);
endFill();
```

Im Folgenden finden Sie noch einmal den kompletten Code des Beispiels zur Übersicht abgedruckt:

```
fluegel.duplicateMovieClip("fluegel_kopie", 0);
fluegel_kopie._rotation = 45;

this.createEmptyMovieClip("quadrat", 1);
with (quadrat) {
```

```

farben = [0x0000FF, 0x000000];
alpha = [100, 100];
farbauftrag = [0, 0xFF];
matrix = { matrixType:"box", x:170, y:140, w:28, h:28, r:0 };
beginGradientFill("radial", farben, alpha, farbauftrag, matrix);
moveto(170, 140);
lineto(170, 168);
lineto(198, 168);
lineto(198, 140);
lineto(170, 140);
endFill();
}

```

In Abbildung 12.24 sehen Sie das Ergebnis unserer schweißtreibenden Arbeit. Wenn es Ihnen einmal zu mühselig wird, denken Sie nur an die armen Müller, die noch wesentlich schwerer schuften mussten.

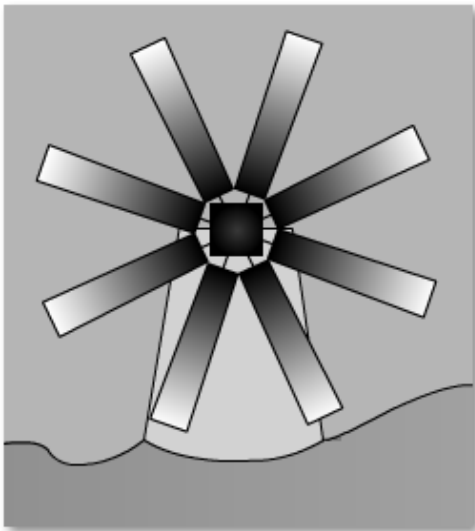


Abbildung 12.24 Die Windmühle mit quadratischer Flügelmitte

Die fertige Datei finden Sie unter dem Namen **windmuehle_empty.fla** auf der CD-ROM zum Buch.

Aus Symbol wird Instanz

`attachMovie` erzeugt aus einem Symbol eine Instanz. Von diesem Symbol muss bisher auch noch keine Instanz existieren. Im **swf**-Film muss das Symbol allerdings vorhanden sein. Welche Konsequenzen diese letzte Bedingung hat, wird klar, wenn man überlegt, wie Flash beim Veröffentlichen mit Symbolen verfährt. Symbole, die von Hand erstellte Instanzen haben, werden mit in die **swf**-Datei übernommen, Symbole ohne eine solche Instanz bleiben dagegen außen vor. Das heißt, dass standardmäßig alle Symbole, die keine von Hand erstellte Instanz haben, nicht exportiert werden, auch wenn Instanzen per ActionScript erzeugt werden.

Das ist natürlich kein Zustand. Deswegen können Sie Symbole beim Veröffentlichen mit exportieren. Diese können dann auch (nur) von ActionScript für Instanzen verwendet werden.

Den Export legen Sie in den **Symboleigenschaften** fest. Bewegen Sie den Mauszeiger auf das gewünschte Symbol und klicken Sie **Eigenschaften** an, schon öffnet sich das Fenster. Es ist unter Umständen noch eingeklappt. Dann hilft das Anklicken von **Erweitert**.

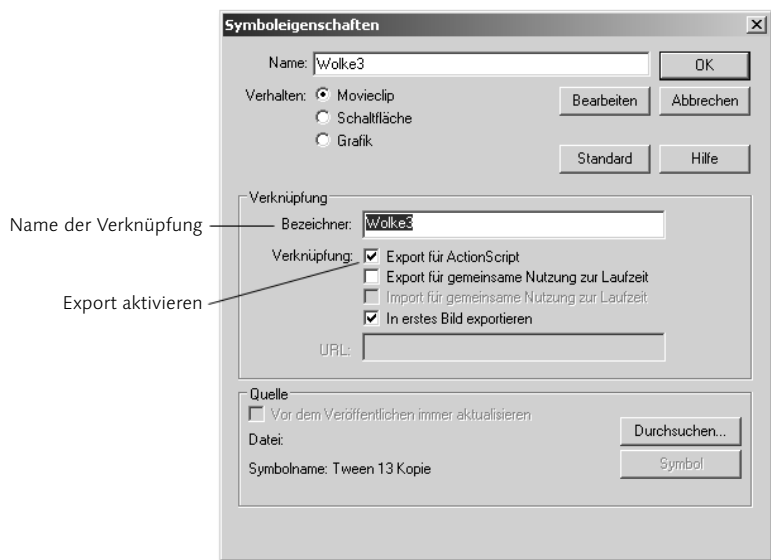


Abbildung 12.25 Das Dialogfenster **Symboleigenschaften**

Folgende Optionen stehen für den Export zur Auswahl:

- ▶ **Export für ActionScript** aktiviert den Export. Für das Movieclip-Symbol wird eine Verknüpfung angelegt und es wird immer mit dem Film exportiert.
- ▶ Sie müssen dann unter **Bezeichner** einen Namen für das exportierte Symbol angeben. Vorgeschlagen wird immer der Symbolname. Das ist in den meisten Fällen in Ordnung. Sie sollten allerdings darauf achten, dass der Name den Namenskonventionen für Variablen (siehe den Abschnitt »Namenskonventionen« in Kapitel 9, ActionScript: Grundlagen) entspricht. Umlaute sind also beispielsweise nicht empfehlenswert.
- ▶ **In erstes Bild exportieren** bestimmt, dass das Symbol sofort im ersten Bild geladen werden soll.

Wir wollen Ihnen das wiederum an einem kleinen Beispiel in **windmuehle fla** zeigen. Wir erstellen einfach eine weitere Wolke und machen sie zu einem Movieclip-Symbol mit dem Namen **Wolke3**.

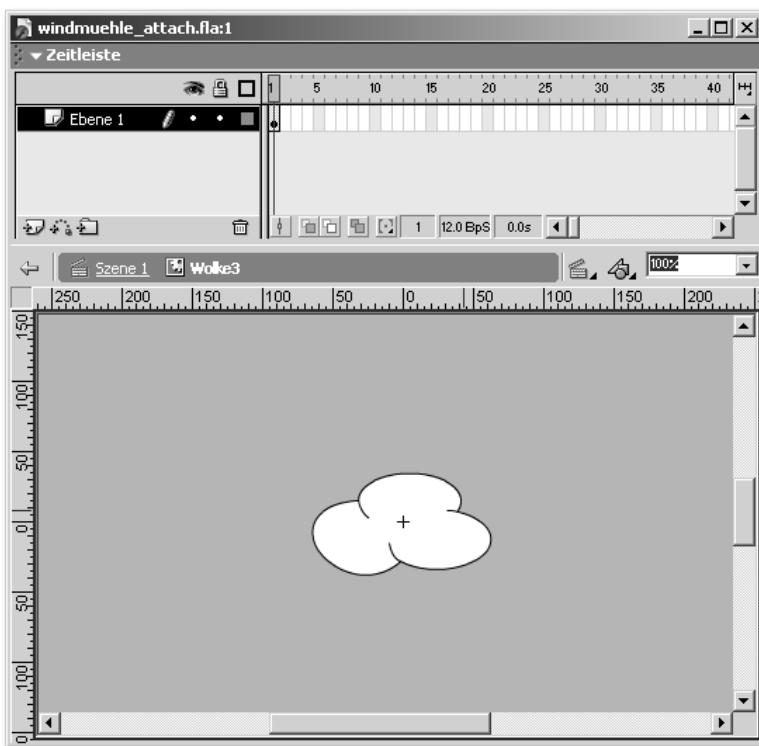


Abbildung 12.26 Ein weiteres Wolken-Symbol

1. Diese Wolke müssen wir im ersten Schritt in der Bibliothek für den Export fertig machen. Öffnen Sie dazu die Bibliothek (**F11**).
2. Dort wählen Sie im Kontextmenü (durch Anklicken mit der rechten Maustaste) den Befehl **Eigenschaften**.
3. In dem Dialogfenster **Symboleigenschaften** aktivieren Sie **Export für ActionScript**.
4. Den Bezeichner belassen Sie bei **Wolke3** (Einstellungen siehe Abbildung 12.26) und bestätigen mit **OK**.

Nun ist das Movieclip-Symbol für den Export bereit und kann per ActionScript und `attachMovie` als Instanz hinzugefügt werden.

5. Wechseln Sie in das erste Bild des Hauptfilms auf die Ebene **Skript**.
6. Mit folgendem Code fügen wir eine Instanz des Symbols `Wolke3` an den aktuellen Film, also den Haupt-Movieclip, an. Die Instanz erhält den Namen `wolke3`. Als Tiefe im Ebenenstapel geben wir `0` an.

```
this.attachMovie("Wolke3", "wolke3", 0);
```

Wenn Sie den Film jetzt testen, sehen Sie, dass die Wolke noch links oben in der Ecke hängt (siehe Abbildung 12.27). Das liegt daran, dass mit `attachMovie` eingefügte Movieclips automatisch in den Ursprung des Movieclips, an das sie angehängt sind, gesetzt werden. In unserem Fall ist das die Bühne, und deren Ursprung liegt links oben. Dabei wird übrigens genau die Mitte des Movieclips auf dem Ursprung platziert, da der Ursprung des Koordinatensystems eines Movieclip-Symbols in der Mitte liegt.

7. Wir müssen also im letzten Schritt die Wolke noch ein wenig verschieben. Dazu dienen die Eigenschaften `_x` und `_y`, die die Position vom Ursprung des übergeordneten Movieclips auf der x-Achse und der y-Achse festlegen (siehe Abbildung 12.28).

```
wolke3._x = 400;
wolke3._y = 80;
```

Die fertige Datei finden Sie auf der CD-ROM unter dem Namen **windmuehle_attach fla**.

Die Wolke in der Ecke

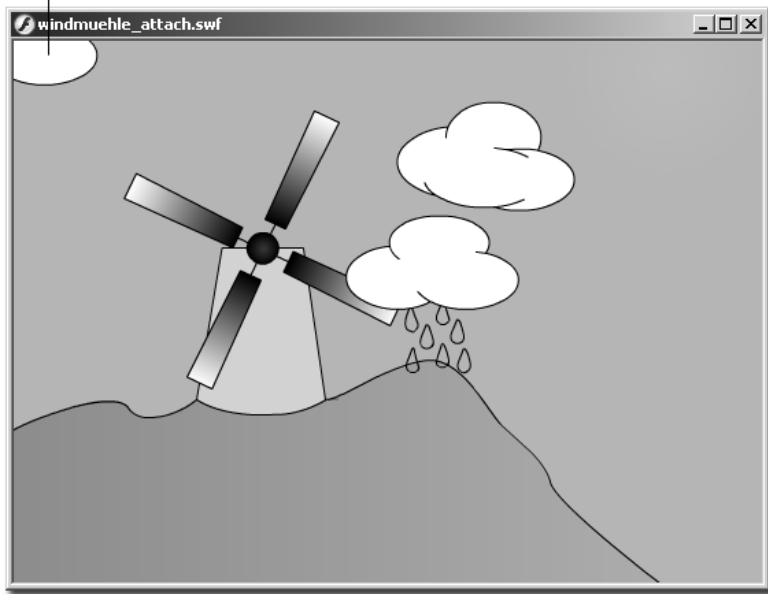


Abbildung 12.27 Die Wolke will noch nicht in den Mittelpunkt.



Abbildung 12.28 Die Wolke hat einen neuen Platz unter der Sonne gefunden.



Sie können von einem exportierten Symbol beliebig viele Instanzen erstellen. Nutzen Sie diese Möglichkeit ausgiebig und verändern Sie die Instanzen mit den Eigenschaften und Methoden für Movieclips. Vergessen Sie aber nicht, für jede Instanz eine neue Tiefe anzugeben.

12.1.4 Löschen

Was man erstellt, muss man auch wieder löschen können. Das gilt natürlich auch für Movieclips. Es gibt für diesen Zweck zwei Methoden:

- ▶ `removeMovieClip` löscht die Instanz eines duplizierten oder aus einem Symbol erstellten Movieclip, nicht aber eine von Hand erstellte Instanz.
- ▶ `unloadMovie` entfernt den Inhalt einer Instanz oder einen ganzen Film. Die Instanz wird durch einen leeren Platzhalter ersetzt.

Um Ihnen die Anwendung der beiden Methoden zu zeigen, haben wir in der Datei **windmuehle_loeschen fla** verschiedene Arten von Instanzen zusammengemischt. Neben den von Hand erstellten Instanzen gibt es eine duplizierte, eine angefügte und eine leere, die gefüllt wurde. Im Folgenden sehen Sie den gesammelten Code für die drei in ActionScript erstellten Instanzen. Der Code wurde wie gewohnt in das erste Schlüsselbild des Hauptfilms auf der Ebene **Skript** eingefügt.

```
this.attachMovie("Wolke3", "wolke3", 0);
wolke3._x = 400;
wolke3._y = 80;

fluegel.duplicateMovieClip("fluegel_kopie", 1);
fluegel_kopie._rotation = 45;

this.createEmptyMovieClip("quadrat", 2);
with (quadrat) {
    farben = [0x0000FF, 0x000000];
    alpha = [100, 100];
    farbauftrag = [0, 0xFF];
    matrix = { matrixType:"box", x:170, y:140, w:28, h:28, r:0 };
    beginGradientFill("radial", farben, alpha, farbauftrag, matrix);
    moveto(170, 140);
    lineto(170, 168);
    lineto(198, 168);
    lineto(198, 140);
```

```

        lineto(170, 140);
        endFill();
    }

```

Vom Prinzip her ist Ihnen der Code bekannt. Allerdings müssen Sie bei mehreren Instanzen darauf achten, dass jede eine eigene Tiefe (Position) im Ebenenstapel erhält.



Nun wollen wir diese drei Instanzen und den Regen der zweiten Wolke entfernen. Dazu fügen wir einfach in das erste Schlüsselbild des Hauptfilms nach dem Code zum Erstellen der Instanzen folgenden Code ein:

```

fluegel_kopie.unloadMovie();
wolke3.removeMovieClip();
quadrat.removeMovieClip();

wolke.removeMovieClip();
regenwolke.mitregen.regen.unloadMovie();

```

Wenn Sie sich das Ergebnis in Abbildung 12.29 ansehen, fällt Ihnen sofort auf, dass die duplizierten, hinzugefügten und leer erstellten Movieclips mit beiden Methoden klaglos verschwunden sind. Das ist wirklich so, beide Funktionen löschen durch ActionScript erzeugte Instanzen.

Bei den von Hand erstellten Instanzen sieht es ein wenig anders aus. Der Regen wird mit `unloadMovie()` entfernt, `removeMovieClip()` versagt dagegen bei von Hand erstellten Movieclips. Die erste Wolke ist also noch vorhanden.

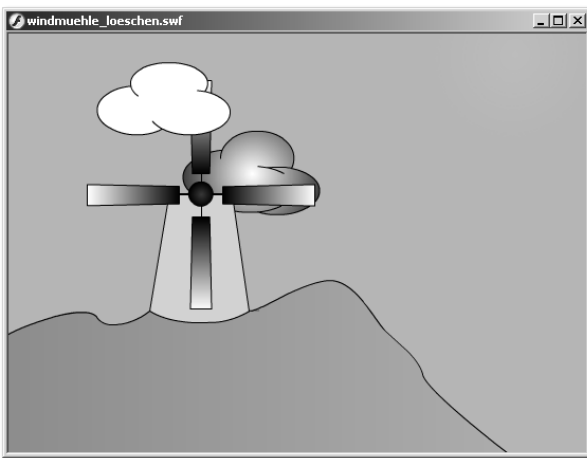


Abbildung 12.29 Was fehlt?



Natürlich ist es in der Praxis meist nicht sinnvoll, in einem Schlüsselbild Instanzen zu definieren und gleich wieder zu löschen. Hier geht es uns allerdings nur ums Testen.



Für `unloadMovie` gibt es noch eine abweichende Syntax. Sie können die Methode auch als globale Funktion verwenden, bei der das zu löschende Ziel in Klammern als Parameter übergeben wird, also beispielsweise:

```
unloadMovie("fluegel_kopie");
```

12.1.5 with-Anweisung

Die with-Anweisung ist eigentlich eine angenehme Vereinfachung. Sie erlaubt es, ein Objekt oder einen Movieclip nur einmal anzugeben und dann innerhalb der geschweiften Klammern alle möglichen Methoden und Eigenschaften des Objekts zu verwenden, ohne das Objekt noch einmal davor schreiben zu müssen.

```
with (Objekt) {  
    Methode;  
    Eigenschaft;  
}
```

Der Interpreter prüft, ob es sich jeweils um eine Eigenschaft oder Methode des Objekts handelt. Natürlich sind auch ganz andere Funktionen, Variablen und sogar Objekte mit ihren Methoden möglich.

Ein Anwendungsbeispiel haben Sie im Abschnitt »Leere Movieclips erzeugen« kennen gelernt. Dort haben wir mit den Zeichnungsmethoden eines Movieclips ein neues Element erstellt.



In Flash 4 gab es die with-Anweisung noch nicht. Dort wurde oft alternativ die Funktion `Tell Target` verwendet. Sie war damals auch die Methode, mit der Befehle für Movieclips angegeben werden konnten. `Tell Target` ist immer noch gestattet, sollte aber nur eingesetzt werden, wenn Abwärtskompatibilität mit Flash 4 notwendig ist.

```
Begin Tell Target (Instanzname)  
    Anweisung /*z. B. Set Property, um eine Eigenschaft zu set-  
zen oder Stop, um den Film anzuhalten.*/  
End Tell Target
```

12.1.6 Ereignisse für Movieclips

Die Instanzen von Movieclips können mit Ereignissen versehen werden. Der zugehörige Event-Handler ist `onClipEvent`. Man kann ihn durchaus als Gegenstück zu `on` bei Schaltflächen bezeichnen.

Achten Sie darauf, dass ein Movieclip-Ereignis immer in eine Movieclip-Instanz integriert ist. Es funktioniert weder in einem Schlüsselbild noch in einem Movieclip-Symbol.



Wir wollen Ihnen zuerst noch ein kleines Beispiel zeigen, bevor wir in einer Tabelle die möglichen Ereignisse für Movieclips zusammenfassen.

Ziel unseres Beispiels soll es sein, dass die erste Wolke aus **windmuehle fla** bei einem Tastendruck unsichtbar wird und beim nächsten Tastendruck wieder auftaucht.

1. Entsperren Sie die Ebene **WolkeNr1**, auf der die Instanz `wolke` liegt.
2. Dann aktivieren Sie die Instanz, indem Sie sie anklicken.
3. Im Aktionen-Bedienfeld können Sie nun Code für die Instanz eingeben. Als Event-Handler verwenden Sie `onClipEvent`, als Ereignis `keyDown`. Dieses Ereignis tritt ein, wenn der Anwender eine **beliebige** Taste drückt.

```
onClipEvent (keyDown) {
```

4. Nun überprüft eine Fallunterscheidung, ob die Wolke sichtbar ist oder nicht. Dazu wird die Eigenschaft `_visible` verwendet. Die aktuelle Movieclip-Instanz wird mit `this` referenziert, da der Code ja in ihr selbst eingefügt worden ist.

```
if (this._visible==true) {  
    this._visible = false;  
} else if (this._visible==false) {  
    this._visible = true;  
}
```

Ist der aktuelle Movieclip bereits sichtbar, wird er unsichtbar. Ist er nicht sichtbar, wird er wieder eingeblendet.

5. Vergessen Sie nicht, den Event-Handler am Schluss wieder mit einer geschweiften Klammer zu schließen.

Ihnen dieses Beispiel in einer Abbildung zu zeigen, wäre ein bisschen unsinnig. Wie sollte man auch zeigen, dass etwas nicht sichtbar ist ☺. Die

fertige Datei finden Sie unter dem Namen **windmuehle_ereignis.fla** auf der CD-ROM zum Buch.

Das Ereignis `keyDown` reagiert auf den Druck jeder beliebigen Taste und unterscheidet sich dadurch von dem Ereignis `keyPress`, das nur bestimmte Tasten erwartet. Dennoch gibt es auch bei `keyDown` Differenzierungsmöglichkeiten. Im Objekt **Taste** (Aktionen-Bedienfeld: **Objekte • Film • Taste**) finden Sie einige Methoden, mithilfe derer Sie mit den Tasteneingaben arbeiten können. Beispielsweise können Sie mit der Methode `getCode` den Tastencode der zuletzt gedrückten Taste auslesen. Da der Rückgabewert von `getCode` ein wenig schwierig auf die Taste umzulegen ist, gibt es alternativ mit `getAscii` auch die Möglichkeit, den ASCII-Code der Taste auszulesen. Dieser kann dann mit der Methode `fromCharCode` des String-Objekts umgewandelt werden.

Im folgenden Beispiel geben wir die Tasteneingaben einfach umgewandelt in das **Ausgabefenster** aus (siehe Abbildung 12.30).

```
onClipEvent (keyDown) {  
    var kc = Key.getAscii();  
    trace(String.fromCharCode(kc));  
}
```

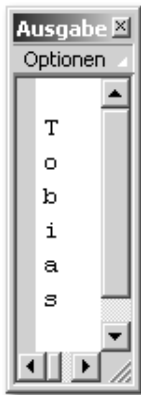


Abbildung 12.30 Die Tasten werden ausgegeben.



Ein ähnliches Objekt wie für Tasten gibt es auch für die Maus. Es findet sich unter **Objekte • Film • Maus**. Mit den zwei Methoden `hide` und `show` können Sie den Mauszeiger ein- und ausblenden.

In der folgenden Tabelle finden Sie alle Ereignisse für Movieclips in der Übersicht.

Ereignis	Beschreibung
Data	Daten werden in den Flash-Film geladen. Das können beispielsweise Variablen (mit der Movieclip-Methode <code>loadVariables</code>) oder ein Movieclip sein (<code>loadMovie</code>).
enterFrame	Wenn der Abspielkopf ein neues Bild im Film betritt. Dieses Ereignis wird für alle Bilder im Movieclip aufgerufen. Vorsicht: Wenn der Abspielkopf angehalten wird (mit den Methoden <code>Stop</code> oder <code>gotoAndStop</code>), wird das Ereignis für dieses Bild einmal ausgeführt, bevor der Abspielkopf gestoppt wird.
keyDown	Drücken einer beliebigen Taste
keyUp	Loslassen einer beliebigen Taste
load	Beim Laden eines Movieclips. Dies kann durch <code>ActionScript</code> (Duplizieren, Erzeugen, Anhängen) oder das Wandern des Abspielkopfs in ein Bild mit neuer Instanz geschehen. Ereignis wird auch aufgerufen, wenn Inhalte eines Movieclips mit <code>unloadMovie</code> entfernt werden, da die leere Instanz übrig bleibt, also »geladen« wird.
unload	Entladen (Entfernen) eines Movieclips. Dies passiert, wenn er zu Ende abgespielt oder wenn <code>removeMovieClip</code> entfernt wurde. Wenn externe Filme mit <code>unloadMovie</code> entfernt werden, wird das Ereignis ebenfalls aufgerufen.
mouseDown	Drücken der Maustaste. Dabei ist es egal, wo sich der Mauszeiger befindet.
mouseUp	Loslassen der Maustaste. Der Mauszeiger kann sich auch außerhalb des Movieclips befinden.
mouseMove	Bewegen der Maustaste. Der Mauszeiger kann sich auch außerhalb des Movieclips befinden.

Tabelle 12.1 Ereignisse für Movieclips

12.1.7 Ereignisprozeduren

Neu in Flash MX sind die so genannten Ereignisprozeduren. Das sind Ereignisse, die für einen Movieclip direkt und von überall abgefragt werden können und die dann eine Funktion aufrufen.

Die wichtigste Neuerung besteht darin, dass Sie mit Ereignisprozeduren unabhängig von der Instanz und dem `onClipEvent` mit Ereignissen für Movieclips arbeiten können. Außerdem gibt es einige neue Ereignisse für Movieclips, wie beispielsweise `onPress`, das in ähnlicher Form bisher den Schaltflächen vorbehalten war.

Sie funktionieren alle nach dem gleichen Muster.

```
Movieclip.Ereignis = Funktion();
```

Oder mit einem Beispiel:

```
wolke.onKeyDown = verstecken();
```

Sobald also für die Movieclip-Instanz `wolke` das Ereignis Tastendruck eintritt, wird die Funktion `verstecken` aufgerufen.

Ereignis	Beschreibung
onData	Daten werden in den Flash-Film geladen. Das können beispielsweise Variablen (mit der Movieclip-Methode <code>loadVariables</code>) oder ein Movieclip sein (<code>loadMovie</code>).
onDragOut	Die Maustaste wird innerhalb des Movieclips gedrückt und der Mauszeiger wird außerhalb des Movieclips gezogen.
onDragOver	Gegenstück zu <code>onDragOut</code> . Der Mauszeiger wird wieder in den Movieclip gezogen. Funktioniert nicht, wenn der Mauszeiger zwischendurch losgelassen wird. Auf der CD-ROM finden Sie ein Beispiel mit den beiden Ereignisprozeduren (<code>windmuehle_drag fla</code>). Im Beispiel wird eine Wolke ein- bzw. ausgeblendet, wenn der Anwender die Maustaste aus dem Quadrat auf der Mühle entfernt und wieder hineinzieht.
onEnterFrame	Tritt ein, wenn der Abspielkopf ein neues Bild im Film betritt. Dieses Ereignis wird für alle Bilder im Movieclip aufgerufen. Es gelten dieselben Besonderheiten wie beim Ereignis <code>enterFrame</code> .
onKeyDown	Drücken einer beliebigen Taste
onKeyUp	Loslassen einer beliebigen Taste
onKillFocus	Der Movieclip verliert den Fokus. Fokus heist, dass der Movieclip gerade per Tasteneingabe aktivierbar ist (in Windows erkennt man dies an einer schwarzen gestrichelten Umrandung, am Mac je nach Layout). In der Funktion für die Ereignisprozedur kann übergeben werden, welches Element anschließend den Fokus erhalten soll (Movieclip, Textfeld etc.).
onLoad	Beim Laden eines Movieclips. Dies kann durch ActionScript (Duplizieren, Erzeugen, Anhängen) oder das Wandern des Abspielkopfs in ein Bild mit neuer Instanz geschehen. Ereignis wird auch aufgerufen, wenn Inhalte eines Movieclips mit <code>unloadMovie</code> entfernt werden, da die leere Instanz übrig bleibt, also »geladen« wird.

Tabelle 12.2 Ereignisprozeduren

Ereignis	Beschreibung
onUnload	Entladen (Entfernen) eines Movieclips. Geschieht, wenn er zu Ende abgespielt oder wenn <code>removeMovieClip()</code> entfernt wurde. Wenn externe Filme mit <code>unloadMovie()</code> entfernt werden, wird das Ereignis ebenfalls aufgerufen.
onMouseDown	Drücken der Maustaste. Unabhängig von der Position des Mauszeigers
onMouseUp	Loslassen der Maustaste
onMouseMove onMouseUp	Bewegen der Maus
onPress onMouseUp	Beim Anklicken des Movieclips mit der Maustaste. Sie können also Movieclips wie Schaltflächen behandeln.
onRelease onMouseUp	Loslassen der Maustaste auf dem Movieclip
onReleaseOutside	Loslassen der Maustaste außerhalb des Movieclips
onRollOver	Mauszeiger bewegt sich über den Movieclip.
onRollOut	Mauszeiger verlässt den Movieclip wieder.
onSetFokus	Ereignis, wenn der Fokus auf einen Movieclip gesetzt wird.

Tabelle 12.2 Ereignisprozeduren (Forts.)

Ereignisprozeduren sind gegenüber normalen Ereignissen besonders flexibel, da sie nicht nur aus einer Instanz selbst aufgerufen werden können.



12.1.8 Eigenschaften

Mit den Eigenschaften steuern und lenken Sie das Aussehen und Verhalten eines Movieclips und erhalten wichtige Informationen, beispielsweise über die Position des Abspielkopfs oder über die Position des Mauszeigers.

In der folgenden Tabelle finden Sie die Eigenschaften von Movieclips im Überblick.

Eigenschaft	Beschreibung
<code>_alpha</code>	Transparenz (Alphawert) der gesamten Filmsequenz. Werte reichen von 100 (sichtbar, Standardeinstellung) bis 0 (unsichtbar).
<code>_currentframe</code>	Bild, bei dem sich der Abspielkopf aktuell befindet.

Tabelle 12.3 Eigenschaften von Movieclips

Eigenschaft	Beschreibung
<code>_droptarget</code>	Absoluter Pfad der Movieclip-Instanz, auf der der Movieclip abgelegt wurde. Der Pfad ist in alter Syntax dargestellt, also mit Schrägstrichen. Wandeln Sie diese also vor der Verwendung mit der Funktion <code>eval</code> in Punktsyntax um. Diese Eigenschaft kann nur gelesen werden.
<code>_foucsrect</code>	Eigenschaft, die festlegt, ob ein Movieclip, der den Fokus hat, mit einem gelben Rechteck hervorgehoben wird (ähnlich wie der Fokus auf Windows-Schaltflächen mit schwarzen Strichen). Wenn ja, ist der Wert <code>true</code> , wenn nein, <code>false</code> . Diese Funktion ist neu in Flash MX.
<code>_framesloaded</code>	Stellt fest, wie viele Bilder von einem gestreamten Film bereits geladen wurden. Kann nur gelesen werden und wird oft für Preloader eingesetzt.
<code>_height</code>	Die Höhe eines Movieclips in Pixeln
<code>_name</code>	Liefert den Instanznamen eines Movieclips. Wenn Sie diese Eigenschaft im Movieclip-Symbol abfragen, werden die Namen aller Instanzen der Reihe nach geliefert.
<code>_rotation</code>	Die Drehung des Movieclips in Grad. Achtung, hier wird nicht mit dem Bogenmaß gearbeitet.
<code>_target</code>	Liefert den absoluten Pfad für eine Movieclip-Instanz. Wird mit Schrägstrichen ausgegeben, kann aber mit der Funktion <code>eval</code> in Punktsyntax umgewandelt werden. Kann nur gelesen werden
<code>_totalframes</code>	Gibt die Anzahl der Bilder in der Zeitleiste des Movieclips aus. Kann nur gelesen werden
<code>_url</code>	Gibt den Link auf die .swf -Datei aus, in der der Movieclip beheimatet ist
<code>_visible</code>	Steuert, ob ein Movieclip sichtbar (<code>true</code>) oder unsichtbar (<code>false</code>) ist
<code>_width</code>	Die Breite eines Movieclips in Pixeln
<code>_x</code>	Enthält die x-Koordinate der linken oberen Ecke des Movieclips im Verhältnis zu der Koordinate des übergeordneten Movieclips. Der Hauptfilm hat in der linken oberen Ecke seinen Ursprung mit den Koordinaten <code>x = 0</code> und <code>y = 0</code> .
<code>_xmouse</code>	Liefert die Position des Mauszeigers auf der x-Achse. Kann nur gelesen werden
<code>_xscale</code>	Enthält die Skalierung des Films entlang der x-Achse (horizontal) in Prozent

Tabelle 12.3 Eigenschaften von Movieclips (Forts.)

Eigenschaft	Beschreibung
<code>_y</code>	Enthält die y-Koordinate der linken oberen Ecke des Movieclips im Verhältnis zu der Koordinate des übergeordneten Movieclips. Der Hauptfilm hat in der linken oberen Ecke seinen Ursprung mit den Koordinaten $x = 0$ und $y = 0$.
<code>_ymouse</code>	Liefert die Position des Mauszeigers auf der y-Achse. Kann nur gelesen werden
<code>_yscale</code>	Enthält die Skalierung des Films entlang der y-Achse (vertikal) in Prozent

Tabelle 12.3 Eigenschaften von Movieclips (Forts.)

Noch ein Hinweis: Für Eigenschaften gibt es auch noch eine andere Syntax, nämlich mit eckigen Klammern. Folgender Code würde die erste Wolke unseres Windmühlen-Beispiels unsichtbar machen:

```
wolke["_visible"] = false;
```

Es geht noch weiter: Mit den Funktionen `getProperty(Instanzpfad, Eigenschaft)` und `setProperty(Zielinstanz, Eigenschaft, Wert)` können Sie ebenfalls Eigenschaften auslesen und setzen. Diese Methoden wurden in Flash 4 verwendet, da es dort die Punktsyntax für den Zugriff auf Objekte noch nicht gab.

Die Wolke, die wir eben unsichtbar gemacht haben, können wir so wieder auftauchen lassen:

```
setProperty("wolke", _visible, true);
```

Bei `getProperty` und `setProperty` werden Instanzpfade immer als Strings, also in Anführungszeichen, angegeben.



12.1.9 Methoden

Von den verschiedenen Movieclip-Methoden haben Sie bereits einige kennen gelernt. Alle Movieclip-Methoden funktionieren nach derselben Syntax.

```
Movieclip.Methode(Parameter1, Parameter2, ...);
```

In der folgenden Tabelle stellen wir Ihnen die Movieclip-Methoden vor.

Methode	Beschreibung
<code>attachMovie(Symbolverknüpfung, Instanzname, Tiefe);</code>	Erstellt eine Instanz aus einem exportierten Symbol und fügt sie an den angegebenen Movieclip an (siehe den Abschnitt »Aus Symbol wird Instanz«)
<code>createEmptyMovieClip(Instanzname, Tiefe);</code>	Erstellt einen leeren Movieclip (siehe den Abschnitt »Leere Movieclips erzeugen«). Neu in Flash MX
<code>createTextField(Instanzname, Tiefe, x-Achse, y-Achse, Breite, Höhe);</code>	Erstellt ein Textfeld. Das Textfeld wird wie ein Movieclip behandelt und unter den Movieclip gehängt, in dem es vorkommt. Als Angaben werden der neue Name des Textfelds, die Tiefe im Ebenstapel, die x- und die y-Koordinate sowie Breite und Höhe angegeben. Diese Funktion ist neu in Flash MX.
<code>duplicateMovieClip(Instanzname, Tiefe);</code>	Kopiert eine bestehende Instanz (siehe den Abschnitt »Instanz duplizieren«)
<code>getBounds(Zielkoordinatensystem);</code>	Gibt die rechteckige Größe eines Movieclips in Koordinaten des Koordinatensystems der angegebenen Zielinstanz an. Es werden vier Werte übergeben: <code>xMin</code> , <code>xMax</code> , <code>yMin</code> , <code>yMax</code> . Die Variable, die die Rückgabe aufnimmt, ist vom Datentyp <code>Object</code> und hat diese vier Werte als setz- und auslesbare Eigenschaften. Sie finden im Anschluss an die Tabelle noch ein Beispiel.
<code>getBytesLoaded();</code>	Gibt an, wie viele Bytes eines gestreamten Movieclips bereits geladen worden sind. Wird verwendet, um den Ladezustand zu testen
<code>getBytesTotal();</code>	Gibt die Gesamtzahl der Bytes an, die ein Movieclip hat. Kann mit den bereits geladenen Bytes (<code>getBytesLoaded()</code>) verglichen werden
<code>getDepth();</code>	Liefert die Tiefe einer Instanz im Ebenstapel. Neu in Flash MX
<code>getURL(URL);</code>	Lädt eine relativ oder absolut verlinkte externe Seite in den Browser
<code>globalToLocal(Punkt);</code>	Wandelt die Koordinaten eines Punktes von Koordinaten des Koordinatensystems der Bühne in Koordinaten des Movieclip-Koordinatensystems um

Tabelle 12.4 Methoden für Movieclips

<code>gotoAndPlay(Bild);</code>	Springt auf das angegebene Bild des Movieclips und bewegt den Abspielkopf weiter
<code>gotoAndStop(Bild);</code>	Springt auf das angegebene Bild des Movieclips und hält den Abspielkopf an
<code>hitTest(x, y, Form);</code> <code>hitTest(Instanzname);</code>	Diese Methode testet auf Kollision und gibt einen Wahrheitswert zurück. Es kann für den aktuellen Movieclip entweder gegen eine Instanz, normalerweise ein Textfeld oder eine Grafik, oder gegen einen Bereich auf der Bühne mit festgelegten Koordinaten getestet werden. Die Form bestimmt, ob die Form selbst (<code>true</code>) für die Kollision entscheidend ist, oder das Begrenzungsrechteck (<code>false</code>). Im Anschluss an diese Tabelle finden Sie dazu noch ein kleines Beispiel.
<code>loadMovie(URL);</code>	Lädt einen externen Flash-Film (im swf-Format). Mehr dazu finden Sie im Abschnitt »Externe Filme«.
<code>loadVariables(URL);</code>	Lädt Variablen aus einer externen Datei oder über eine CGI-Schnittstelle (Common Gateway Interface)
<code>localToGlobal(Punkt);</code>	Wandelt die Koordinaten eines Punktes von Movieclip-Koordinaten in Koordinaten des Koordinatensystems der Bühne um
<code>nextFrame();</code>	Setzt den Abspielkopf auf das nächste Bild und hält die Wiedergabe an
<code>play();</code>	Setzt das Abspielen des Movieclips fort
<code>prevFrame();</code>	Setzt den Abspielkopf auf das vorhergehende Bild und hält an
<code>removeMovieClip();</code>	Entfernt per ActionScript erstellte Movieclips (siehe den Abschnitt »Löschen«)
<code>setMask(Maske);</code>	Setzt einen Movieclip für einen anderen als Maske. Es werden nur die sichtbaren Teile des maskierten Movieclips angezeigt, die auch in der Maske enthalten sind. Ein Beispiel finden Sie weiter unten in diesem Abschnitt.

Tabelle 12.4 Methoden für Movieclips (Forts.)

<code>startDrag(Einrasten);</code> <code>startDrag(Einrasten, x1, y1, x2, y2);</code>	Macht einen Movieclip ziehbar, dass heißt mit gedrückter Maustaste verschiebbar. Mit dem Wert <code>true</code> für das <code>Einrasten</code> geben Sie an, dass das Objekt mit dem Mittelpunkt einrastet. Mit <code>false</code> bestimmen Sie, dass es an der Stelle einrastet, an der es angeklickt wurde. Optional können Sie noch die Koordinaten (linke obere und rechte untere Ecke) eines Rechtecks angeben, in dem sich der Movieclip maximal bewegen darf. Diese Methode wird meistens bei einem Ereignis oder einer Ereignisprozedur wie <code>onPress</code> verwendet.
<code>stopDrag();</code>	Damit beenden Sie das Ziehen mit der Maus für einen Movieclip.
<code>stop();</code>	Hält den Abspielkopf an
<code>swapDepths(Tiefe);</code> <code>swapDepths(Instanzname);</code>	Vertauscht die Tiefe (Position) zweier Movieclips im Ebenenstapel. Der zweite Movieclip kann als Tiefe oder per Instanzname angegeben werden.
<code>unloadMovie();</code>	Entfernt den Inhalt einer Instanz oder einen ganzen Film. Die Instanz wird durch einen leeren Platzhalter ersetzt (siehe den Abschnitt »Löschen«).

Tabelle 12.4 Methoden für Movieclips (Forts.)

Wir hatten Ihnen noch ein kleines Beispiel für `getBounds` versprochen. Hier lösen wir unser Versprechen ein und lesen aus der Datei `windmuehle fla` die Koordinaten der Instanz `wolke` in Bezug auf das Koordinatensystem des Haupt-Movieclips aus.

```
var ob = wolke.getBounds(_root);
for (var teil in ob) {
    trace(teil + ":" + ob[teil]);
}
```

Als Ergebnis werden die vier Werte angezeigt (siehe Abbildung 12.31).

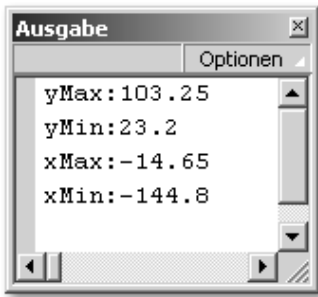


Abbildung 12.31 Die Koordinaten der ersten Wolke

Als zweites Beispiel wollen wir noch `setMask` einsetzen, um die zweite Wolke aus **windmuehle.fla** nur innerhalb der Flügel der Windmühle sichtbar zu machen.

Schreiben Sie einfach folgenden Code in das erste Schlüsselbild der Hauptzeitleiste:

```
fluegel.duplicateMovieClip("fluegel_kopie", 0);  
regenwolke.setMask(fluegel_kopie);
```

Es ist wichtig, dass Sie den Flügel duplizieren, da die Maske selbst nicht mehr sichtbar ist. In Abbildung 12.32 sehen Sie die Wolke im Flügel.

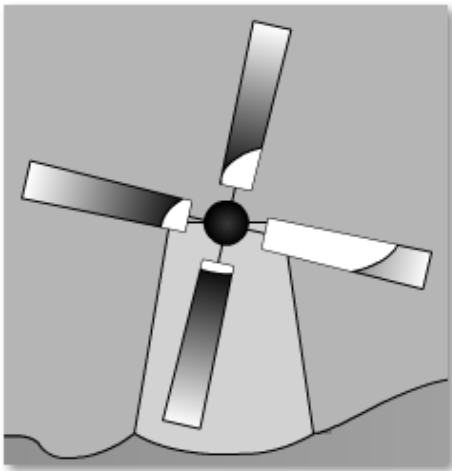


Abbildung 12.32 Die Wolke im Flügel

Zum Abschluss noch ein kleines Beispiel für `hitTest`. Mehr zu dieser Methode erfahren Sie dann in Kapitel 19, Spiele programmieren.

Hier befehlen wir unserer Instanz `regenwolke`, dass sie, wenn sie angeklickt wird, gleich um zehn Pixel nach oben und links springt.

```
regenwolke.onPress = function() {  
    trace(regenwolke.hitTest(_xmouse, _ymouse, true));  
    if (regenwolke.hitTest(_xmouse, _ymouse, true)) {  
        trace("yes");  
        regenwolke._x = regenwolke._x-10;  
        regenwolke._y = regenwolke._y-10;  
    }  
};
```

Die zugehörige Datei finden Sie unter dem Namen **windmuehle_hit-test.fla** auf der CD-ROM zum Buch.

12.2 Externe Filme

In diesem Abschnitt schneiden wir ein Thema an, das wir bisher beiseite gelassen haben, um Sie nicht mit verschiedensten Aussagen über Ebenen, Ebenenstapel und Ähnliches zu verwirren. Sie können mit ActionScript komplette **swf**-Filme in Ihren Hauptfilm laden.

Zum Laden von kompletten Filmen dient die Funktion `loadMovie`. Sie können Filme auf zwei Arten laden: in eine Instanz und in eine eigene Dokumentebene (Level).

Wenn Sie einen externen **swf**-Film in eine Instanz eines Movieclips laden, wird er in dieser Instanz in den Ebenenstapel eingefügt. Dafür muss die Instanz auf jeden Fall existieren.

```
Instanz.loadMovie("Filmpfad");
```

Der Filmpfad wird als normale URL angegeben. Liegt die geladene Datei im gleichen Verzeichnis, reicht der Dateiname. Aber auch die Angabe einer kompletten URL (z.B. <http://www.hauser-wenz.de/asjs/flash-film.swf>) ist möglich.

Es gibt noch eine andere Syntax, einen externen Film in eine Movieclip-Instanz zu laden. Dabei wird `loadMovie` als globale Funktion verwendet:

```
loadMovie("Filmpfad", "Instanz");
```

Die Instanz muss übrigens bei beiden Methoden korrekt adressiert werden.

Wenn Sie einen externen **swf**-Film auf eine eigene Dokumentebene (Level) laden, müssen Sie nur die Ebene angeben.

```
loadMovie("Filmpfad", Level);
```

Das Level wird entweder als ganze Zahl von 0 (unten) ab angegeben oder mit dem Schlüsselwort `_level`, gefolgt von der Nummer; dies dann aber als String. Hier ein Beispiel für beide Varianten:

```
loadMovie("Film.swf", 1);  
loadMovie("Film.swf", "_level1");
```

Beide Code-Zeilen produzieren das gleiche Ergebnis; sie laden den Film auf die Dokumentebene 1.

Als zweiten bzw. dritten Parameter kann `loadMovie` noch die Methode angeben, wie die Daten übertragen werden. Alternativen sind hier "GET" und "POST".



Sie stellen jetzt sicherlich zu Recht die Frage, was eine Dokumentebene eigentlich ist. Sie kennen bereits das Konzept der normalen Ebenen in der Zeitleiste. Außerdem wissen Sie, dass im Ebenenstapel über den Ebenen der Zeitleiste immer die Ebenen mit den Instanzen stehen, die per ActionScript erzeugt wurden (Tiefe und Konsorten, siehe den Abschnitt »Movieclips«).

Mit diesem Wissen wird es einfacher, das Dokumentebenen-Konzept zu verstehen. Es besagt, dass ein externer Film immer eine eigene Dokumentebene zugewiesen bekommt. Die Dokumentebene 0 (Level 0) ist der Hauptfilm. Wird ein Film in Level 1 geladen, liegen alle Instanzen, Ebenen der Zeitleiste etc. über dem Film auf Level 0.

Es handelt sich bei den Dokumentebenen also um eine übergeordnete Hierarchie. In Abbildung 12.33 sehen Sie das Prinzip noch einmal visualisiert.

Wenn Sie einen externen Film in Level 0 laden, wird der Hauptfilm überschrieben.



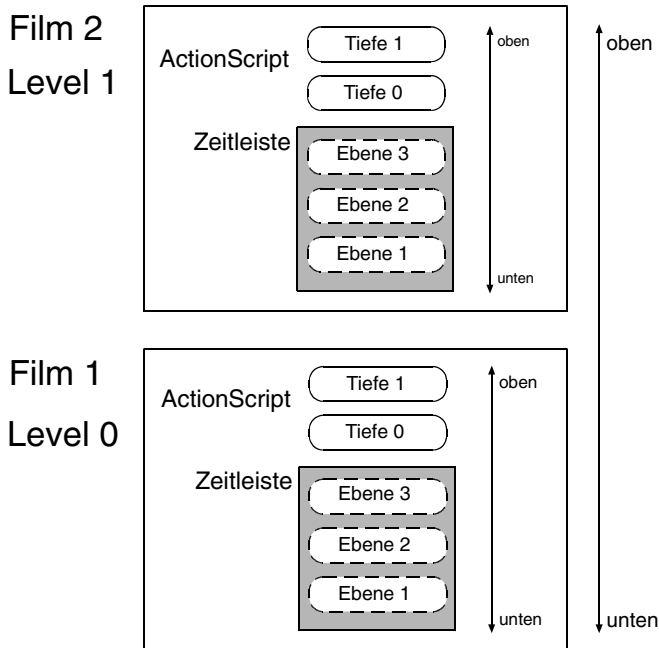


Abbildung 12.33 Ebenenstapel für verschiedene Filme



In Flash 3 und 4 wurde häufig auch die Methode `loadMovieNum` verwendet. Mittlerweile gilt sie als veraltet, `loadMovie` hat ihre Funktionalität übernommen. Mit ihr können Sie ein JPEG-Bild oder einen **swf**-Film auf ein Level laden, das Sie nur über die Level-Nummer angeben. Sie sehen, das kann **loadMovie** auch. Das gibt es aber eben erst ab Flash 4. `loadMovieNum` hat noch eine Besonderheit: Es funktioniert nicht als Methode einer Movieclip-Instanz, sondern nur als globale Funktion.



Die Funktion `attachMovie` funktioniert nicht mehr für Instanzen, die mit `loadMovie` mit einem externen Film gefüllt wurden. Das liegt daran, dass `attachMovie` auf ein Dokument und seine Bibliothek begrenzt ist.

12.2.1 Ein Beispiel

Bisher bestand dieser Abschnitt aus viel Theorie. Jetzt wird es konkreter. Wir verwenden eine externe **swf**-Datei, die wir dann in unseren Beispielfilm **windmuehle.fla** laden können.

Zuerst zum Aufbau der **swf**-Datei. Ihre Flash-Mutter ist **gewitter.fla**. Wenn Sie die Datei öffnen, fallen sofort ein paar Dinge auf:

- Es befindet sich kein Element in der Hauptzeitleiste.
- Es gibt nur eine Ebene namens **skript** mit einem ActionScript.

Wenn Sie daraus schließen, dass alle Elemente dieses Films (exportierte) Movieclip-Symbole sind, die per ActionScript aufgerufen werden, haben Sie richtig getippt. Es gibt insgesamt zwei Movieclip-Symbole:

- die Wolken, die aus Kreisen erstellt wurden. Sie haben einen linearen Verlauf von Schwarz nach Weiß,
- den Blitz, der mit Stiftwerkzeug erstellt wurde und der mit einem Verlauf von Gelb nach Weiß gefüllt ist.

Wenn Sie den Film testen (**Strg** + **↵**), sehen Sie, dass nacheinander drei Blitze auftauchen.

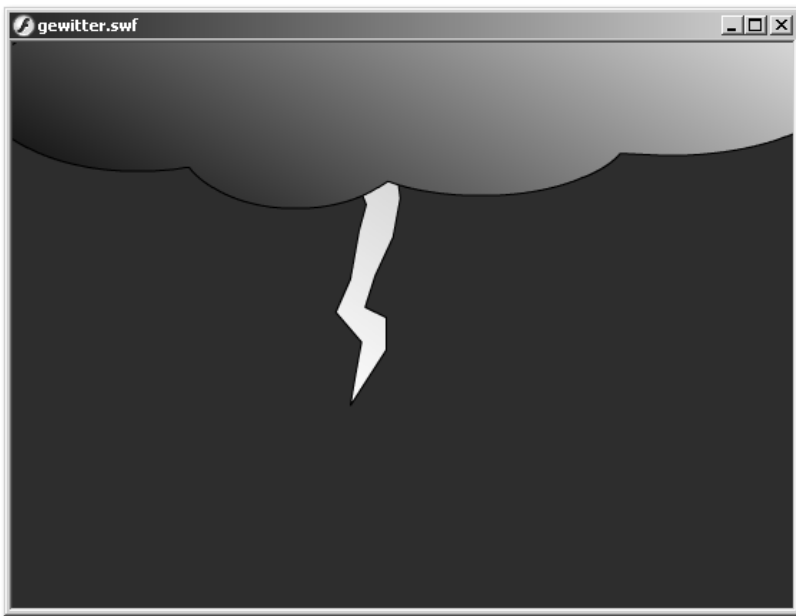


Abbildung 12.34 Ein Blitz im Gewitter

Das Auftauchen wurde im Movieclip-Symbol `blitz` realisiert. Dort wird der Blitz per Form-Tweening vergrößert, was ein Einschlagen simulieren soll (siehe Abbildung 12.35).

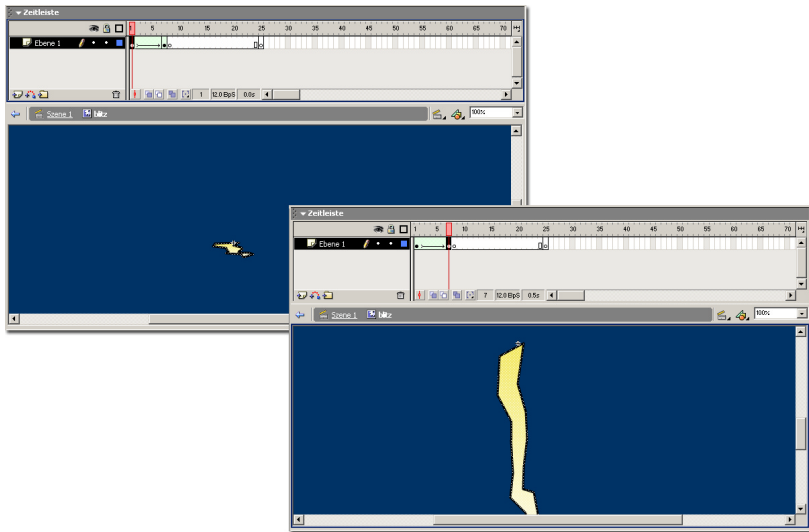


Abbildung 12.35 Der Blitz wird von klein (links oben) zu groß (rechts unten).

Dass die Blitze nacheinander erscheinen, wird durch das ActionScript im ersten Schlüsselbild der Hauptzeitleiste realisiert. Nachdem das Gewitter und die Blitze erstellt wurden, werden der zweite und der dritte Blitz angehalten. Anschließend wird für den ersten Blitz überprüft, ob er schon Bild 25 erreicht hat. Die zweite Bedingung ist, dass der zweite Film am Anfang ist. Sind beide Bedingungen erfüllt, wird der erste Blitz gestoppt und der zweite gestartet. Dies wird dann analog für den zweiten und den dritten Blitz durchgeführt.

```
stop();

this.attachMovie("gewitter", "gewitter", 4);
gewitter._x = 300;
gewitter._y = 40;
this.attachMovie("blitz", "blitz1", 1);
blitz1._x = 100;
blitz1._y = 30;
this.attachMovie("blitz", "blitz2", 2);
blitz2._x = 400;
blitz2._y = 20;
this.attachMovie("blitz", "blitz3", 3);
blitz3._rotation = 15;
blitz3._x = 280;
```

```

blitz3._y = 40;

blitz2.stop();
blitz3.stop();
blitz1.onEnterFrame = function () {
if (blitz1._currentframe == 25 && blitz2._currentframe == 1) {
    blitz1.gotoAndStop(1);
    blitz2.gotoAndPlay(1);
}
}
blitz2.onEnterFrame = function () {
if (blitz2._currentframe == 25 && blitz3._currentframe == 1) {
    blitz2.gotoAndStop(1);
    blitz3.gotoAndPlay(1);
}
}
blitz3.onEnterFrame = function () {
    if (blitz3._currentframe == 25) {
        blitz3.gotoAndStop(1);
        blitz1.gotoAndPlay(1);
    }
}
}

```

Wenn Sie eine Flash-Datei erstellen, von der Sie wissen, dass sie in eine Instanz einer anderen Flash-Datei geladen werden soll, verwenden Sie nur relative Adressierungen. Wenn Sie nämlich in dieser Datei beispielsweise `_root` verwenden würden, wären all diese Referenzen falsch, da der neue Haupt-Movieclip ein anderer ist.



Beide Dateien finden Sie auf der CD-ROM zum Buch. Die Flash-Datei heißt **gewitter fla**, die **swf**-Datei **gewitter swf**. Kopieren Sie für das weitere Beispiel die **swf**-Datei in dasselbe Verzeichnis wie **windmuehle fla**.



12.2.2 Film laden

Nun aber zur eigentlichen Intention des Beispiels, dem Laden des Films. Zuerst laden wir den Film auf eine neue Dokumentenebene von **windmuehle fla**. Dazu sind folgende Schritte nötig:

1. Wechseln Sie in die **Skript**-Ebene der Hauptzeitleiste und aktivieren Sie das erste (und einzige) Bild.
2. Im **Aktionen**-Bedienfeld können Sie nun den Code einfügen.

- Wir verwenden `loadMovie` hier als globale Funktion und übergeben die Dokumentebene als String mit dem Schlüsselwort `_level`.

```
loadMovie("gewitter.swf", "_level1");
```

Alternativ könnten Sie die Dokumentebene auch als Zahl angeben:

```
loadMovie("gewitter.swf", 1);
```

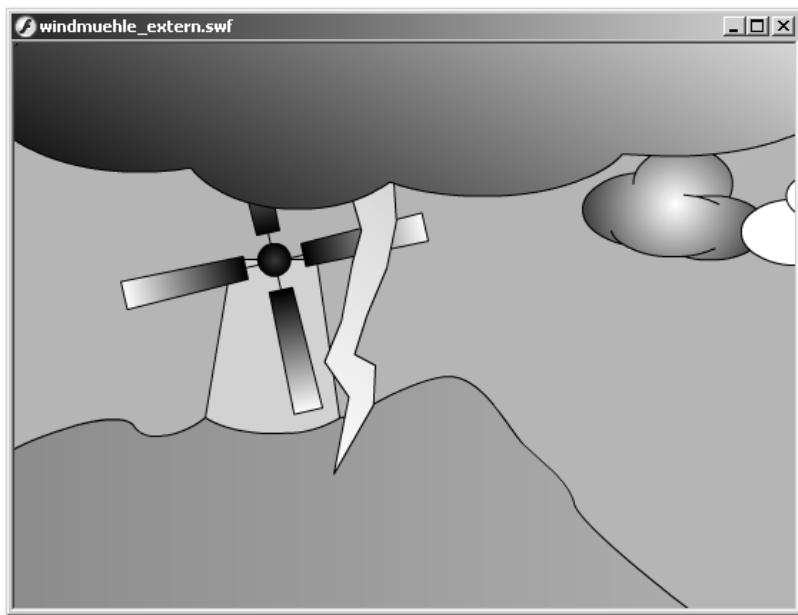


Abbildung 12.36 Jetzt blitzt es richtig.

In Abbildung 12.36 sehen Sie das Ergebnis. Der neue Flash-Film wurde einfach über den bestehenden gelegt. Die Datei zum Beispiel finden Sie auf der CD-ROM unter dem Namen **windmuehle_extern fla**.

Wenn Sie den externen **swf**-Film als neue Dokumentebene anlegen, können Sie natürlich kein Element, auch kein per ActionScript erzeugtes, darüber legen. Um das tun zu können, müssen Sie den externen Film in eine Instanz laden. Das führen wir ebenfalls für **windmuehle fla** vor:

- Zuerst geht es wieder in das erste Schlüsselbild der Ebene **Skript**, und nachdem die aktiviert ist, in das **Aktionen**-Bedienfeld.
- Nun erstellen wir mit `createEmptyMovieClip()` eine leere Instanz, in die dann der externe Film geladen werden kann. Natürlich können Sie

auch eine bestehende Instanz duplizieren oder von Hand eine Instanz erstellen. Der externe Film löscht so oder so alle Elemente der Instanz.

```
_root.createEmptyMovieClip("gewater", 1);
```

3. Dann laden wir den Flash-Film in die Instanz.

```
gewater.loadMovie("gewater.swf");
```

Wenn Sie lieber mit `loadMovie` als globaler Funktion arbeiten, übernehmen Sie die folgende Syntax. Von der Wirkung her macht das keinen Unterschied.

```
loadMovie("gewater.swf", "gewater");
```

Der Film wird in einem Anweisungsblock immer zuletzt geladen. Sie können den Film also nicht im selben Anweisungsblock noch manipulieren. Bei unserem Beispiel ist die Reihenfolge des Codes allerdings nicht von Bedeutung.



4. Nun wollen wir noch einige Anpassungen vornehmen, um die Datei zu verschönern und um Ihnen zu zeigen, was alles möglich ist. Zuerst duplizieren wir den Flügel und legen ihn eine Ebene (Position im Ebenenstapel, **nicht** Dokumentebene) über den hereingeladenen externen Film.

```
fluegel.duplicateMovieClip("fluegel_kopie", 2);  
fluegel._visible = false;
```

5. Als letzten Schritt färben wir noch die Movieclip-Instanz `wolke` um. Dazu verwenden wir das Objekt `Color`. Zuerst instanzieren wir ein Objekt. Dabei übergeben wir als Parameter den Movieclip, den wir färben wollen. Hier ist das `wolke`. Dann verwenden wir die Funktion `setRGB`, um einen Hexadezimalwert für die Farbe anzugeben. Damit wird die komplette Movieclip-Instanz eingefärbt.

```
var farbe = new Color(wolke);  
farbe.setRGB(0x333399);
```

Das Objekt `Color` verfügt über einige nützliche Methoden und Eigenschaften. Sie finden diese im **Aktionen**-Bedienfeld unter **Objekte • Film • Farbe**.



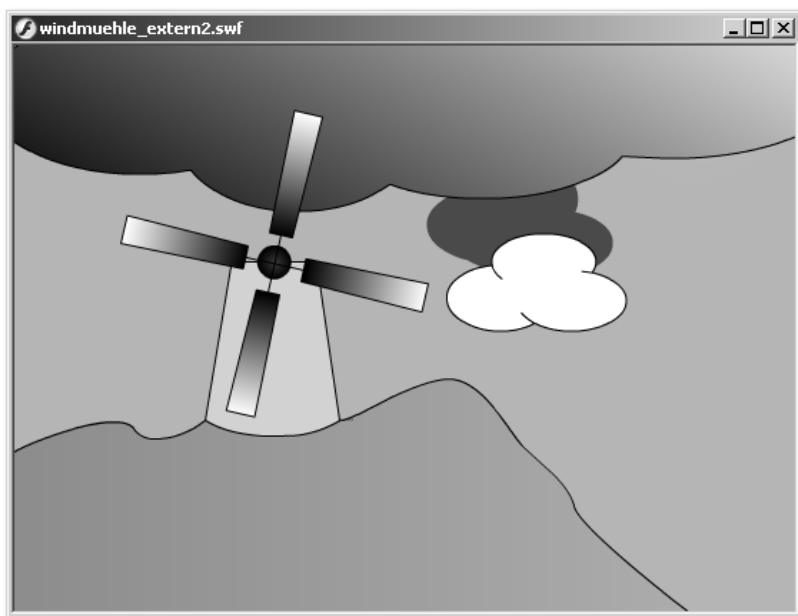


Abbildung 12.37 Die Gewitterstimmung



Wenn Sie möchten, können sie für die Gewitterstimmung die Hintergrundfarbe des Films noch etwas abdunkeln. Vorsicht: Sie müssen im Beispiel auch die blaue Verlaufsfarbe im Rechteck auf der Ebene **Himmel** anpassen.

Die Hintergrundfarbe des Hauptfilms gilt auch für alle externen Filme. Dasselbe gilt übrigens für die Maße. Ein neu geladener Film mit anderen Maßen wird immer am Ursprung des Hauptfilms ausgerichtet. Überstehende Bereiche werden nicht dargestellt. Sie sollten also möglichst mit gleich großen Filmen arbeiten.

Das Beispiel finden Sie auch auf der CD-ROM unter dem Namen **windmuehle_extern2 fla**.

12.2.3 Film löschen

Externe Filme löschen Sie genauso wie interne Movieclips, nämlich mit der Funktion `unloadMovie()`. `removeMovieClip()` funktioniert nur, wenn der externe Film in eine Instanz geladen worden ist.

Das Entladen mit `unloadMovie()` funktioniert über die Dokumentebene, also entweder mit

```
unloadMovie("_level1");
```

oder mit

```
unloadMovie(1);
```

Eine Instanz können Sie mit `removeMovieClip()` auf die im Abschnitt »Löschen« beschriebenen Arten entfernen.

13 Textfelder

*Ein Mann ein Wort, eine Frau ein Wörterbuch
– Autor unbekannt, wissenschaftlich widerlegt unter
<http://statistik.wu-wien.ac.at/mathstat/hatz/vasina.html>*

Das Großartige an Textfeldern in Flash ist, dass sie nun wirklich für jede Art von Text in Frage kommen. Egal, ob Sie einfach nur Text anzeigen wollen oder dem Anwender die Eingabe von Text erlauben. Dynamische Textfelder können sogar sehr einfach mit einer einzigen Variablen gefüllt werden. All dies wissen Sie schon, auch, dass Textfelder ein- oder mehrzeilig sein können und sogar HTML-Tags enthalten können.

In diesem Kapitel lernen Sie etwas ganz anderes kennen, nämlich, wie Sie Textfelder mit ActionScript erzeugen. Das geht erst in Flash MX, weswegen wir es in den meisten unserer Beispiele bisher nicht eingesetzt haben und auch in Zukunft nicht einsetzen werden; aber wenn Abwärtskompatibilität nicht erforderlich ist, sollten Sie diese Möglichkeit in Erwägung ziehen.

Welche Konsequenzen hat es, ein Textfeld mit ActionScript zu erstellen? Sie müssen es nur erstellen, wenn Sie es brauchen. Sie können es flexibel an Ihre Bedürfnisse anpassen und es sogar dynamisch aus Informationen einer Datenbank generieren. Gemach, gemacht, fangen wir langsam an. Zuerst zeigen wir Ihnen, wie Sie ein Textfeld erstellen, dann, welche Formatierungs- und Einstellungsmöglichkeiten Sie haben.

Die Formatierungs- und Einstellungsmöglichkeiten gelten allerdings nicht nur für per ActionScript erstellte Textfelder, sondern auch für manuell erstellte Textfelder. Auf die Felder greifen Sie dann mit dem Instanznamen zu, der allerdings vergeben sein muss.

13.1 Textfeld mit ActionScript erstellen

Ein Textfeld zu erstellen, ist ziemlich einfach und eine Methode des Movieclip-Objekts, nämlich `createTextField`.

```
this.createTextField("feld",1,200,150,300,100);
```

Wie man in der obigen Code-Zeile sieht, hat `createTextField` sechs wichtige Parameter:

- ▶ den Instanznamen des Textfelds,
- ▶ die Tiefe (Position) des Textfelds im Ebenenstapel. Das neue Textfeld wird dabei so behandelt wie ein neuer, per ActionScript erstellter Movieclip. Es erhält eine eigene Position im Ebenenstapel,
- ▶ die x-Koordinate der linken oberen Ecke des Textfelds,
- ▶ die y-Koordinate der linken oberen Ecke des Textfelds,
- ▶ die Breite des Textfelds,
- ▶ die Höhe des Textfelds.

Mit diesen Parametern wird das Textfeld erstellt. Allerdings erhält es noch keinen Text. Dies geschieht mit der Eigenschaft `text`.

```
feld.text = "Ein Textfeld";
```

Moment, eine Eigenschaft? Ja, genau, das Textfeld ist ein eigenes Objekt geworden, auf dessen Eigenschaften man ganz einfach zugreifen kann. Die Eigenschaften lernen Sie im nächsten Abschnitt kennen. Das Textfeld-Objekt finden Sie auch im **Aktionen**-Bedienfeld unter **Objekte • Film • TextField**.

13.2 Die Eigenschaften eines Textfelds

Zuerst ein paar Überlegungen zu den Eigenschaften. Wenn Sie ein Textfeld erstellen, ist es ein Objekt. Es hat standardmäßig alle Eigenschaften. Mit folgender Schleife können wir die Eigenschaften unseres Textfelds ganz einfach auslesen:

```
for (e in feld) {
    trace(e + ": " + feld[e]);
}
```

In Abbildung 13.1 sehen Sie alle Eigenschaften mit ihren Werten. Da wir außer **text** noch keine Eigenschaft geändert haben, sind das außerdem auch viele der Standardeinstellungen.

Natürlich gelten die Eigenschaften auch für nicht per ActionScript, also manuell erstellte Textfelder. Sie können auch bestehende Textfelder ändern.



Statische Textfelder lassen sich so natürlich nicht ändern, da ihnen kein Instanzname im Eigenschafteninspektor zugewiesen werden kann.

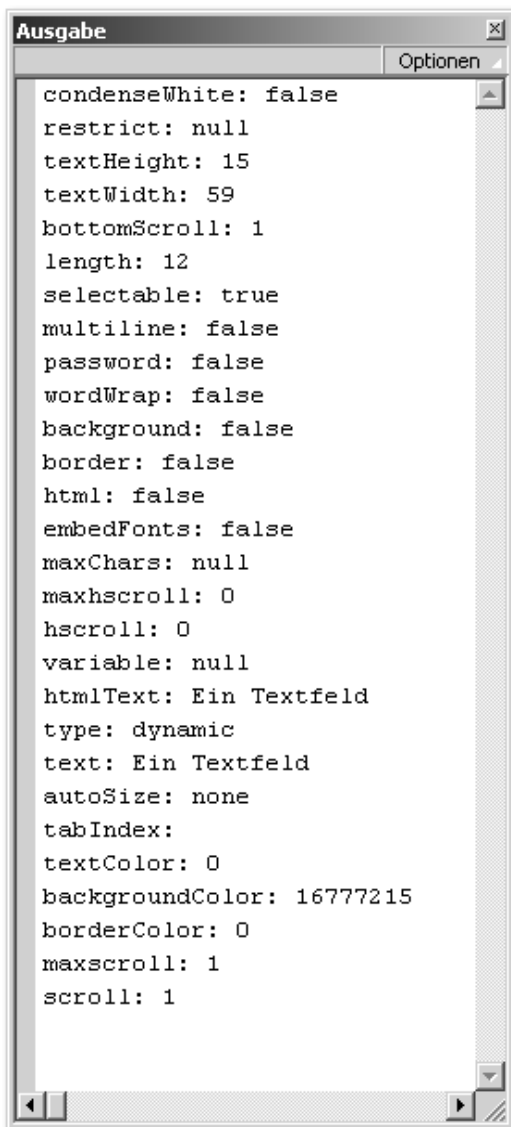


Abbildung 13.1 Die Eigenschaften für ein Textfeld

Sie finden die Datei mit der Schleife auf der CD-ROM unter dem Namen **text fla**.



Damit Sie die Eigenschaften nicht nur ausgeben können, sondern auch wissen, was sie bedeuten, haben wir in der folgenden Tabelle alle Eigenschaften für Sie zusammengefasst.

Eigenschaft	Standardwert	Beschreibung
autoSize	none	Gibt an, ob ein Textfeld sich an die Größe des Textes anpasst (<code>true</code>) oder nicht (<code>false</code>)
background	false	Gibt an, ob die Hintergrundfarbe angezeigt werden soll (<code>true</code>) oder nicht (<code>false</code>)
backgroundColor	16777215 entspricht 0xFFFFFF = Weiß	Hintergrundfarbe des Textfeldes als hexadezimale Farbe (0x000000 ist Schwarz). Wird nur angezeigt, wenn das Textfeld einen Rahmen hat
border	false	Gibt an, ob der Rahmen angezeigt wird (<code>true</code>) oder nicht (<code>false</code>)
borderColor	0 entspricht 0x000000 = Schwarz	Farbe des Textrahmens
bottomScroll	1	Die unterste im Textfeld sichtbare Zeile als ganze Zahl. In Verbindung mit <code>scroll</code> wird so der sichtbare Bereich des Textfelds dargestellt
embedFonts	false	Gibt an, ob für das Textfeld Schrift eingebettet wird (<code>true</code>) oder die installierten Schriften verwendet werden (<code>false</code>). In die Bibliothek eingebettete Schriften müssen als Verknüpfung exportiert werden.
hscroll	0	Gibt die horizontale Scrollposition als ganze Zahl an. Der Wert 0 verhindert horizontales Scrollen.
html	false	Es ist möglich, HTML-Tags in dynamischen Textfeldern zu verwenden. Diese Eigenschaft erlaubt dies (<code>true</code>) oder schaltet es aus (<code>false</code>).
htmlText	" "	Text des Textfelds mit HTML-Tags
length	12	Gibt die Länge des Textfelds in Zeichen an
maxChars	null	Gibt die maximale Zahl der Zeichen des Textfelds an. Gilt nur für Eingabetextfelder, also für die Eingabe des Anwenders. ActionScript kann mehr Zeichen einfügen.

Tabelle 13.1 Die Eigenschaften eines Textfelds

Eigenschaft	Standardwert	Beschreibung
maxhscroll	0	Gibt an, wie weit horizontal maximal gescrollt werden kann
maxscroll	1	Gibt den maximalen Wert für vertikales Scrollen an
multiline	false	Gibt an, ob ein Textfeld mehrere Zeilen haben kann (<code>true</code>) oder nicht (<code>false</code>)
password	false	Mit dieser Eigenschaft kann das Textfeld zu einem Passwortfeld werden, dessen Zeichen nicht angezeigt werden (<code>true</code>).
restrict	null	Kann die Eingabe von manchen Zeichen ausschließen. Die Zeichen werden als String angegeben. Ein Bereich kann mit Bindestrich (–) festgelegt werden (z.B. 1–5). Mit ^ werden alle Zeichen, die danach folgen, nicht zugelassen.
scroll	1	Die oberste sichtbare Zeile im Textfeld, also die vertikale Scroll-Position
selectable	true	Der Text im Textfeld ist auswählbar (<code>true</code>) oder nicht (<code>false</code>). Um aus einem dynamischen Textfeld quasi ein statisches zu machen, müssen Sie diese Eigenschaft auf <code>false</code> setzen.
tabIndex	undefined	Legt die Tabulator-Reihenfolge in der Seite fest, wenn die Eigenschaft für ein Objekt des Films festgelegt wurde
text	""	Der Text des Textfelds
textColor	0 entspricht 0x000000 = Schwarz	Bestimmt die Farbe des Textes als hexadezimalen Wert
textHeight		Höhe des Texts
textWidth		Breite des Texts
type	dynamic	Gibt die Art des Textfelds an. Die beiden Optionen sind »dynamic« (dynamisches Textfeld) und »input« (Eingabefeld).

Tabelle 13.1 Die Eigenschaften eines Textfelds (Forts.)

Eigenschaft	Standardwert	Beschreibung
<code>variable</code>	<code>null</code>	Der Name der Variablen, mit der auf ein dynamisches Textfeld zugegriffen werden kann als Zeichenkette
<code>wordWrap</code>	<code>false</code>	Erlaubt den automatischen Zeilenumbruch (<code>true</code>) oder unterbindet ihn (<code>false</code>)

Tabelle 13.1 Die Eigenschaften eines Textfelds (Forts.)



Neben diesen Eigenschaften gelten für Textfelder auch die Eigenschaften für Movieclips wie beispielsweise `_visible`. Außerdem gibt es noch die undokumentierte Eigenschaft `condenseWhite`.

13.3 Die Methoden und Ereignisse des Textfelds

Für ein Textfeld-Objekt gibt es natürlich auch einige Methoden und Ereignisse.

Die Ereignisse sind in der folgenden Tabelle zusammengefasst.

Ereignis	Beschreibung
<code>onChanged</code>	Ereignis beim Ändern des Textfelds. Es kann in einem Skript aufgerufen werden.
<code>onKillFocus</code>	Ereignis, wenn das Textfeld den Fokus verliert. In einer Funktion, die aufgerufen wird, kann das Objekt angegeben werden, das den Fokus erhalten soll.
<code>onScroller</code>	Wenn gescrollt wird, sich also die Position des Scrollbalkens verändert (siehe z. B. Eigenschaft <code>scroll</code>)
<code>onSetFocus</code>	Ereignis, wenn ein Textfeld den Fokus erhält, also für Tasten aktiv ist

Tabelle 13.2 Ereignisse des Textfeld-Objekts

Die Methoden finden Sie in der nächsten Tabelle.

Methode	Beschreibung
<code>addListener(Objekt);</code>	Legt ein Objekt fest, das aufgerufen wird, wenn das Ereignis <code>onChanged</code> oder <code>onScroller</code> eintritt
<code>getDepth();</code>	Liefert die Tiefe des Textfelds

Tabelle 13.3 Methoden des Textfeld-Objekts

<code>getFontList();</code>	Liefert alle Schriften inklusive der in der swf -Datei exportierten als Array
<code>getNewTextFormat();</code>	Liefert das Textformat-Objekt für neu eingegebenen Text
<code>getTextFormat();</code>	Liefert das Textformat-Objekt. Mehr dazu im nächsten Abschnitt
<code>removeListener();</code>	Entfernt ein Listener-Objekt
<code>removeTextField();</code>	Löscht ein Textfeld
<code>replaceSel(Text);</code>	Ersetzt ausgewählten Text durch neuen Text. Funktioniert nur, wenn <code>selectable true</code> ist
<code>setNewTextFormat(Textformat-Objekt);</code>	Setzt das Textformat-Objekt, also die Formatierungen für Text, der neu eingegeben wird
<code>setTextFormat(Textformat-Objekt);</code>	Setzt das Textformat-Objekt, das alle Formatierungen für ein Textfeld festlegt

Tabelle 13.3 Methoden des Textfeld-Objekts (Forts.)

13.4 Textfeld formatieren

Ein Textfeld können Sie nun also erstellen und auch sein Aussehen ein wenig beeinflussen. Was Sie noch nicht kennen gelernt haben, sind die Möglichkeiten, den Text im Textfeld zu formatieren. Hierfür gibt es ein eigenes Objekt namens **TextFormat**. Sie finden es auch im **Aktionen-Bedienfeld** unter **Objekte • Film • TextFormat**. Das **TextFormat**-Objekt weisen Sie einem Textfeld mit der Methode `setTextFormat` (bzw. `setNewTextFormat` für neu eingegebenen Text) zu und lesen es mit `getTextFormat` (bzw. `getNewTextFormat` für neu eingegebenen Text) aus.

Wir zeigen Ihnen das gleich einmal an einem Beispiel:

```
var format = new TextFormat("Arial", 24, 0xFFFFFF);

this.createTextField("feld", 1, 200, 200, 200, 50);
feld.text = "Ein Textfeld";
feld.background = true;
feld.backgroundColor = "0x660000";
feld.setTextFormat(format);
```

Im obigen Beispiel wird zuerst ein **TextFormat**-Objekt definiert. Es erhält als Standardwerte die Schriftart, die Schriftgröße und die Schriftfarbe. Anschließend wird ein Textfeld mit Hintergrundfarbe erstellt und das **TextFormat**-Objekt als Formatierung zugewiesen.

Das Beispiel finden Sie unter dem Namen **format fla** auf der CD-ROM.



Das TextFormat-Objekt muss definiert worden sein, bevor es zugewiesen werden kann.



Abbildung 13.2 Ein Textfeld mit formatiertem Text

Die Formatierungsangaben beim TextFormat-Objekt können entweder als Parameter angegeben oder später als Eigenschaften definiert werden. Außerdem gibt es noch einige Eigenschaften, die keine Parameter sind. Im Folgenden sehen Sie die allgemeine Syntax mit den (optionalen) Parametern. Die Eigenschaften finden Sie in der nachfolgenden Tabelle.

```
var format = new TextFormat(font, size, color, bold, italic,
underline, url, target, align, leftMargin, rightMargin, indent,
leading);
```

Eigenschaft	Beschreibung
font	Schriftart als Zeichenkette. Wenn nicht angegeben, wird die Standardschriftart verwendet
size	Schriftgröße in Punkt
color	Schriftfarbe als hexadezimaler Wert
bold	Gibt an, ob der Text fett ist (true) oder nicht (false)
italic	Gibt an, ob der Text kursiv ist (true) oder nicht (false)
underline	Gibt an, ob der Text unterstrichen ist (true) oder nicht (false)
url	Vergibt einen Hyperlink für den Text. Ein leerer String bedeutet, dass kein Hyperlink vorhanden ist.

Tabelle 13.4 Die Eigenschaften für das TextFormat-Objekt

Eigenschaft	Beschreibung
target	Gibt das Ziel des Hyperlinks an. Funktioniert wie das target-Attribut in HTML. Mögliche Werte sind <code>_blank</code> (im neuen Fenster), <code>_top</code> (im gleichen Fenster als oberstes Frameset), <code>_self</code> (im gleichen Fenster/Frame), <code>_parent</code> (im übergeordneten Frameset).
align	Textausrichtung. Werte sind <code>left</code> (linksbündig), <code>center</code> (zentriert) und <code>right</code> (rechtsbündig). Wird oft in mehrzeiligen Textfeldern (Eigenschaft <code>multiline</code>) mit Zeilenumbruch (<code>word-wrap</code>) eingesetzt
leftMargin	Abstand des Textes vom linken Rand des Feldes in Punkt
rightMargin	Abstand des Textes vom rechten Rand des Feldes in Punkt
indent	Einzug der ersten Zeile des Textes vom linken Rand (auch bei rechts ausgerichteten Text!)
leading	Zeilenabstand in Punkt. Ist standardmäßig zwar nicht definiert, es wird aber ein Zeilenabstand von etwa 120% der Schriftgröße verwendet
blockIndent	Einzug für einen Textblock in Punkt. Der Einzug gilt auch für mehrere Absätze, die beispielsweise mit <code>\n</code> eingefügt wurden.
bullet	Text wird als Aufzählungsliste formatiert (<code>true</code>) oder nicht (<code>false</code>). Das Aufzählungszeichen wird nicht an die Farbe des Textes angepasst.
tabStops	Gibt ein Array mit benutzerdefinierten Tabulatoren an. Ein Tabulator ist standardmäßig vier Punkt breit. Ein Tabulator wird mit <code>\t</code> eingefügt. Dies funktioniert aber nur in normalem, nicht in <code>htmlText</code> .

Tabelle 13.4 Die Eigenschaften für das `TextFormat`-Objekt (Forts.)

Sie können die Eigenschaften eines `TextFormat`-Objekts natürlich auch einfach mit einer `for-in`-Schleife auslesen. Die Standardwerte für alle Eigenschaften sind `null`, also nicht definiert.



Es gibt für das `TextFormat`-Objekt genau eine Methode: `getTextExtent(text)` liefert ein Objekt mit den Eigenschaften `width` und `height`, die die Größe des übergebenen Textes mit den entsprechenden Formatierungen angeben. Sie können damit also messen, wie groß ein Text mit den Formatierungen des `TextFormat`-Objekts in Pixeln wird.

Eine alternative Möglichkeit, Text in einem Textfeld zu formatieren, ist natürlich, die Eigenschaft `html` des Textfelds auf `true` zu setzen oder im Eigenschafteninspektor HTML zu akzeptieren. Dann stehen Ihnen ein



paar HTML-Tags zur Verfügung. Allerdings kennt Flash hier leider keine CSS-Style-Sheet-Befehle.

13.5 Ein Beispiel

Warum nicht Altes recyceln? In diesem Beispiel greifen wir auf die Funktionalität der Datei **namenssplitter.fla** aus Kapitel 11, Objekte, Eigenschaften und Methoden, zurück. Diese Datei enthält drei sehr unterschiedliche Textfelder: ein statisches, ein Eingabefeld und ein dynamisches Textfeld. Wir wollen nun alle drei mit ActionScript erzeugen.

1. Löschen Sie die drei Textfelder aus dem ersten (und einzigen) Schlüsselbild der Ebene **Textfelder**. Merken Sie sich aber die Koordinaten und die Größe, damit Sie sie exakt abbilden können. Sie können sie natürlich auch erst am Ende löschen, dann brauchen Sie sie sich nicht zu merken.
2. Wechseln Sie nun in das **Aktionen**-Bedienfeld für das erste Schlüsselbild dieser Ebene. Hier sollen die drei Textfelder als Elemente unter dem Haupt-Movieclip entstehen.
3. Zuerst definieren wir zwei TextFormat-Objekte. Das erste ist für den statischen Text und die Ausgabe bestimmt, das zweite für den Eingabetext, der etwas größer sein soll.

```
var klein = new TextFormat("Arial", 16, 0x000000, true);
var gross = new TextFormat("Arial", 24, 0x000000, true);
```

4. Als Erstes erstellen wir das statische Textfeld. Es erhält den Namen `beschriftung` und den Text »Ihr Name:«. Das Format wird mit der Methode `setTextFormat` zugewiesen.

```
_root.createTextField("beschriftung", 1, 176, 68, 78, 22);
beschriftung.text = "Ihr Name:";
beschriftung.setTextFormat(klein);
```

5. Nun folgt das Ausgabefeld. Hierfür sind schon ein paar Einstellungen mehr vorzunehmen. Es erhält ebenfalls sein Format, aber natürlich noch keinen Text. Es muss mehrere Zeilen umfassen können. Also setzen wir `multiline` auf `true`. Ein Zeilenumbruch mit `wordWrap` ist eigentlich nicht nötig, kann aber eingefügt werden, falls längere Namen erwartet werden. Zum Schluss muss unbedingt die Variable für das Ausgabefeld festgelegt werden. Sie heißt `ausgabe` und wird vom Skript in der Schaltfläche verwendet, um dem Feld später Text zuzuweisen.

```
_root.createTextField("unsichtbar", 2, 180, 195, 232, 184);
unsichtbar.setNewTextFormat(klein);
```

```
unsichtbar.multiline = true;  
unsichtbar.variable = "ausgabe";
```

Beachten Sie die Standardeinstellungen vieler Eigenschaften für Textfeld-Objekte. Diese müssen Sie natürlich nicht explizit setzen. So sind unsere beiden bisher erstellten Textfelder dynamische Textfelder, ohne dass wir die Eigenschaft `type` bestimmt hätten.



6. Das Eingabefeld wird erst zu einem solchen, wenn es den `type "input"` erhält. Ihm wird ebenfalls eine Variable namens `name` zugewiesen. Anschließend werden noch der Hintergrund und der Rahmen aktiviert.

```
_root.createTextField("eingabe", 3, 180, 108, 232, 30);  
eingabe.setNewTextFormat(gross);  
eingabe.type = "input";  
eingabe.variable = "name";  
eingabe.background = true;  
eingabe.border = true;
```

Das Skript mit der Funktionalität zum Teilen der eingegebenen Namen befindet sich nach wie vor in der Schaltfläche. Daran müssen Sie nichts ändern, da Sie den per `ActionScript` erstellten Textfeldern die dort verwendeten Variablen zugewiesen haben.

In Abbildung 13.3 sehen Sie das Ergebnis einer einfachen Eingabe. Die fertige Datei finden Sie unter dem Namen **textfelder fla** auf der CD-ROM zum Buch.

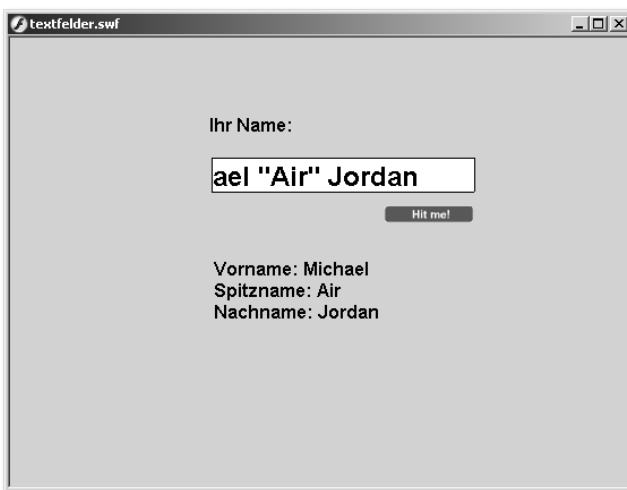


Abbildung 13.3 Die Ausgabe funktioniert mit per `ActionScript` erstellten Textfeldern.

14 Sound und Video

Ich verstehe nichts von Musik. In meinem Fach ist das nicht nötig.

– Elvis Presley

In diesem Kapitel wird es multimedial. Wir zeigen Ihnen, wie Sie mit ActionScript Sound und Videos steuern können und die eigene Klasse Sound von ActionScript nutzen.

14.1 Sound in Flash

Flash ist natürlich kein Sound-Programm. Daher werden Sie auch keine Möglichkeiten finden, eigene Sounds zu erstellen. Sie können allerdings sehr viele Arten von bestehenden Sounds importieren und verfügen über eine vorgefertigte Bibliothek mit eigenen Sounds (**Fenster • Allgemeine Bibliotheken • Sounds**).

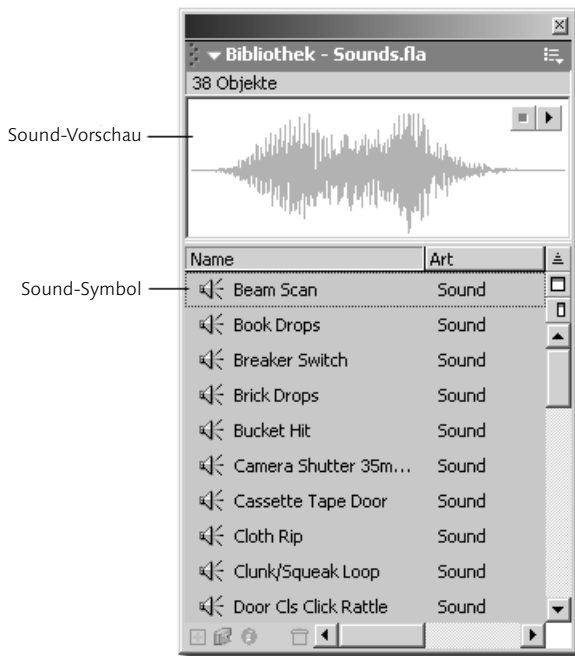


Abbildung 14.1 Die allgemeine Sound-Bibliothek

In der Bibliothek sehen Sie den Lautsprecher als Symbol für einen Sound und den Sound selbst im Vorschaufenster. Sie können die Sounds aus der Bibliothek einfach in die Bibliothek Ihrer Datei ziehen, um sie dort zu verwenden.



Die Bibliothek mit den Sounds ist im Prinzip nichts anderes als eine Flash-Datei, die im Programmverzeichnis von Flash liegt.

Man unterscheidet in Flash grob zwischen Streaming-Sound und Ereignis-sound. Bei Streaming-Sound wird schon während des Ladens mit dem Abspielen begonnen und der Sound wird mit der Animation synchronisiert. Da der Sound nicht verlängert oder verkürzt werden kann (klänge ziemlich komisch ☺), werden stattdessen automatisch Bilder aus der Animation entfernt, wenn die Synchronität nicht mehr erreicht werden kann. Ereignissound muss vollständig geladen werden und wird dann so lange abgespielt, bis er angehalten wird. So lassen sich sehr einfach Sound-schleifen (so genannte Loops) produzieren. Sie können also schon mit sehr wenig Sound und damit geringer Dateigröße eine längere Zeitspanne füllen.



In der Praxis sollten Sie Sound nicht zu lange in Loops abspielen, da der Anwender irgendwann genervt ist, wenn er immer das Gleiche hört. Ein guter Trick besteht darin, ab und zu ein Geräusch über einen Loop zu legen. Dadurch wirkt die Schleife nicht monoton.

Um Sound in Ihren Flash-Film einzubinden, müssen Sie ihn, wie schon erwähnt wurde, importieren. Dazu muss auf jeden Fall eine Ebene mit einem Schlüsselbild aktiviert sein, es sei denn, Sie importieren den Sound direkt in die Bibliothek (**Datei • In Bibliothek importieren**).

Der Import mit **Datei • Importieren** oder **Datei • In Bibliothek importieren** funktioniert für folgende Sound-Formate:

- ▶ **WAV, AIFF** und **MP3** unter Windows,
- ▶ **QuickTime** und **Sun AU** unter Windows, wenn mindestens QuickTime 4 installiert ist,
- ▶ alle eben genannten Formate unter Macintosh. Zusätzlich noch **SoundDesigner II** und **System 7**, wobei diese beiden wie Quicktime und **Sun AU** QuickTime 4 benötigen.

In Flash haben Sie dann mehrere Möglichkeiten, mit Sound umzugehen:

- ▶ Sie fügen ihn in ein Schlüsselbild ein (ein Sound pro Schlüsselbild). In der Zeitleiste wird der Sound angezeigt. Wenn Sie zusätzlich noch **Vor-**

schau aktivieren, sehen Sie noch mehr vom Sound und können ihn auch ganz gut mit der Animation synchronisieren.

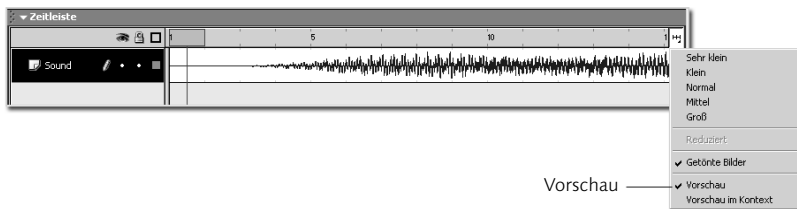


Abbildung 14.2 Sound in der Zeitleiste mit Vorschau

- ▶ Sie weisen den Sound einer Schaltfläche zu, so dass er beim Anklicken einmal aufgerufen wird. Dies funktioniert mit `ActionScript`.
- ▶ Sie laden einen Sound mit `ActionScript` und spielen ihn ab.

14.1.1 Globale Steuerung für Sounds

Bevor wir uns an die detailliertere Soundsteuerung in `ActionScript` heranwagen, rufen wir Ihnen noch einmal kurz in Erinnerung, welche globalen Eigenschaften und Funktionen es für Sounds gibt:

- ▶ `_soundbuftime` ist eine globale Eigenschaft und legt für alle Streaming-Sounds fest, wie lange gewartet wird, bis mit dem Streaming begonnen wird. Die Wartezeit wird in Sekunden angegeben. Die Standardeinstellung sind fünf Sekunden. Dies ist in den meisten Fällen ausreichend.
- ▶ `stopAllSounds()` ist eine globale Funktion, die alle Sounds im Dokument sofort anhält.

14.1.2 Das Sound-Objekt

Um mit Sound sinnvoll umgehen zu können, benötigen Sie das Sound-Objekt. Im **Aktionen**-Bedienfeld finden Sie sie unter **Core • Film • Sound**. Dieses Objekt hält einige Methoden und Eigenschaften für die Arbeit mit Sound bereit.

Ein neues Sound-Objekt wird mit dem Konstruktor `new` instanziiert.

```
var sound = new Sound();
```

Ein Beispiel

Nun wollen wir auch ein wenig mit dem Sound-Objekt arbeiten. Unsere Beispieldatei finden Sie auf der CD-ROM unter dem Namen **sound_vorb fla**. Sie enthält folgende Elemente:

- Zwei aus der allgemeinen Sound-Bibliothek (**Fenster • Allgemeine Bibliotheken • Sounds**) von Flash übernommene Sounds namens **Beam Scan** und **Cloth Rip**. Natürlich können Sie auch beliebige eigene Sounds verwenden.
- Der Sound **Beam Scan** wurde in eine Ebene **Sound** in die Hauptzeitleiste von Bild 2 bis Bild 25 eingefügt. Der Sound wird also automatisch abgespielt, sobald der Abspielkopf in Bild 2 angekommen ist.

Für den Sound können Sie einige Einstellungen im Eigenschafteninspektor vornehmen. Eine Vorstellung aller Möglichkeiten würde den Rahmen sprengen, deswegen greifen wir ein paar besonders interessante heraus. Mit den verschiedenen Effekten steuern Sie auf einfache Art die Wiedergabe. Sie können beispielsweise einem Kanal das Übergewicht geben. Hinter der Schaltfläche **Bearbeiten** verbirgt sich ein komplettes Dialogfenster, in dem Sie die Verteilung der Kanäle und die jeweilige Lautstärke noch detaillierter festlegen können. Warum sollten Sie solche Einstellungen in Flash und nicht in einem besser geeigneten Sound-Programm vornehmen? Hauptsächlich deshalb, weil Sie so verschiedene Instanzen desselben Sounds leicht ändern können, ohne die Dateigröße des Films mit einem weiteren Sound zu belasten. Die **Synchronisation** bestimmt, ob es sich um einen Ereignis- oder einen Streaming-Sound handelt. **Start** und **Stoppen** sind hier noch zwei weitere Optionen. **Start** spielt den Sound beim Erreichen eines Schlüsselbilds nur ab, wenn er noch nicht läuft, **Stoppen** hält den Sound an.

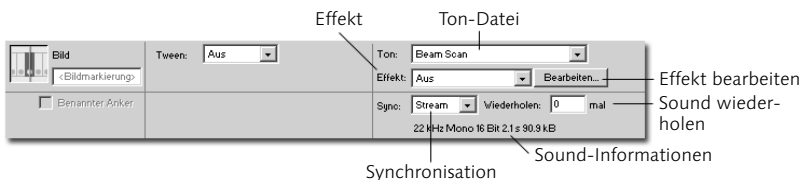


Abbildung 14.3 Einstellungen für den Sound im Eigenschafteninspektor

- Eine weitere Ebene, **Schaltfläche**, enthält einen Button mit der Beschriftung Sound von Bild 1 bis Bild 25. Der Button ist ein Schaltflächen-Symbol. Im Zustand **Drücken** ändert sich die Schriftfarbe von einem Verlauf in Weiß.

Erfahrene Flash-Programmierer kennen den Trick: Um Text mit einem Verlauf zu füllen, müssen Sie ihn zweimal teilen (**Modifizieren • Teilen** oder **Strg** + **B**).

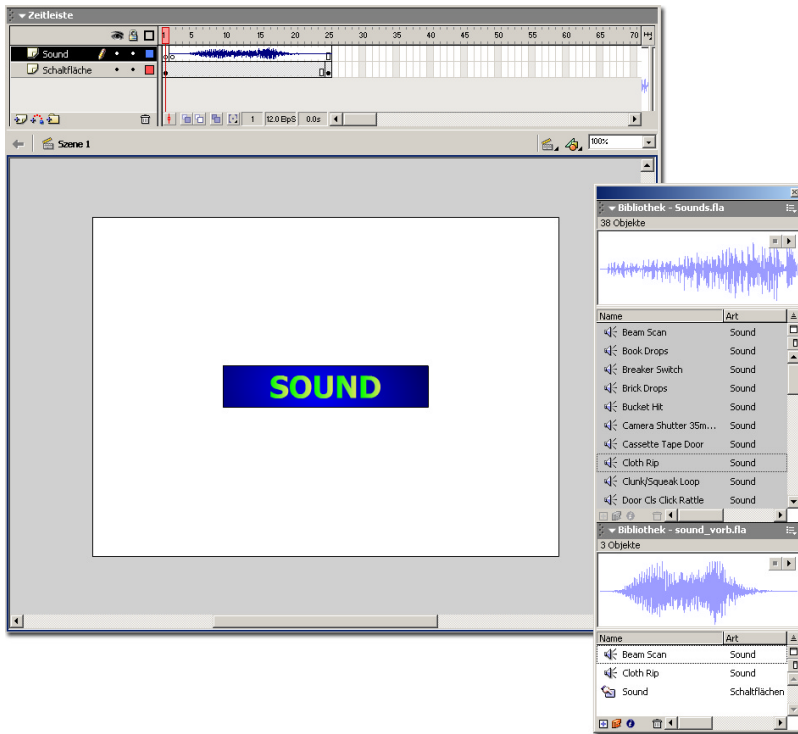


Abbildung 14.4 Die Beispieldatei ist bereits vorbereitet.

Nachdem wir die Vorbereitungen getroffen haben, beginnen wir mit der ActionScript-Programmierung.

1. Zuerst fügen wir eine Ebene für Skripts in die Zeitleiste ein.
2. In diese Ebene kommt als Erstes der Befehl `stop`, damit der Film nicht sofort weiterläuft und den Sound in Bild 2 startet.

```
stop();
```

3. Nun wollen wir ein neues Sound-Objekt mit dem zweiten Sound **Cloth Rap** aus der Bibliothek erstellen. Ganz wichtig ist, dass dieser Sound für ActionScript exportiert wird. Das ist im Prinzip genauso wie bei Movieclips: Ein nicht exportierter Sound wird einfach nicht mit veröffentlicht. Allerdings verbirgt sich das Exportieren beim Sound nicht unter **Eigenschaften**, sondern unter **Verknüpfungen**.



Abbildung 14.5 Der Export für ActionScript verbirgt sich bei Sound unter **Verknüpfung** im Kontextmenü.

4. Nun müssen Sie **Export für ActionScript** anklicken und einen **Bezeichner** eingeben. Wir haben **ritsch** gewählt. Mit diesem Bezeichner können Sie dann per ActionScript auf das Symbol in der Bibliothek zugreifen und den Sound verwenden.

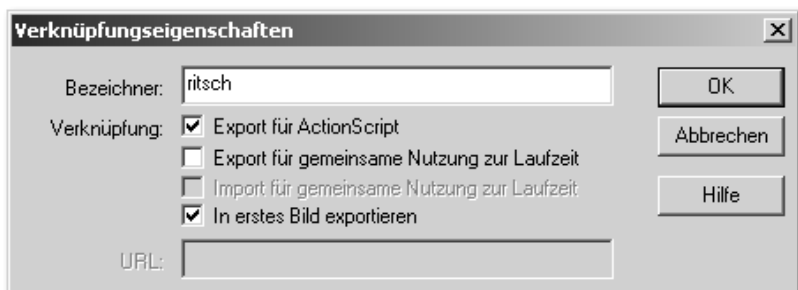


Abbildung 14.6 Verknüpfung und Bezeichner angeben

5. Als Nächstes erstellen wir im **Aktionen**-Bedienfeld für das erste Schlüsselfeld der Ebene **Sound** ein neues Sound-Objekt. Sie könnten dies

auch erst in der Schaltfläche tun, so ist der Code allerdings besser strukturiert und das Objekt ist in der ganzen Hauptzeitleiste verfügbar.

```
var ritsch = new Sound();
```

6. Die Methode `attachSound` des `Sound`-Objekts fügt den eben exportierten `Sound ritsch` hinzu.

```
ritsch.attachSound("ritsch");
```

7. Nun wechseln wir in das Skript der Schaltfläche. Dort wollen wir beide Sounds starten. Wir verwenden wir dafür das Ereignis `release`.

```
on (release) {  
    //Anweisungen  
}
```

8. Die erste Anweisung startet das neu erstellte `Sound`-Objekt. Hierzu dient die Methode `start`. Sie kann optional zwei Methoden annehmen: zum einen den Startzeitpunkt in Sekunden. Wir lassen den Sound mit 0 sofort starten. Zum anderen die Anzahl der Wiederholungen. Hier wählen wir 3.

```
ritsch.start(0,3);
```

Wenn Sie einen Sound in einer Schleife (Loop) abspielen möchten, gibt es dafür keinen eigenen Befehl, sondern Sie müssen die Anzahl der Wiederholungen einfach sehr hoch wählen. Die maximal mögliche Anzahl ist die größtmögliche Zahl in ActionScript (214783647).



9. Jetzt steuern wir den Sound in der Zeitleiste. Dafür brauchen wir keine Methode des `Sound`-Objekts. Wir setzen einfach den Abspielkopf auf Bild 2.

```
gotoAndPlay(2);
```

Wenn Sie einen Sound in einen `Movieclip` verfrachten, können Sie besonders flexibel mit ihm arbeiten und auf ihn zugreifen wie auf normale `Movieclips`.



Methoden und Eigenschaften

In diesem Abschnitt haben wir die verschiedenen Methoden und Eigenschaften der Klasse `Sound` gesammelt.

In der folgenden Tabelle finden Sie eine Übersicht über die verschiedenen Methoden.

Methode	Beschreibung
<code>attachSound(Name);</code>	Fügt einen durch Name identifizierten, vorher exportierten Sound aus der Bibliothek an
<code>getBytesLoaded();</code>	Gibt bei einem gestreamten Sound die Zahl der schon geladenen Bytes zurück. Kann mit <code>getBytesTotal</code> abgeglichen werden
<code>getBytesTotal();</code>	Liefert die Gesamtzahl der Bytes eines Sounds
<code>getPan();</code>	Liefert die Links-Rechts-Balance eines Sounds (welcher Kanal hat das Übergewicht) mit Werten von -100 (linker Kanal) bis 100 (rechter Kanal). Den Balance-Regler haben Sie beispielsweise auch an Ihrer Stereoanlage.
<code>getTransform();</code>	Liefert ein Transformationsobjekt. Ein Transformationsobjekt wurde als normales Objekt definiert und kann die folgenden vier Eigenschaften annehmen: ▶ <code>ll</code> – gibt an, wie viele Geräusche für den linken Kanal auch dort abgespielt werden ▶ <code>lr</code> – gibt an, wie viele Geräusche des linken Kanals rechts abgespielt werden ▶ <code>rr</code> – gibt an, wie viele Geräusche des rechten Kanals dort abgespielt werden ▶ <code>rl</code> – gibt an, wie viele Geräusche des rechten Kanals links abgespielt werden Bei Stereo-Sound sind standardmäßig alle Geräusche auf die beiden Kanäle gleich verteilt, also <code>ll = 100</code> und <code>rr = 100</code>, bei Monosound sind alle Geräusche im linken Kanal (<code>ll = 100</code>, <code>lr = 100</code>).
<code>getVolume();</code>	Liefert die Lautstärke von 0 (nicht hörbar) bis 100 (sehr laut)
<code>loadSound(URL, Streaming);</code>	Dient der Arbeit mit MP3-Dateien. Lädt eine Datei von einer URL. Der zweite Parameter gibt an, ob die Datei im Streaming geladen werden soll, ob also schon beim Laden mit dem Abspielen begonnen werden soll (<code>true</code>). Wenn nicht (<code>false</code>), handelt es sich um Ereignissound, der vollständig geladen werden muss, bevor das Abspielen startet. Dies ist beispielsweise beim Anklicken einer Schaltfläche sinnvoll, da hier ein verzögertes Streaming unsinnig wäre. Diese Methode ist neu in Flash MX.
<code>setPan(Balance)</code>	Setzt die Links-Rechts-Balance mit Werten von -100 (links) bis 100 (rechts)
<code>setTransform(Transformationsobjekt);</code>	Setzt ein Transformationsobjekt (siehe Erläuterung bei <code>getTransform</code>). Ein Beispiel finden Sie in der Datei transformation fla .

Tabelle 14.1 Die Methoden für Sound

<code>setVolume(Lautstärke);</code>	Bestimmt die Lautstärke von 0 (nicht hörbar) bis 100 (sehr laut)
<code>start();</code> <code>start(Start, Wiederholung);</code>	Startet einen Sound. Optional können Sie den Startzeitpunkt in Sekunden und die Anzahl der Abspielwiederholungen angeben.
<code>stop();</code> <code>stop(Name);</code>	Hält alle Sounds bzw. den Sound mit dem angegebenen Namen an

Tabelle 14.1 Die Methoden für Sound (Forts.)

In Flash MX gibt es außerdem für das Sound-Objekt noch zwei Eigenschaften. Beide können nur gelesen, nicht aber gesetzt werden.

- ▶ `duration` enthält die Dauer des Sounds in Millisekunden.
- ▶ `position` gibt an, wie viele Millisekunden der Sound schon abgespielt wurde.

Für Sound gibt es außerdem zwei spezielle Ereignisse. `onLoad` wird aktiv, wenn der Sound geladen ist, `onSoundComplete` tritt ein, wenn der Sound vollständig wiedergegeben wurde. `onSoundComplete` ist neu in Flash MX und kann vor allem dazu genutzt werden, das exakte Ende auch bei Streaming-Sounds abzugreifen.



14.1.3 Fähigkeiten des Clients testen

Im **Aktionen**-Bedienfeld unter **Objekte • Film • System** gibt es ein Objekt namens `capabilities`. Es liefert Informationen über das System, auf dem der Flash-Player läuft. Hierunter sind auch einige über Sounds:

- ▶ Die Eigenschaft `hasAudio` überprüft, ob der Computer überhaupt eine Audio-Unterstützung hat. Die Qualität der Audio-Unterstützung kann allerdings nicht getestet werden.
- ▶ `hasAudioEncoder` testet, welche Audio Encoder (Player für bestimmte Dateiformate) vom System unterstützt werden. Die Encoder werden in einem Array abgelegt, zurückgeliefert wird ein Wahrheitswert, der angibt, ob es AudioEncoder gibt. Dies funktioniert nur, wenn die Informationen auch korrekt vom Flash-Player übertragen werden. Ansonsten ist das Array leer.
- ▶ `hasMP3` überprüft, ob das System MP3 unterstützt.

Im folgenden Beispiel werden alle drei Eigenschaften getestet:

```
trace(System.capabilities.hasAudio);  
trace(System.capabilities.hasAudioEncoder);  
trace(System.capabilities.hasMP3);
```

Für Videos gibt es eine entsprechende Eigenschaft. `hasVideoEncoder` liefert einen Wahrheitswert, ob es einen Video Encoder gibt. Ein Array enthält die möglichen Video Encoder.

```
trace(System.capabilities.hasVideoEncoder);
```



`System.capabilities` hält noch einige Eigenschaften bereit. Die mit am häufigsten verwendeten sind `screenResolutionX` und `screenResolutionY`, die die Breite und Höhe der Bildschirmauflösung enthalten.

```
trace(system.capabilities.screenResolutionX);
```

```
trace(system.capabilities.screenResolutionY);
```

In der Datei **ueberpruefung fla** auf der CD-ROM finden Sie die hier vorgestellten Eigenschaften.

14.2 Video

Zur Video-Behandlung in ActionScript lässt sich eigentlich wenig sagen. Prinzipiell müssen Sie ein Video in einen Movieclip konvertieren, um dann mit ActionScript daran herumspielen zu können. Wenn Sie ein Video nicht in einen Movieclip konvertiert haben, können Sie natürlich auch die normale Zeitleiste steuern und beispielsweise anhalten oder weiterspielen.

Wir zeigen Ihnen im Folgenden, wie Sie ein Video importieren, in einen Movieclip konvertieren und mit einfachen ActionScript-Befehlen steuern.



Videodaten werden in Flash entweder als **swf** oder als **mov** (Movie; Quicktime-Dateiformat) veröffentlicht. Importierte Videos können aber auch das Format **avi** (Audio Video Interleaved; Video-Format unter Windows) oder **mpg** beziehungsweise **mpeg** (Motion Picture Expert Group) haben. Für diese Formate muss allerdings unter Windows und Macintosh Quicktime 4 oder unter Windows DirectX ab 7 vorhanden sein. Für importierte **mov**-Filme und **dv**-Dateien (Digital Video) benötigen Sie unbedingt Quicktime, für **wmf** (Windows Media File) DirectX.

Zuerst benötigen wir einen geeigneten Film. Wir haben die Datei **clock.avi** gewählt. Unter Windows finden Sie sie im Hauptverzeichnis. Unter MacOS wählen Sie eine beliebige andere Videodatei.

Aus urheberrechtlichen Gründen können wir die Video-Datei nicht beilegen. Wenn Sie aber, egal ob am Mac oder unter Windows, nach einer Datei mit der Dateierendung eines Videoformats wie **.avi** oder **.mov** suchen, werden Sie sicher fündig und können diese Datei zum Testen verwenden.



1. Zuerst benötigen Sie eine neue Datei (**Datei • Neu**). Die Bühnengröße beträgt 550 * 400 Pixel, die Hintergrundfarbe ist Weiß und für das Beispiel eigentlich unerheblich.
2. Dann importieren wir das Video (**Datei • Importieren**). Sie sehen dort gleich eine neue Oberfläche gegenüber Flash 5. In Flash MX wurde der Sorenson Spark Codec, ein Komprimier-Codec für Videos, integriert.

Das ist übrigens ein sehr interessantes Thema, mit dem Sie sich bei Interesse unbedingt einmal beschäftigen sollten. In der Online-Hilfe (**Hilfe • Flash verwenden**) oder in einem guten Flash-Buch wie beispielsweise Sascha Wolters, bei Galileo Press erschienen, finden Sie weitere Informationen.

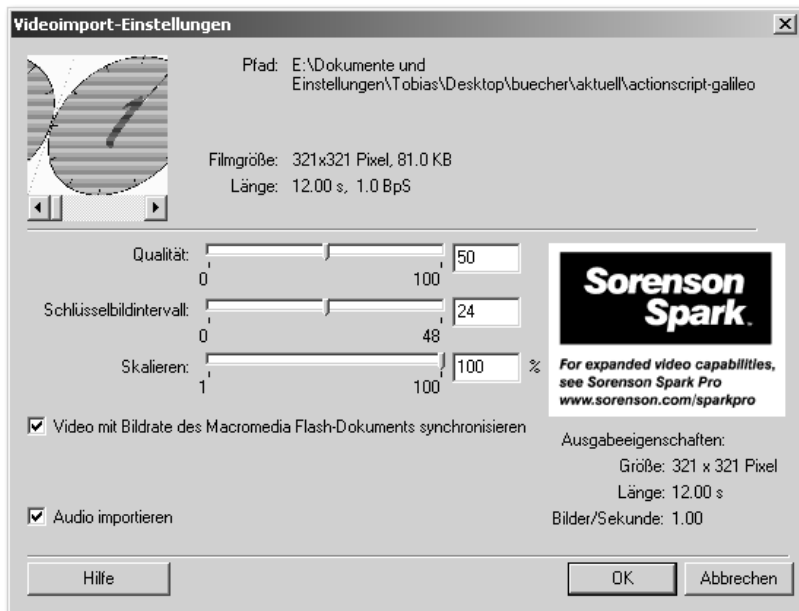


Abbildung 14.7 Ein Video importieren

3. Nun wird gefragt, ob Sie das Bild auf mehr Bilder ausdehnen wollen, da die neue Datei nur ein Bild beinhaltet. Die Anzahl der Bilder wird aus der Abspiellänge und der Bildabspielrate des Films errechnet. Unser Video ist also etwas über 11 Sekunden ($133 \text{ Bilder} / 12 \text{ bps} = 11,083$) lang.

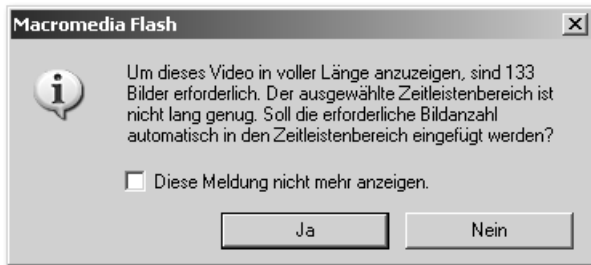


Abbildung 14.8 Wollen Sie 133 Bilder?

4. Klicken Sie auf **Nein**. Sonst würden Sie die Hauptzeitleiste ausdehnen. Wir wollen das Video aber als Movieclip laufen lassen.
5. Anschließend konvertieren wir das Video in ein Symbol. Hierzu können Sie es auf der Bühne mit der rechten Maustaste anklicken und im Kontextmenü **In Symbol konvertieren** wählen.
6. Das Symbol wird natürlich zum Movieclip. Wichtig ist, dass Sie **Export für ActionScript** aktivieren. Das Video hat hier dieselben Funktionen wie ein normaler Movieclip (siehe Kapitel 12, MovieclipsÜberprüfen).
7. Wenn Sie das Video in ein Symbol konvertieren, erscheint noch einmal die Frage, ob Sie die Zahl der Bilder erhöhen möchten (siehe Abbildung 14.9). Diesmal ist die Antwort **Ja**, denn damit werden 133 Bilder im Movieclip geschaffen.
8. Nun müssen noch die Steuerungselemente für das Video hinzugefügt werden. Wir holen sie aus der Datei **tennisball_mit_schalt fla**, die sich auf der CD-ROM in dem Verzeichnis für das Kapitel »Fortgeschrittene Programmierung« befindet.
9. Importieren Sie die Schaltflächen von der Bibliothek von **tennisball_mit_schalt fla** in die Bibliothek mit dem Flash-Film des Videos.



Abbildung 14.9 Das Video in ein Symbol konvertieren

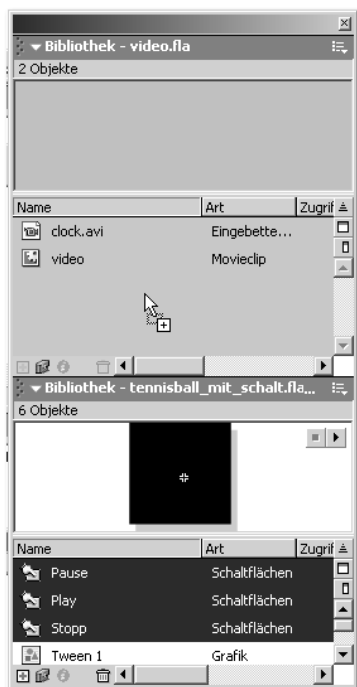


Abbildung 14.10 Die Schaltflächen in die Bibliothek des Flash-Films mit dem Video verschieben

10. Nun müssen Sie von jeder Schaltfläche eine Instanz auf der Bühne erstellen. Diese Instanzen nehmen dann den Action-Script-Code für die Videosteuerung auf.
11. Zuerst kümmern wir uns um die Stopp-Taste. Wenn sie gedrückt wird, springt der Abspielkopf auf den Anfang des Movieclips und hält dort an. Dadurch wird das Video sozusagen zurückgespult.

```
on (release) {  
    video.gotoAndStop(1);  
}
```

12. Die Pause-Taste hält den Abspielkopf an. Dazu wird die Movieclip-Methode `stop` verwendet.

```
on(release) {  
    video.stop();  
}
```

13. Die Play-Taste setzt den Movieclip mit der Methode `play` fort.

```
on (release) {  
    video.play();  
}
```

14. Wenn Sie den Film testen, haben Sie jetzt die komplette Kontrolle über den Ablauf des Videos.

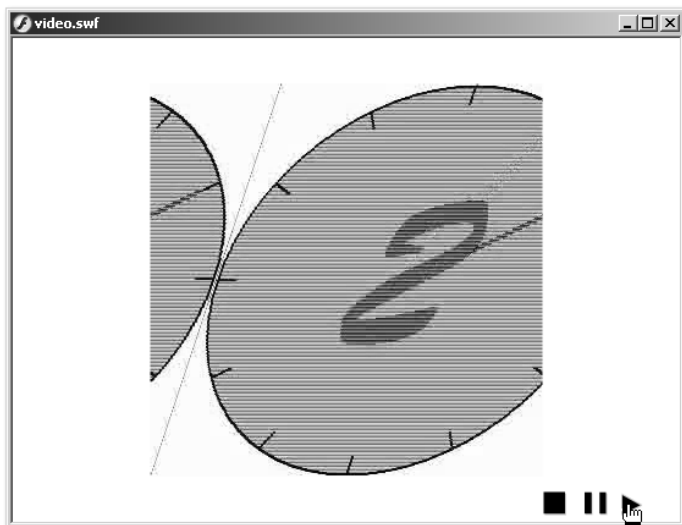


Abbildung 14.11 Nach einer Pause weiterspielen

15 Debugging

Seien Sie vorsichtig mit Gesundheitsbüchern – Sie können an einem Druckfehler sterben.

– Mark Twain

In der Programmierung unterscheidet man grob zwei Arten von Fehlern: Syntaxfehler und logische Fehler. In Flash kommt noch ein dritter Fehler hinzu, der »dümmer Anwender«-Fehler. Ein Fehler in der Bedienung, mit dem keiner rechnet, den ein sehr unerfahrener Anwender aber dennoch machen kann.

15.1 Testen

Die erste Grundregel zur Fehlervermeidung heißt: immer wieder testen. Sie sollten Ihren Code schon in der Entstehungsphase öfter testen, damit Sie später nicht in umfangreichen Anwendungen kleine Fehler suchen müssen.

Die wichtigste Methode zum Testen ist natürlich **Steuerung • Film testen** (oder **Strg** + **↵**). Bei Syntaxfehlern ist dies eine gute Option. Sie können vorher allerdings schon im **Aktionen**-Bedienfeld nach Syntaxfehlern suchen.

Um Ihnen die Fehlersuche zeigen zu können, haben wir in ein Beispiel aus einem vorangegangenen Kapitel einen Fehler eingebaut. Sie finden es unter dem Namen **split_fehler fla** auf der CD-ROM. Vielleicht finden Sie den Fehler auch beim bloßen Ansehen der Datei? Ein Tipp: Er ist in der Schaltfläche versteckt.

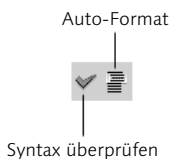


Abbildung 15.1 Die zwei Optionen im **Aktionen**-Bedienfeld

Im **Aktionen**-Bedienfeld gibt es zwei Schaltflächen, die etwas mit Fehlererkennung zu tun haben: **Auto-Format** rückt Ihren Quellcode automatisch ein und ordnet ihn korrekt an. Dies ist aber nur möglich, wenn die

Syntax stimmt. Ist das nicht der Fall, wie in unserem Beispiel **split_fehler fla**, wird gleich eine Fehlermeldung ausgegeben.

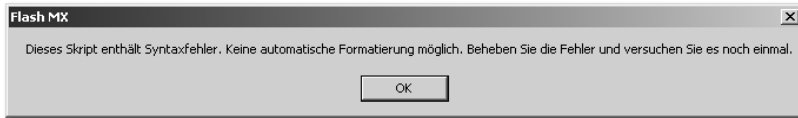


Abbildung 15.2 Syntaxfehler passen der automatischen Formatierung nicht.

Leider ist diese Fehlermeldung nicht sehr konkret. Mit der Schaltfläche **Syntax überprüfen** kommen wir ein bisschen weiter. Sie sagt nicht nur, dass es einen Syntaxfehler gibt, sondern gibt im **Ausgabe**-Fenster auch an, wo er sich befindet und welcher Art er ist.

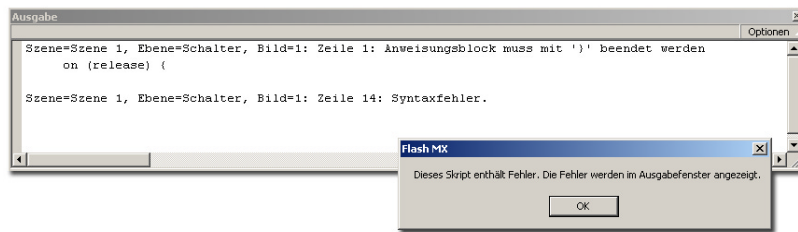


Abbildung 15.3 Die Fehlerart wird angezeigt.

Diese Anzeige im **Ausgabe**-Fenster würde auch erscheinen, wenn Sie den Film direkt testen würden. Allerdings erscheinen hier Syntaxfehler für den gesamten Film, bei Syntax überprüfen dagegen nur die des aktuellen Codes im **Aktionen**-Bedienfeld. In unserem Beispiel handelt es sich um eine vergessene geschweifte Klammer am Ende (siehe Abbildung 15.3).



Mit dem Befehl **Steuerung • Szene testen** können Sie auch nur eine Szene des Films testen.



Beim Testen eines Films finden Sie im Menü noch einige interessante Optionen. Für das Debugging sind vor allem die zwei Optionen **Objekte auflisten** und **Variablen auflisten** aus dem Menü **Fehlersuche** von Bedeutung. **Objekte auflisten** bietet im **Ausgabe**-Fenster einen Überblick über die Objekte des Films. Dies wird vor allem für umfangreichere Filme mit vielen Movieclips wichtig. **Variablen auflisten** bietet einen Überblick über die Variablen in dem Code.

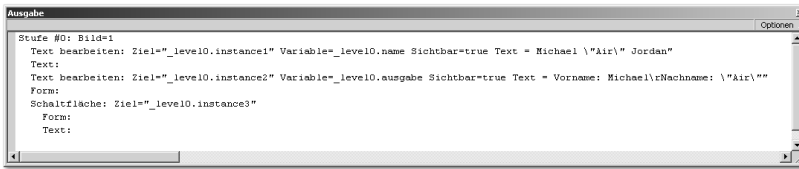


Abbildung 15.4 Objekte auflisten für unseren Beispielfilm

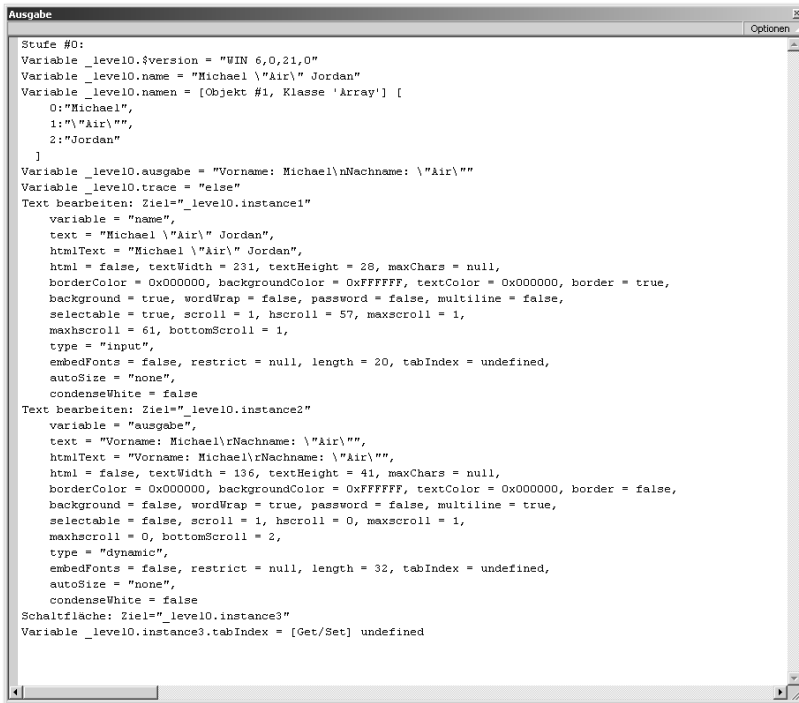


Abbildung 15.5 Variablen auflisten für unseren Beispielfilm

15.2 trace etc.

Standard-Syntaxfehler lassen sich also recht gut identifizieren. Ein wenig schwieriger wird es, wenn Sie beispielsweise für ein Objekt einfach einen falschen Namen verwendet haben oder irgendeine Methode falsch benannt haben. Der ActionScript-Interpreter gibt in solchen Fällen keine Fehlermeldung aus, aber die entsprechende Funktionalität steht einfach nicht zur Verfügung.

Einen solchen Fall haben wir, wenn wir aus **split_fehler fla** das Syntaxproblem mit der geschweiften Klammer entfernt haben. Die Syntax ist jetzt

einwandfrei, dennoch liefert das Skript eine falsche Ausgabe, wenn der Anwender mehr als Vor- und Nachnamen eingibt.



Abbildung 15.6 Das kann nicht stimmen.

Ein Blick in das Skript der Schaltfläche verrät, dass der Fehler vermutlich in der Fallunterscheidung liegt. Wenn Sie den Fehler nicht sofort bemerken, sollten Sie trace-Anweisungen einbauen, die an bestimmten Stellen aussagekräftige Ausgaben vornehmen. Wir haben zuerst einmal Ausgaben für die if- und die else-Anweisung eingefügt. So können wir prüfen, ob die if- oder die else-Bedingung zutrifft.

```
if (nemen.length>2) {  
    trace="if";  
    for (var i = 1; i<namen.length-1; i++) {  
        ausgabe += "Weiterer Vorname: "+namen[i] +"\n";  
    }  
    ausgabe += "Nachname: "+namen[namen.length-1];  
} else {  
    trace="else";  
    ausgabe += "Nachname: "+namen[1];  
}
```

Wenn Sie den Film testen, wird »else« ausgegeben, die else-Anweisung kommt also auch bei mehr als zwei Namen zur Anwendung. Dies legt den Verdacht nahe, dass entweder das Aufsplitten in mehrere Strings oder die Bedingung in der if-Anweisung einen Fehler enthält.

In unserem Beispiel ist es die Bedingung. Hier steht `nemen.length>2` statt `namen.length>2`.

`trace`-Anweisungen sollten Sie zur Fehlersuche sehr viel einsetzen. Sie sind ausgesprochen praktisch. Sie mit zu veröffentlichen, ist allerdings keine gute Idee, da sie relativ viel Performance kosten. Sie können deshalb in den **Einstellungen für Veröffentlichungen** (zu erreichen über **Datei • Einstellungen für Veröffentlichungen** und dort das Register **Flash**) das Veröffentlichen von `trace`-Anweisungen einfach ausschalten.

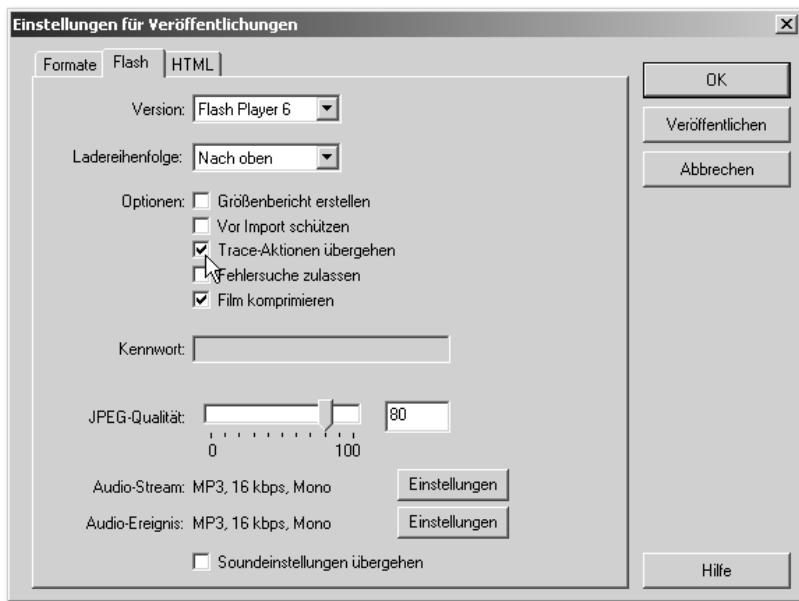


Abbildung 15.7 trace-Anweisungen aus dem Flash-Film entfernen

15.3 Debugger

Wenn weder der Syntax-Test noch eine `trace`-Anweisung in Verbindung mit einem Durchgehen des Codes die gewünschte Lösung bringt, ist es Zeit für den Debugger. Er verfolgt automatisch Variablen mit und erlaubt das schrittweise Ausführen von Code.

Bevor Sie aber mit dem Debugger starten, benötigen Sie so genannte Breakpoints im Code. Das sind Stellen, an denen der Debugger anhält, die aktuellen Variablenwerte preisgibt und Ihnen erlaubt, manuell einzugreifen.

Die Breakpoints setzen Sie entweder erst im Debugger oder schon im **Aktionen**-Bedienfeld. Dort verbergen sich die Breakpoints hinter der zweiten Schaltfläche rechts oben. Ein gesetzter Breakpoint wird mit einem roten Punkt angezeigt. Sie können ihn natürlich ebenfalls über das Menü hinter der Schaltfläche wieder entfernen oder sogar mit **Alle Breakpoints entfernen** allen auf einmal den Garaus machen.

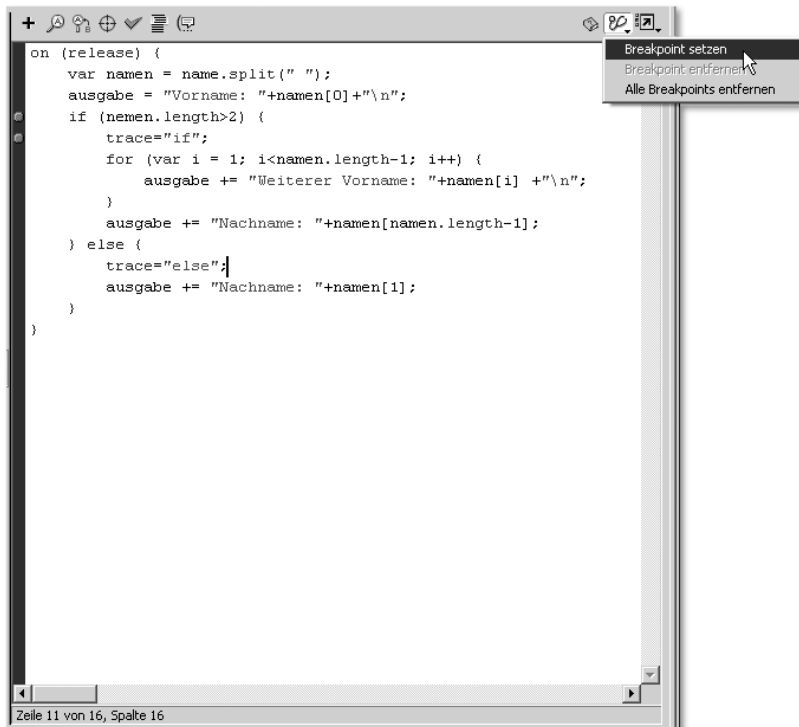


Abbildung 15.8 Breakpoints setzen

Wenn Sie die Breakpoints gesetzt haben, wechseln Sie mit dem Menübefehl **Steuerung • Fehlersuche** oder dem Tastenkürzel `[Strg] + [⇧] + [↩]` in den Debug-Modus.

In unserem Beispiel haben wir drei Breakpoints gesetzt, einen davon bei der if-Anweisung. Er soll prüfen, ob die if-Anweisung überhaupt getestet wird oder ob der Fehler im Code davor liegt. Die anderen beiden befinden

den sich bei den trace-Anweisungen im if- und im else-Block. Sie sollen feststellen, welche der beiden Anweisungen ausgeführt wird.

Sobald Sie im Debug-Modus sind, können Sie ganz normal mit Ihrem Flash-Film arbeiten und sehen seine Vorschau. Erst, wenn Code ausgeführt wird (in unserem Beispiel beim Anklicken der Schaltfläche), geschieht etwas im Debugger.

Im Beispiel sehen wir, dass der Debugger, sobald wir im Film einen Namen eingegeben und die Schaltfläche angeklickt haben, zum ersten Breakpoint springt. Er überprüft also die if-Anweisung korrekt.

Ein Blick in das Register Variablen auf der linken Seite zeigt auch, dass das Splitting der Variablen in ein Array hervorragend funktioniert hat.

Wenn wir dann mit den Schaltflächen oben im Debugger das Skript weiterlaufen lassen (Schaltfläche **Weiter**, ganz links), springt der Debugger gleich zum Breakpoint in der else-Anweisung. Das ist ein klares Anzeichen dafür, dass die if-Bedingung nicht zutrifft. Es wird also bestätigt, was wir schon wussten: Dort liegt der Fehler.

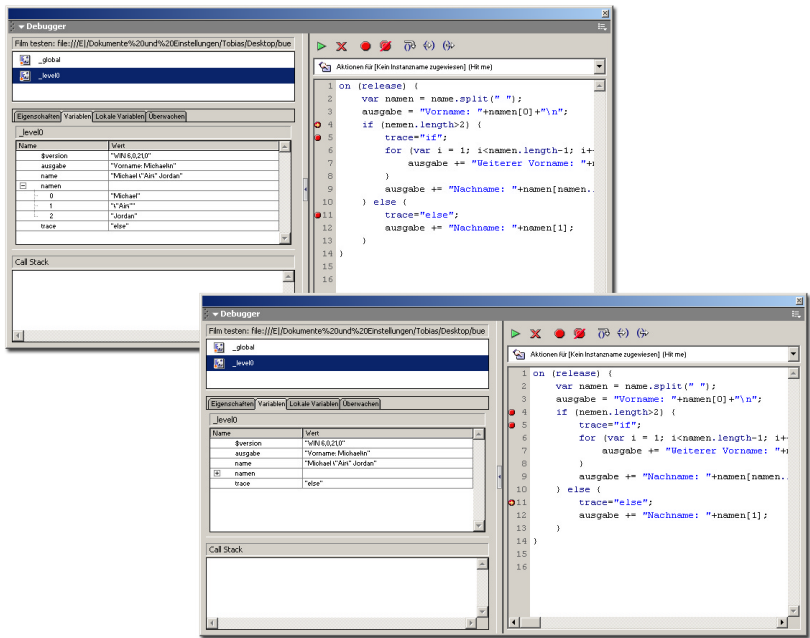


Abbildung 15.9 Zuerst wird die if-Anweisung geprüft, dann aber sofort zu else weitergesprungen.

15.4 Bandbreite

Ein letztes Werkzeug soll hier nicht unerwähnt bleiben, auch wenn es nur am Rande mit ActionScript zu tun hat. Der Bandbreiten-Profiler, den Sie im Testmodus (**Strg** + **↩**) über den Menübefehl **Ansicht • Bandbreiten-Profiler** aufrufen können, erlaubt eine Abschätzung der Performance des Flash-Films.

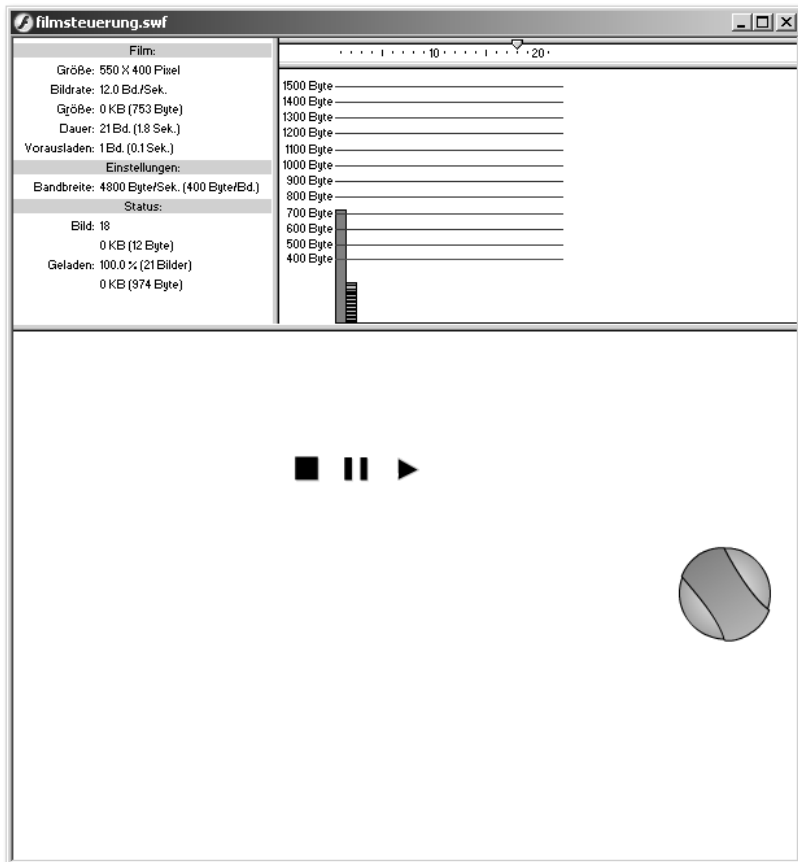


Abbildung 15.10 Bandbreiten-Profiler

Aus Sicht des ActionScript-Programmierers gibt es natürlich einiges, was er in Sachen Performance tun kann. Die wichtigsten Empfehlungen sind:

- ▶ Übergehen Sie trace-Anweisungen für den Export (siehe oben).
- ▶ Halten Sie oft benötigten Code nur einmal bereit, am besten in einer Funktion in der Hauptzeitleiste oder in einem leeren Movieclip.

- ▶ Exportieren Sie nur Symbole, die auch wirklich im ActionScript-Code eingesetzt werden.
- ▶ Verwenden Sie Symbole wieder und modifizieren Sie sie beispielsweise mit den Movieclip-Eigenschaften.
- ▶ Modifizieren Sie Sound mit den Sound-Eigenschaften und verwenden Sie ihn mehrmals, anstatt nur einen Sound einzusetzen.
- ▶ Verwenden Sie lokale Variablen, da diese nach dem Ausführen der Funktion aus dem Speicher gelöscht werden.
- ▶ Vermeiden Sie Schleifen, die nicht einfach beendet werden. Damit sind noch nicht einmal Endlosschleifen gemeint, sondern Schleifen, die einfach nur sehr lange auf ein Ereignis warten.
- ▶ Durchsuchen Sie große Arrays oder lange Strings nach Möglichkeit nicht und durchlaufen Sie sie nicht einzeln.

Macromedia hat sich natürlich auch Gedanken über die Performance gemacht. Unter http://www.macromedia.com/support/flash/ts/documents/optimizing_actionscript.htm finden Sie einen sehr interessanten Artikel zur Optimierung von ActionScript unter Performance-Gesichtspunkten. Die Empfehlung, manchmal die Flash 4-Syntax zu verwenden (z.B. `tell-Target`), weil dies schneller ist, wurde für Flash MX aufgehoben, da der ActionScript-Interpreter schneller ist. Auch sonst ist die Performance unserer Meinung nach meist kein ausreichender Grund, auf eine ältere Syntax zurückzugreifen.



16 Preloader

Warten macht unmoralisch.

– Friedrich Nietzsche

Einer der Vorteile von Flash ist die bekanntermaßen recht geringe Dateigröße, die eine schnelle Datenübertragung ermöglicht. Nicht allein deswegen hat sich Flash so rasend verbreitet, sind doch (mehr oder minder) plattformunabhängige Animationen möglich, ohne dass große Datenmengen über den Äther geschickt werden müssen.

Mit der zunehmenden Verbreitung von schnellen Internetzugängen für Heimanwender (erst Analog-Modems, dann ISDN, jetzt DSL) trauen sich Webdesigner immer häufiger, auch größere und komplexere Flash-Filme zu erstellen, die dann oft ein halbes Megabyte groß oder größer sind. Vorbei sind die Zeiten, in denen noch die goldene Regel galt, für eine Webseite inklusive aller eingebetteten Grafiken und eben auch Flash-Filme nicht viel mehr als 50 Kilobyte zu verwenden.

Insbesondere bei größeren Filmen ist natürlich die Gefahr, dass ein Benutzer schon während des Ladens wegklickt, nicht unerheblich. Einige Browser, beispielsweise der Netscape Navigator 4, zeigen in der Statusleiste den aktuellen Ladefortschritt an, oft auch bei größeren eingebetteten Dateien; beim Internet Explorer gibt es einen Ladebalken, der den (ungefähren) Ladestand der gesamten Seite, also HTML inklusive Grafiken und Flash-Filmen, wiedergibt. Besonders genau ist das nicht; außerdem sieht der Benutzer in der Regel nichts auf dem Bildschirm.

Ein möglicher Ausweg aus diesem Dilemma besteht in der Verwendung eines so genannten Preloaders – »Vorlader« ist der krampfhafteste Versuch einer Eindeutschung, wenngleich der englische Ausdruck recht gebräuchlich ist. Dabei handelt es sich um eine vorgeschaltete Filmsequenz, in der dem Benutzer der Ladestand mitgeteilt wird. Erst, nachdem der komplette Film geladen worden ist, wird zum ersten Bild des »eigentlichen« Films gesprungen; davor erscheint die Ladeanzeige.

Der Begriff Preloader ist bereits aus dem JavaScript-Teil bekannt; dort wurden in einem Beispiel Grafiken in den Zwischenspeicher des Browsers »vorgeladen«.



In diesem Kapitel wollen wir zwei Möglichkeiten für eine Ladeanzeige an einem kleinen Beispiel demonstrieren. Sie können dann diese Techniken

selbstredend auch auf komplexere Filme anwenden, denn dort erst macht ein Preloader wirklich Sinn!

16.1 Teile vorladen

Normalerweise geht das Flash-Plug-In wie folgt vor: Wenn ein Flash-Film in eine HTML-Seite eingebettet ist, wird er heruntergeladen und schon während des Herunterladens angezeigt, sofern möglich – also Streaming par excellence, wie beispielsweise bei Web-Radio. Es ist durchaus möglich, einen Film schon zu betrachten, bevor er ganz heruntergeladen worden ist; allerdings müssen Sie dann unter Umständen Wartezeiten in Kauf nehmen, wenn ein Bild angezeigt werden soll, das noch nicht vollständig übertragen worden ist.

ActionScript bietet seit Flash 5 eine Möglichkeit zu überprüfen, ob ein Bild bereits komplett geladen worden ist (Flash 4 bot übrigens ebenfalls diese Möglichkeit, wenn auch die Syntax der Sprache damals noch etwas anders war). Die Methode `ifFrameLoaded(X)` gibt an, ob das Bild mit der Nummer `X` oder mit dem Namen `X` bereits übertragen worden ist.

Der Einsatz ist etwas trickreich, und es liegt auch kein Tippfehler vor. Die Methode heißt explizit `ifFrameLoaded()` und nicht etwa `if FrameLoaded()`. Der Einsatz ähnelt dem einer `if`-Kontrollstruktur, die Unterschiede liegen aber – genauso wie der sprichwörtliche Teufel – im Detail. Hier sehen Sie eine Einsatzmöglichkeit:

```
ifFrameLoaded(X) {  
    gotoAndPlay(Y);  
}
```

Wenn also das Bild mit der Nummer (oder dem Namen) `X` schon geladen worden ist, wird das Bild mit der Nummer (oder dem Namen) `Y` angezeigt. Die Funktion – oder besser: das Konstrukt – `ifFrameLoaded()` verhält sich also ähnlich wie `if`; in einem nachfolgenden Skriptblock, umgeben von geschweiften Klammern `{ }`, stehen diejenigen Anweisungen, die ausgeführt sollen, wenn das angegebene Bild schon vorhanden ist. Aber noch einmal zur Warnung: Es handelt sich nicht um eine `if`-Anweisung. Ein `else` ist also nicht möglich:

```
ifFrameLoaded(X) {  
    gotoAndPlay(Y);  
} else {  
    gotoAndPlay(1); //zurück zum Start  
}
```


Obiger Code würde eine Fehlermeldung ausgeben: `else` ohne vorhergehendes `if` gefunden.

Eine Möglichkeit, einen Preloader zu erstellen, sieht also folgendermaßen aus:

- ▶ Im ersten Bild wird der Film zunächst angehalten.
- ▶ Per Intervall (das heißt: `setInterval()`) wird überprüft, ob ein bestimmtes, späteres Bild bereits geladen worden ist.
- ▶ Falls ja, wird der Film fortgesetzt, meistens natürlich bei dem Bild, das per `ifFrameLoaded()` überprüft wurde.

Um den Ladevorgang (und das Warten auf Bild x) nicht zu öde erscheinen zu lassen, können Sie natürlich noch eine kleine Animation einbauen:

- ▶ In den ersten vier Bildern (es können natürlich auch weniger oder mehr Bilder sein) bauen Sie eine kleine Animation ein.
- ▶ In Bild fünf wird überprüft, ob Bild zehn bereits geladen worden ist (auch hier gilt wieder: Die Zahlen fünf und zehn können Sie beliebig durch Ihre eigenen Anforderungen ersetzen).
- ▶ Falls das Bild geladen worden ist, wird die Filmausführung bei Bild zehn fortgesetzt.
- ▶ Ist Bild zehn noch nicht übertragen worden, springt der Flash-Interpreter wieder zu Bild eins – und damit dem Anfang der Animation – zurück.

Der entsprechende Code könnte etwa folgendermaßen aussehen:

```
ifFrameLoaded(10) {  
    gotoAndPlay(10);  
}  
gotoAndPlay(1);
```

Sie haben ja bereits zuvor gesehen, dass bei `ifFrameLoaded()` kein `else` möglich ist. Allerdings wird durch `gotoAndPlay()` die Ausführung des aktuellen Bildskripts sofort beendet, da das Bild verlassen wird. Aus diesem Grund wird in obigem Codeschnipsel der Aufruf `gotoAndPlay(1)` nur dann ausgeführt, wenn Bild zehn noch nicht geladen worden ist, der Codeblock innerhalb von `ifFrameLoaded(10) { ... }` also nicht zur Ausführung gekommen ist.

In unserem Beispiel möchten wir beide Techniken einmal vorstellen. Sie finden es auf der CD-ROM zum Buch unter dem Dateinamen **preloader1 fla**. Er besteht aus einer einzigen Ebene.

Im ersten Bild erstellen Sie ein Textfeld (es muss nicht dynamisch sein, statisch genügt), in das Sie den Text **Film wird geladen** eingeben. Bei den Bildern zwei bis vier erstellen Sie jeweils ein neues Schlüsselbild und fügen an derselben Stelle wieder ein Textfeld ein, mit den folgenden Texten:

- ▶ Bild zwei: Film wird geladen .
- ▶ Bild drei: Film wird geladen ..
- ▶ Bild vier: Film wird geladen ...

Hierdurch wird etwas Bewegung erzeugt: Hinter dem Text **Film wird geladen** werden drei Punkte animiert.

Über den Eigenschafteninspektor können Sie dafür sorgen, dass sich die vier Textfelder jeweils an derselben Stelle befinden, was Sie aus optischen Gründen unbedingt so einrichten sollten.



Wenn Sie das Textfeld per Copy-and-Paste jeweils in die vier Bilder einfügen, landet es meistens an derselben Stelle; nur das ursprüngliche, erste Textfeld (in Bild eins) müssen Sie per Eigenschafteninspektor an die richtige Position setzen.

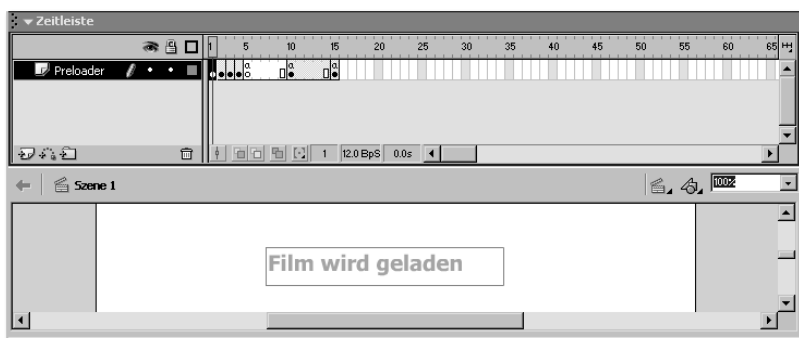


Abbildung 16.1 Das Textfeld im ersten Bild

Interessant wird es in Bild fünf, in dem Sie ein neues, leeres Schlüsselbild einfügen. Dort wird überprüft, ob Bild zehn schon geladen worden ist. Falls ja, wird genau zu diesem Bild gesprungen, andernfalls geht Flash zurück zu Bild eins. Das Listing kennen Sie bereits aus den obigen Ausführungen:

```

ifFrameLoaded(10) {
    gotoAndPlay(10);
}
gotoAndPlay(1);

```

Listing 16.1 Das Skript für Bild fünf

Bild zehn ist wieder ein neues, leeres Schlüsselbild. Damit es ein wenig dauert, dieses Bild zu laden, platzieren wir eine Grafik in diesem Bereich. Wir haben hierfür einen Screenshot von der Website von Galileo Press gewählt.

Screenshots sind meist ziemlich gut zu komprimieren, da nicht allzu viele Farben verwendet werden und auch große einfarbige Flächen vorhanden sind. Deswegen ist der Flash-Film am Ende auch nicht sonderlich umfangreich.



Abbildung 16.2 Die in Bild zehn eingefügte Bitmap

In diesem Bild wird ein weiteres Skript platziert. Dieses überprüft, ob Bild 15 bereits komplett übertragen worden ist. Falls ja, wird zu diesem Bild weitergesprungen, andernfalls bleibt der Film stehen. Diese Überprüfungsfunktion wird alle halbe Sekunde aufgerufen. Hier sehen Sie den zugehörigen Code:

```
stop();
var ID = setInterval(function(){
    ifFrameLoaded(15) {
        clearInterval(ID);
        gotoAndPlay(15);
    } }, 500);
```

Listing 16.2 Das Skript in Bild zehn

Obiger Code hält also zunächst einmal den Film an:

```
stop();
```

Danach wird per `setInterval()` folgende Funktion permanent ausgeführt:

```
ifFrameLoaded(15) {
    clearInterval(ID);
    gotoAndPlay(15);
}
```

Sobald also Bild 15 geladen worden ist, wird dieses Bild aufgerufen. Zuvor sollte jedoch noch die regelmäßige Ausführung der Funktion unterbunden werden, indem der Intervall-Aufruf mit `clearInterval()` gelöscht wird.



Die Definition einer Funktion direkt innerhalb einer Kontrollstruktur ist etwas, was mit JavaScript 1.0 und 1.1. noch nicht möglich war, weswegen Sie dies in HTML-Seiten eher selten sehen werden; in ActionScript-Programmen jedoch ist diese Vorgehensweise gang und gäbe.

Das einzige, was in der Applikation noch fehlt, ist Bild 15. Auch dort binden wir erneut eine Bitmap-Grafik ein, damit das Laden etwas dauert (siehe Abbildung 16.3).

Sie können jetzt den Film einmal aus Flash heraus testen (`[Strg] + [↵]` oder **Steuerung • Film testen**). Beim ersten Mal erscheint sofort Bild 15; wenn Sie den Test dann erneut starten (wieder `[Strg] + [↵]`), dauert es schon etwas länger. Bild zehn wird recht kurz angezeigt (je nach Rechnerausstattung), dann wird zu Bild 15 gesprungen.

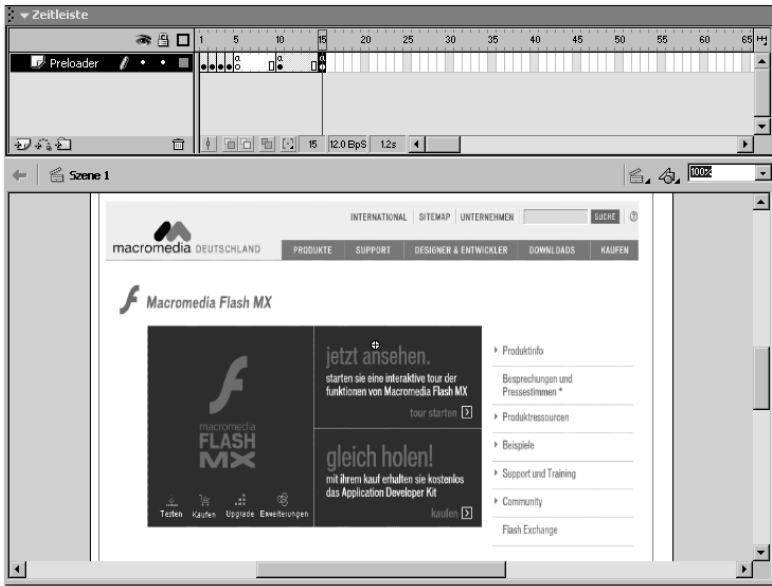


Abbildung 16.3 Bild fünfzehn – mit der Macromedia-Website

Im Testfenster können Sie die simulierte Übertragungsrate des Flash-Films über das Menü **Fehlersuche** einstellen. Die voreingestellten Werte – **14,4**, **28,8** und **56K** (also Kilobit pro Sekunde, Standard-Modemwerte) sind jedoch heutzutage nicht mehr ganz up to date. Während **56K** und zur Not auch **28,8** noch realistisch ist, leiden heutzutage nur noch Benutzer, die mit dem Mobiltelefon online gehen, unter 14,4 KBit – und diese Anwender deaktivieren Flash zumeist von vornherein. Über das Menükommando **Fehlersuche • Anpassen** können Sie jedoch auch andere Werte einstellen (siehe Abbildung 16.5).

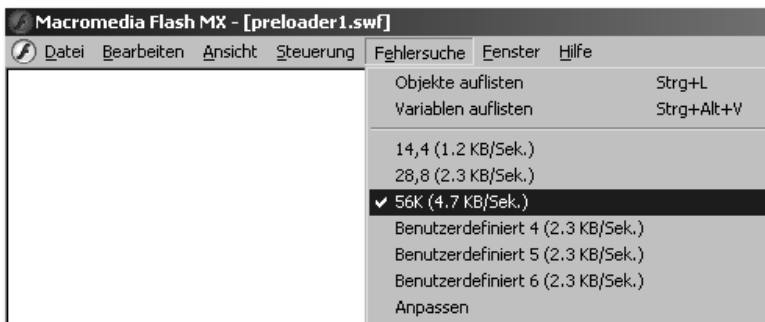


Abbildung 16.4 Die virtuelle Übertragungsrate kann eingestellt werden.

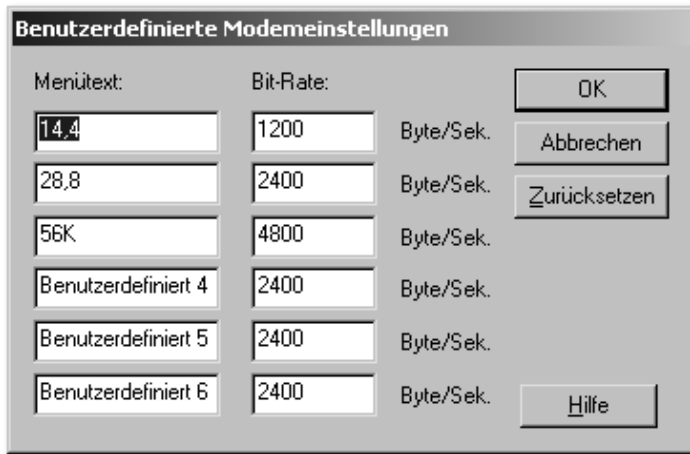


Abbildung 16.5 Die Voreinstellungen können auch geändert werden.

16.2 Gesamten Film vorladen

Am naheliegendsten ist es jedoch, einen Preloader für den gesamten Film zu erstellen. Bei der im Folgenden vorgestellten Methode bieten sich zudem mehr technische Möglichkeiten, was auch zu mehr Optionen zur grafischen Darstellung führt. Von der eigentlichen Programmierung her ist dieses Beispiel tendenziell weniger aufwändig als das aus dem vorhergehenden Abschnitt, allerdings ist die Erstellung des Flash-Films etwas schwieriger.



Wenn Sie die Schritte nicht selbst nachvollziehen möchten, finden Sie auf der CD-ROM zum Buch den hier vorgestellten Film unter dem Dateinamen **preloader1 fla** und können ihn direkt in Flash öffnen.

Der Film besteht aus zwei Ebenen. Die erste Ebene, bei uns mit **Preloader** betitelt, enthält die meisten Bestandteile des Films aus dem vorherigen Abschnitt:

- Bild eins enthält den Text `Film wird geladen`
- Bild zwei enthält den Text `Film wird geladen .`
- Bild drei enthält den Text `Film wird geladen ..`
- Bild vier enthält den Text `Film wird geladen ...`
- Bild fünf ist ein neues, leeres Schlüsselbild.
- Bild zehn ist ebenfalls ein neues, leeres Schlüsselbild und enthält einen Screenshot von der Galileo-Website.

Schon interessanter ist da die zweite Ebene, bei uns mit `Balken` bezeichnet. Das erste Bild (Schlüsselbild) enthält ein schwarz umrandetes Textfeld. Innerhalb dieses Felds gibt es noch einen etwas schmäleren und deutlich weniger hohen roten Balken. Diesen haben wir über sein Kontextmenü und den dortigen Eintrag **In Symbol konvertieren** in einen Movieclip umgewandelt. Dieser rote Balken wird später einmal unser Fortschrittsbalken werden.

Zusätzlich enthält das erste Bild noch ein dynamisches Textfeld mit Namen `ladestand`. Dieses Textfeld wird während des Ladens den Prozentwert enthalten, welcher Anteil des Films bereits übertragen worden ist. In Abbildung 16.6 sind diese Elemente dargestellt. Beachten Sie die gepunktete Umrandung des Textfelds für den Text `Film wird geladen`. Dieses Textfeld wurde bereits in der anderen Ebene, `Preloader`, erstellt.



Abbildung 16.6 Der Ladebalken und die Prozentanzeige

Beginnen wir nun mit der Programmierung. Am einfachsten ist der Code für das zehnte Bild in der Ebene `Preloader`. Wenn der Film dort einmal angekommen ist, muss er mit `stop()` angehalten werden.

Etwas komplizierter ist es in Bild fünf. Dort wird überprüft, ob der Film bereits komplett geladen worden ist oder nicht. Falls ja, wird zu Bild zehn

gesprungen (dem Screenshot); andernfalls beginnt der Film wieder vorne bei Bild eins.

Im vorherigen Abschnitt haben wir mit `ifFrameLoaded()` überprüft, ob ein bestimmtes Bild schon vollständig geladen worden ist. Diesmal wollen wir eine etwas fortgeschrittenere Methode einsetzen und nachsehen, ob der gesamte Film bereits übertragen wurde.

Flash kennt hierzu zwei Eigenschaften, die Sie abfragen können:

- ▶ `_framesloaded` gibt an, wie viele Bilder des Films bereits geladen worden sind.
- ▶ `_totalframes` gibt die Gesamtzahl aller Bilder im Film zurück.

Der folgende Code sorgt also für den gewünschten Effekt – falls noch nicht ganz geladen, zurück an den Anfang, ansonsten weiter zum Ende:

```
if (_framesloaded >= _totalframes) {  
    gotoAndPlay (10);  
} else {  
    gotoAndPlay(1);  
}
```

Listing 16.3 Das Skript für Bild fünf in der Ebene `Preloader`



Eigentlich wäre `(_framesloaded == _totalframes)` ausreichend; es ist aber schon vorgekommen, dass `_framesloaded` einen größeren Wert als `_totalframes` zurückgeliefert hat. Dieser Fehler lässt sich durch obigen Code bequem abfangen.

Alternativ dazu ist es auch möglich, anstelle der Anzahl der (übertragenen) Bilder die Anzahl der (übertragenen) Bytes im Film zu zählen. Dazu gibt es die folgenden zwei Methoden:

- ▶ `getBytesLoaded()` ermittelt die bereits übertragenen Bytes des Films.
- ▶ `getBytesTotal()` ermittelt die Gesamtgröße des Films in Bytes.

Sie können also alternativ für Bild fünf in der Ebene `Preloader` auch den folgenden Code verwenden:

```
if (getBytesLoaded() >= getBytesTotal()) {  
    gotoAndPlay (10);  
} else {  
    gotoAndPlay(1);  
}
```

Listing 16.4 Ein alternatives Listing für Bild fünf in der Ebene `Preloader`

Zu guter Letzt muss noch in der Ebene `Balken` der Code zur Aktualisierung des Fortschrittsbalkens erstellt werden. Um die Länge des Balkens zu berechnen, wird der Anteil der bisher übertragenen Daten benötigt. Dieser ließe sich nur recht ungenau und damit unzureichend aus `_frames-loaded` und `_totalframes` berechnen, denn nicht jedes Bild im Film ist gleich groß. Aus diesem Grund ist es genauer, auf `getBytesLoaded()` und `getBytesTotal()` zurückzugreifen.

Zunächst wird ein Prozentwert berechnet, also eine Zahl zwischen 0 und 1:

```
var stand = getBytesLoaded() / getBytesTotal();
```

Dieser Wert wird mit hundert multipliziert und mit `Math.round()` gerundet; dies führt zu einem ganzzahligen Wert zwischen 0 und 100. Dieser wird in das Textfeld `ladestand` geschrieben.

```
ladestand = ((stand<0.095) ? "0" : "") +  
            Math.round(100 * stand) + "%";
```

Die Anweisung `((stand<0.095) ? "0" : "")` liefert 0, falls weniger als (gerundet) zehn Prozent geladen worden sind; dadurch kann die Prozentanzeige auf jeden Fall zweistellig gemacht werden.



Im letzten Schritt werden noch die Länge und die Position des Balkens angepasst. Dazu sind die beiden folgenden Schritte nötig:

- Unser Balken hat eine Maximallänge von 187; dieser Wert muss mit dem Anteil geladener Daten multipliziert werden:

```
Balken._width = 187*stand;
```

- Die x-Position des Balkens ist abhängig von seiner Länge, denn es wird vom Mittelpunkt aus gemessen. Sie müssen also zum Startpunkt (bei uns: 164) die halbe Breite addieren, um den neuen Mittelpunkt zu erhalten:

```
Balken._x = 164 + (Balken._width / 2);
```

Nachfolgend sehen Sie noch einmal das komplette Listing:

```
var stand = getBytesLoaded() / getBytesTotal();  
ladestand = ((stand<0.095) ? "0" : "") +  
            Math.round(100 * stand) + "%";  
with (Balken) {  
    _width = Math.round(187 * stand);
```

```

_x = 164 + (_width / 2);
}

```

Listing 16.5 Das Listing für Bild eins in der Ebene Balken

Wenn Sie nun den Film speichern und testen (wie gehabt: `Strg` + `↵`), sehen Sie spätestens beim zweiten Test den Effekt: Der Fortschrittsbalken baut sich mehr oder minder fix auf; parallel dazu »bewegen« sich die Punkte im Textfeld. Sobald der Balken 100% erreicht hat, wird die Grafik angezeigt.

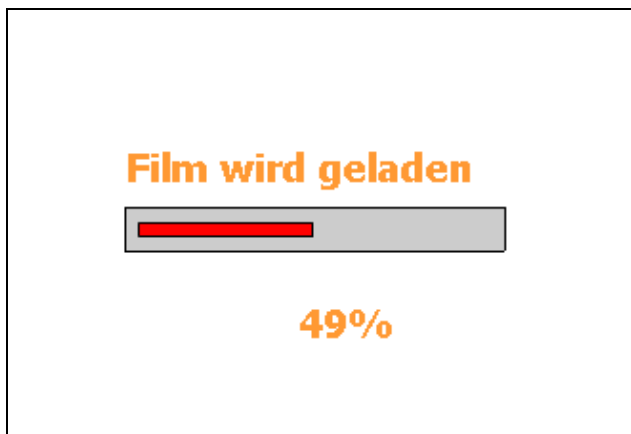


Abbildung 16.7 Die Fortschrittsanzeige in Aktion

Sie sehen also: Wenn Sie erst einmal den Bogen heraus haben, ist es ziemlich einfach, einen Preloader samt Fortschrittsanzeige zu realisieren. Insbesondere die Methode, die die bereits übertragene Datenmenge berücksichtigt, ermöglicht eine recht genaue grafische Darstellung. Somit versüßen Sie Ihrem Publikum die Wartezeit und wecken Lust auf mehr, wenn schon das Warten grafisch pfiffig umgesetzt worden ist.

17 **ActionScript mit JavaScript**

Wenn zwei Philosophen zusammentreffen, ist es am vernünftigsten, wenn sie zueinander bloß »Guten Morgen« sagen.

– Jean Paul Sartre

Der Titel dieses Buchs lautet »ActionScript mit JavaScript«. Wir wollen damit ausdrücken, dass sowohl ActionScript- als auch JavaScript-Kenntnisse vermittelt werden sollen. ActionScript selbst basiert ja – genauso wie JavaScript – auf ECMAScript, die Verwandtschaft ist also offensichtlich.

Warum gibt es ein Kapitel mit demselben Titel wie das Buch selbst? Nun, wir haben bis jetzt die beiden Technologien JavaScript und ActionScript parallel betrachtet. Wir haben immer wieder auf Parallelen und Unterschiede hingewiesen, die beiden Spracheinführungen waren allerdings autark. In diesem Kapitel dagegen schlagen wir die Brücke zwischen beiden Technologien. Es ist möglich, von Flash aus JavaScript-Kommandos aufzurufen – natürlich mithilfe von ActionScript. Ebenso ist es möglich, von JavaScript aus auf Teile des Flash-Films zuzugreifen.

Die Technologie, die das möglich macht, wurde ursprünglich von Netscape für seine Browser eingeführt. Sie heißt LiveConnect und ermöglicht in Netscape-Browsern bis zur Version 4 den Zugriff auf Java-Applets und per `<embed>` eingebettete Daten, sofern das zugehörige Plug-In dies unterstützt.

Der Microsoft Internet Explorer kennt kein Plug-In-Konzept. Die ActiveX-Technologie erlaubt es, Software in den Browser zu integrieren, die dann beispielsweise Flash-Filme abspielen kann. Auch auf diese ActiveX-Controls kann per Skriptcode zugegriffen werden, vorausgesetzt, dies wird unterstützt.

Im neuesten Netscape-Browser, dem Netscape 6, wurde der Skriptzugriff auf Plug-Ins geändert, die neue Methode ist leider nicht abwärtskompatibel. Es gibt kaum Plug-Ins, die dies bereits unterstützen. Das gilt auch für das Flash-Plug-In. Sie werden also die Beispiele in diesem Kapitel nicht mit dem Netscape 6 nachvollziehen können, sehr wohl aber mit älteren Netscape-Versionen (insbesondere mit dem immer noch relativ weit verbreiteten Netscape Navigator 4.x) und natürlich mit dem Marktführer, dem Internet Explorer.

Was das Betriebssystem angeht, ist festzuhalten, dass die Windows-Plattform (welche im Desktop-Bereich im Hinblick auf die Benutzerzahlen Spitzenreiter ist) die folgenden Techniken unterstützt, zumindest solange eine 32-Bit-Version (also Windows 95/98/ME sowie Windows NT/2000/XP) eingesetzt wird. Alte 16-Bit-Versionen, also insbesondere Windows 3.1x, unterstützen den Plug-In-Zugriff (beziehungsweise den Zugriff auf das ActiveX-Control) nicht. Beim Macintosh gibt es bei einigen Konfigurationen ebenfalls Schwierigkeiten: Beim Internet Explorer funktioniert der Zugriff überhaupt nicht, bei Netscape-Browsern auf alten Macintoshes (68K) auch nicht.

17.1 Von ActionScript nach JavaScript

Zunächst wollen wir zeigen, wie Sie von ActionScript aus JavaScript-Code ausführen können. Diese Ausführung findet innerhalb des aktuellen Browserfensters statt. Sie können also auch JavaScript-Funktionen zur Ausführung bringen, die sich auf der aktuellen HTML-Seite befinden. Flash (beziehungsweise ActionScript) bietet hierzu zwei Möglichkeiten, die wir im Folgenden vorstellen werden.

17.1.1 Die Funktion `fscommand()`

Bereits in der Flash-Version 3 wurde `fscommand()` eingeführt. Der ursprüngliche Sinn dieser Funktion lag in der Übergabe von Kommandos an den Flash-Player. So konnten (und können immer noch) der Player beendet oder externe Programme ausgeführt werden. Im Browser eingesetzt, ermöglicht `fscommand()` die Übergabe von Kommandos an die HTML-Seite. Der Einsatz ist recht komplex, funktioniert allerdings zuverlässig.

Die Funktion `fscommand()` erwartet zwei String-Parameter:

- ▶ das Kommando, das ausgeführt werden soll,
- ▶ optionale Parameter, die dem Kommando mitgegeben werden.

Kommandos im Flash-Player

Wenn Sie einen Flash-Film direkt in einem Flash-Player (Projektor) aufrufen (beispielsweise wenn Sie mit Flash und ActionScript eine Oberfläche für eine CD-ROM erstellen), stehen Ihnen sechs Kommandos zur Verfügung (sie wurden bereits in Kapitel 10, Fortgeschrittene Programmierung, kurz vorgestellt). Alle Kommandos und auch die möglichen Werte sind jeweils Strings. Sie können jedoch anstelle von beispielsweise `"true"`

auch den booleschen Wert `true` übergeben, der Flash-Player führt automatisch die Typkonvertierung durch.

Kommando	Beschreibung
<code>allowscale</code>	Gibt an, ob der Film auf die Playergröße skaliert werden darf (<code>true</code>) oder ob er immer in seiner tatsächlichen Größe dargestellt werden soll (<code>false</code>)
<code>exec</code>	Bringt ein externes Programm auf dem Client zur Ausführung
<code>fullscreen</code>	Schaltet bei <code>true</code> den Flash-Player in den Vollbildschirmmodus, bei <code>false</code> in den normalen Darstellungsmodus zurück
<code>quit</code>	Beendet den Flash-Player
<code>showmenu</code>	Gibt an, ob im Kontextmenü des Players alle Einträge sichtbar sein sollen (<code>true</code> , gilt auch im Webbrowser) oder ob lediglich Copyright-Informationen abgerufen werden können (<code>false</code> , einziger Menüeintrag ist dann z.B. Über Macromedia Flash Player 6...)
<code>trapallkeys</code>	Falls auf <code>true</code> gesetzt, werden alle Tastenkürzel abgefangen und an den Flash-Player gesendet. Dies macht sich insbesondere dadurch bemerkbar, dass sämtliche Tastenkürzel im Player nicht mehr funktionieren, aber mit ActionScript und <code>keyUp/keyDown</code> abgefangen werden können

Tabelle 17.1 Die Kommandos für `fscommand()` im Stand-alone-Player

Das Kommando "exec" wurde in späteren Versionen des Flash 5-Players und in Flash 6 abgeschafft, das heißt dort können keine externen Programme mehr ausgeführt werden. Zur Begründung wurden Sicherheitsbedenken vorgetragen – bösartige Flash-Filme könnten ja theoretisch auch böswillige Programme (beispielsweise Betriebssystemkommandos zum Löschen von Dateien) ausführen. Für die Produktion von CD-ROMs ist das natürlich ein schwerer Schlag; wenn hier externe Programme ausgeführt werden sollen, muss auf die Flash-Version 4 zurückgegangen werden.



Einen möglichen Ausweg gibt es jedoch: Erstellen Sie ein Textfeld und stellen Sie ein, dass dieses Feld HTML-Befehle darstellen kann. Dann schreiben Sie in dieses Textfeld einfach einen Link auf die Datei:



```
<a href="file:c:\winnt\system32\calc.exe">Windows-Taschenrechner (Win NT/2000)</a>
```

Zwar erscheint eine Warnmeldung (siehe Abbildung 17.1), das Programm wird allerdings dann auch ausgeführt.

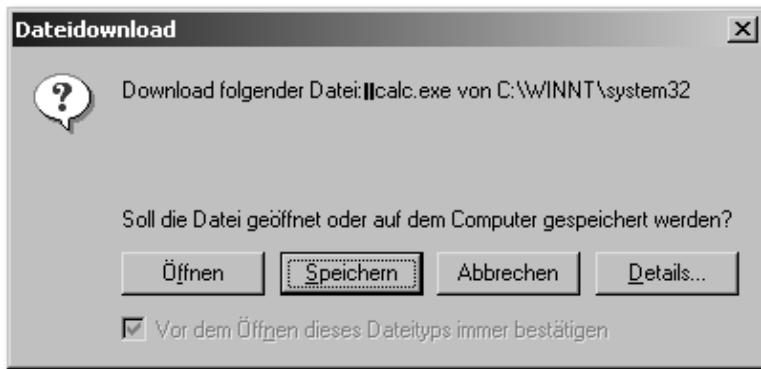


Abbildung 17.1 Die Warnmeldung beim Ausführen einer Datei

Wir erstellen nun einen Beispielfilm, der einige der Kommandos für `fscommand()` im Einsatz zeigt. Dazu wollen wir die `PushButton`-Komponente von Flash MX nutzen.



Wenn Sie eine ältere Flash-Version einsetzen, können Sie die Beispiele analog nutzen; setzen Sie Schaltflächen ein und verwenden Sie das `release`-Ereignis, um nach dem Klicken auf die Schaltfläche eine Aktion auszuführen.

Der Film besteht weitestgehend aus drei Instanzen einer `PushButton`-Komponente. Bezeichnung, Beschriftung und den zugeordneten Click-Handler entnehmen Sie bitte der folgenden Tabelle:

Bezeichnung	Beschriftung	Click-Handler
<code>push_fullscreen</code>	fullscreen	<code>push_fullscreen_click</code>
<code>push_exec</code>	exec	<code>push_exec_click</code>
<code>push_quit</code>	quit	<code>push_quit_click</code>

Tabelle 17.2 Bezeichnungen, Beschriftungen und Click-Handler

Das Resultat sehen Sie in Abbildung 17.2; auf der CD-ROM zum Buch finden Sie den Film unter dem Dateinamen **fscommand-standalone fla**.



Abbildung 17.2 Die Bühne des Beispielfilms

Der Rest spielt sich im Bild-Skript des Films für Bild 1 ab. Zunächst wird der Status des Vollbildmodus' abgespeichert. Beim Starten des Films ist dieser nicht aktiviert, Startwert ist also `false`:

```
var fullscreen = false;
```

Nun müssen die drei Click-Handler-Funktionen für die drei Schaltflächen bestimmt werden. Wir beginnen beim Vollbildmodus. Zunächst wird die Statusvariable geändert – aus `false` wird `true`, aus `true` wird `false`. Dies geht am schnellsten und einfachsten mit dem Negations-Operator (dem Ausrufezeichen `!`). Danach wird dieser Status der Funktion `fscommand()` übergeben:

```
function push_fullscreen_click() {  
    fullscreen = !fullscreen;  
    fscommand("fullscreen", fullscreen);  
}
```

Als Nächstes widmen wir uns der Schaltfläche **exec**; wie wir bereits erwähnt haben, funktioniert diese bei den aktuellen Player-Versionen nicht, ältere Versionen dagegen unterstützen noch das Kommando "exec".

```
function push_exec_click() {  
    fscommand("exec", "start\\tfsccommand.txt");  
}
```



Beachten Sie das Kommando – "start\tfsccommand.txt". Das \t ist ein Tabulatorzeichen, das Kommando lautet also "start fscommand.txt". Wenn Sie dies unter Windows ausführen, wird die Datei **fscommand.txt** (eine einfache Textdatei) mit dem ihr zugeordneten Programm geöffnet. Bei einer Standardinstallation ist das der einfache Texteditor von Windows, Notepad.

Zu guter Letzt fehlt noch das Kommando zum Beenden des Flash-Players:

```
function push_quit_click() {  
    fscommand("quit");  
}
```



Selbstverständlich ist es **nicht** möglich, damit etwa den Webbrowser zu schließen; wie dies bewerkstelligt werden kann, erfahren Sie später.

Nachfolgend noch einmal der vollständige Code:

```
var fullscreen = false;  
  
function push_fullscreen_click() {  
    fullscreen = !fullscreen;  
    fscommand("fullscreen", fullscreen);  
}  
  
function push_exec_click() {  
    fscommand("exec", "start\tfsccommand.txt");  
}  
  
function push_quit_click() {  
    fscommand("quit");  
}
```

Listing 171 Das Bildskript für das erste Bild im Film

Nun muss nur noch der Film als Stand-alone-Player (Projektor) exportiert werden. Öffnen Sie dazu das Dialogfenster **Einstellungen für Veröffentlichungen**, entweder über den Menübefehl **Datei • Einstellungen für Veröffentlichungen** oder über das Tastenkürzel **[Strg] + [⇧] + [F12]**. Aktivieren Sie dort die Checkbox **Windows-Projektor (.exe)** oder die Checkbox **Macintosh-Projektor**, je nachdem, für welche Plattformen Sie entwickeln. Die restlichen Exportformate benötigen Sie für dieses Beispiel nicht. Durch das Anklicken der Schaltfläche **Veröffentlichen** wird die ausführbare Datei generiert.

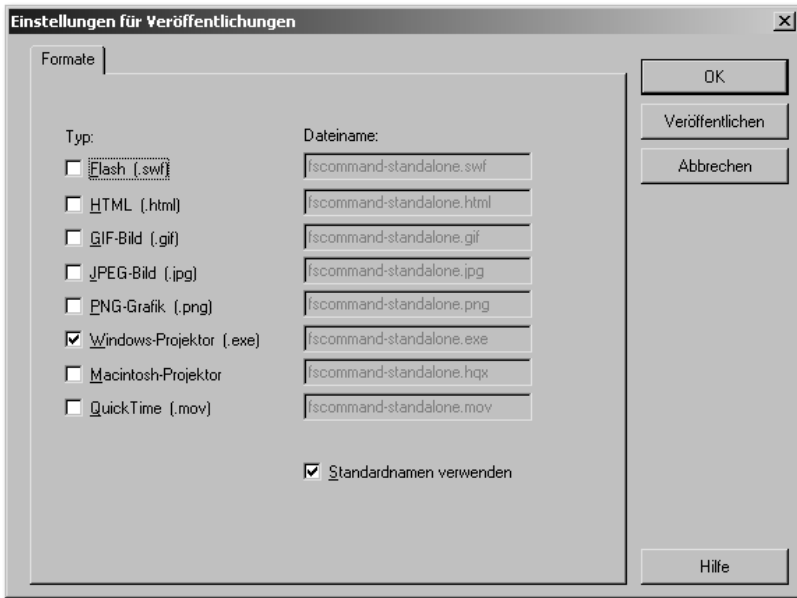


Abbildung 17.3 Die Veröffentlichungseinstellungen

Wenn Sie die Datei starten, sehen Sie Ihren Film als allein stehende Applikation. Mit der Schaltfläche **fullscreen** können Sie in den Vollbildmodus umschalten; beachten Sie hierbei, wie die Größe der Schaltflächen und ihrer Beschriftungen dynamisch angepasst wird. Wenn Sie dieses Verhalten ändern möchten, müssen Sie `fscommand("allowscale", "false")` ausführen.

Wie bereits zuvor erwähnt wurde: `fscommand("allowscale", false)` funktioniert auch, die Typumwandlung wird automatisch durchgeführt.



Abbildung 17.4 Der Film im Projektor

Kommandos im Webbrowser

Für die Erstellung von CD-ROM-Oberflächen wird bei Macromedia-Produkten weiterhin hauptsächlich auf Director gesetzt, während die Hauptdomäne von Flash ganz klar die Produktion für das World Wide Web ist (zumindest äußern sich Macromedia-Produktmanager immer so, wenn sie auf die Zukunft von Director angesprochen werden). Daher liegt es nahe, `fscommand()` auch im World Wide Web einzusetzen.

Damit dies möglich ist, muss zunächst der Film umbenannt werden. Wir haben als Namen **fscommand.fl** gewählt und den Film auch unter diesem Namen auf der CD-ROM zum Buch abgelegt (er ist allerdings identisch mit dem Film **fscommand-standalone.fl**).

Laden Sie diesen Film in Flash, rufen Sie die Veröffentlichungseinstellungen von Flash auf (`(Strg) + (U) + (F12)`) und selektieren Sie die Typen **Flash (.swf)** und **HTML (.html)**. Im Register HTML stellen Sie dann unter **Vorlage** den Wert **Flash mit FSCommand** ein. Durch diese Einstellung wird der HTML-Ausgabe zusätzlicher Code hinzugefügt, der die Ausführung von `fscommand()` und den Zugriff auf JavaScript-Code innerhalb der HTML-Seite ermöglicht.

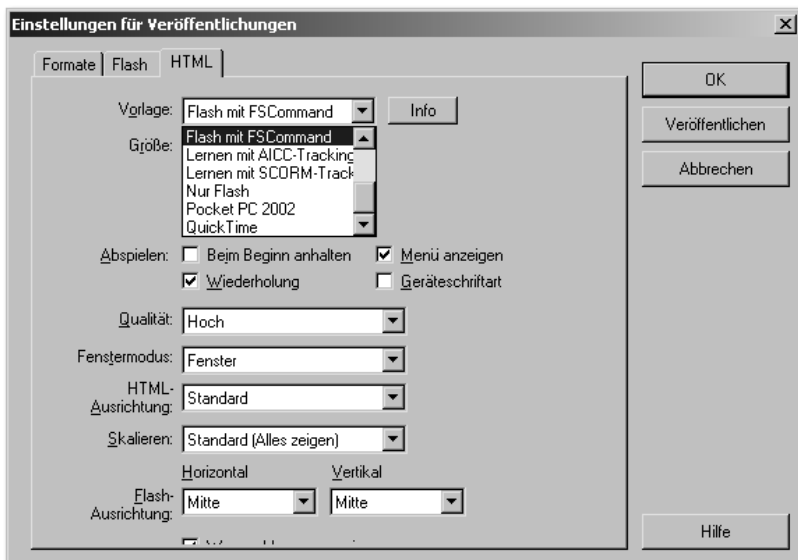


Abbildung 17.5 Als Vorlage muss **Flash mit FSCommand** gewählt werden.

Veröffentlichen Sie den Film und werfen Sie einen Blick auf die resultierende HTML-Seite:

```
<HTML>
<HEAD>
<meta http-equiv=Content-Type content="text/html; charset=ISO-
8859-1">
<TITLE>fscommand</TITLE>
</HEAD>
<BODY bgcolor="#FFFFFF">
<SCRIPT LANGUAGE=JavaScript>
<!--
var InternetExplorer = navigator.appName.indexOf("Microsoft") !=
-1;
// Handle all the FSCommand messages in a Flash movie
function fscommand_DoFSCommand(command, args) {
    var fscommandObj = InternetExplorer ? fscommand : docu-
ment.fscommand;
    //
    // Place your code here ...
    //
}
// Hook for Internet Explorer
if (navigator.appName && navigator.appName.indexOf("Microsoft")
!= -1 &&
    navigator.userAgent.indexOf("Windows") != -1 && naviga-
tor.userAgent.indexOf("Windows 3.1") == -1) {
    document.write('<SCRIPT LANGUAGE=VBScript> \n');
    document.write('on error resume next \n');
    document.write('Sub fscommand_FSCommand(ByVal command,
ByVal args)\n');
    document.write('    call fscommand_DoFSCommand(command,
args)\n');
    document.write('end sub\n');
    document.write('</SCRIPT> \n');
}
//-->
</SCRIPT>
<!-- URLs used in the movie-->
<!-- text used in the movie-->
<!--fscommand()--><OBJECT classid="clsid:D27CDB6E-AE6D-11cf-
```

```

96B8-444553540000"
  codebase="http://download.macromedia.com/pub/shockwave/cabs/
flash/swflash.cab#version=6,0,0,0"
  ID="fscommand" WIDTH="125" HEIGHT="175" ALIGN="">
  <PARAM NAME=movie VALUE="fscommand.swf"> <PARAM NAME=quality
VALUE=high> <PARAM NAME=bgcolor VALUE=#FFFFFF> <EMBED
src="fscommand.swf" quality=high bgcolor=#FFFFFF WIDTH="125"
HEIGHT="175" swLiveConnect=true ID="fscommand" NAME="fscommand"
ALIGN=""
  TYPE="application/x-shockwave-flash" PLUGINSOURCE="http://
www.macromedia.com/go/getflashplayer"></EMBED>
</OBJECT>
</BODY>
</HTML>

```

Listing 17.2 Die von Flash erzeugte HTML-Seite

Wir sezieren im Folgenden den Code und analysieren ihn Stück für Stück.

Der Beginn ist noch recht harmlos – ein HTML-Dokument wird gestartet, als Seitentitel wird der Dateiname der Flash-Datei ohne die Endung **.fla** verwendet:

```

<HTML>
<HEAD>
<meta http-equiv=Content-Type content="text/html; charset=ISO-
8859-1">
<TITLE>fscommand</TITLE>
</HEAD>
<BODY bgcolor="#FFFFFF">
<SCRIPT LANGUAGE=JavaScript>
<!--

```

Nun beginnt der Skript-Teil. Zunächst wird eine boolesche Variable `InternetExplorer` definiert, die ermittelt, ob der Browser ein Internet Explorer ist. Dazu wird `navigator.appName` ausgelesen; wie Sie wissen, wird das in der Praxis üblicherweise mit `navigator.userAgent` erledigt.

```
var InternetExplorer = navigator.appName.indexOf("Microsoft") != -1;
```

Es folgt eine Funktion `fscommand_DoFSCommand()`; wenn Ihr Film den Namen **xyz.fl**a hat, heißt sie `xyz_DoFSCommand()`. In dieser Funktion landen alle `fscommand()`-Aufrufe, wie Sie später sehen werden. Die Funktion ist leer bis auf eine Anweisung, die in der Variablen `fscommandObj` eine

Referenz auf den in der HTML-Seite eingebauten Flash-Film enthält. In Abschnitt o, wo es um den Zugriff auf Flash von JavaScript aus geht, erfahren Sie die Hintergründe, wie und warum das funktioniert.

```
// Handle all the FSCommand messages in a Flash movie
function fscommand_DoFSCommand(command, args) {
    var fscommandObj = InternetExplorer ? fscommand : document.fscommand;
    //
    // Place your code here ...
    //
}
```

Nun ist es leider so, dass diese Methode nur beim Netscape-Browser funktioniert; von einem Plug-In kann dank LiveConnect direkt auf JavaScript-Funktionen innerhalb der HTML-Seite zugegriffen werden. Beim Microsoft Internet Explorer ist das nicht möglich. Allerdings gibt es hier einen Trick: Von einem ActiveX-Control aus (wie eben vom Flash-Player aus) können clientseitige VBScript-Funktionen ausgeführt werden.

Bei VBScript handelt es sich um eine von Microsoft entwickelte, von der Programmiersprache BASIC abgeleitete Skriptsprache, die normalerweise serverseitig eingesetzt wird, im Zusammenhang mit ASP (Active Server Pages). Es ist jedoch auch möglich, die Sprache clientseitig als eine Art Konkurrenz zu JavaScript einzusetzen. In der Praxis kommt das eher selten vor, weil der Netscape Navigator kein VBScript unterstützt. In diesem Fall jedoch ist das erwünscht, denn nur der Internet Explorer wird den Code ausführen.



Erinnern Sie sich noch an die Variable `InternetExplorer?` Diese wird jetzt verwendet. Wenn ein Internet Explorer vorliegt und dieser nicht auf einer der zuvor als »nicht funktionsfähig« vorgestellten Plattformen (16-Bit-Windows, Macintosh) läuft, wird eine VBScript-Funktion erstellt, die wiederum die zuvor definierte JavaScript-Funktion aufruft:

```
// Hook for Internet Explorer
if (navigator.appName && navigator.appName.indexOf("Microsoft")
    != -1 &&
    navigator.userAgent.indexOf("Windows") != -1 && navigator.userAgent.indexOf("Windows 3.1") == -1) {
    document.write('<SCRIPT LANGUAGE=VBScript> \n');
    document.write('on error resume next \n');
    document.write('Sub fscommand_FSCommand(ByVal command,
```

```

ByVal args)\n');
    document.write('  call fscommand_DoFSCommand(command,
args)\n');
    document.write('end sub\n');
    document.write('</SCRIPT> \n');
}

```

Das Kernstück der VBScript-Funktion `fscommand_FSCommand()` ist das folgende Kommando:

```
call fscommand_DoFSCommand(command, args)
```

Dadurch wird die JavaScript-Funktion `fscommand_DoFSCommand()` aufgerufen und somit die Klippe, dass von ActiveX-Controls aus keine JavaScript-Funktionen direkt aufgerufen werden können, umschifft.

Das war auch schon der gesamte Skript-Code. Es folgt noch der Code zur Ausgabe des Flash-Films im Internet Explorer und im Netscape 6 per `<object>`-Element:

```

//-->
</SCRIPT>
<!-- URLs used in the movie-->
<!-- text used in the movie-->
<!--fscommand()--><OBJECT classid="clsid:D27CDB6E-AE6D-11cf-
96B8-444553540000"
    codebase="http://download.macromedia.com/pub/shockwave/cabs/
flash/swflash.cab#version=6,0,0,0"
    ID="fscommand" WIDTH="125" HEIGHT="175" ALIGN="">
    <PARAM NAME=movie VALUE="fscommand.swf"> <PARAM NAME=quality
VALUE=high> <PARAM NAME=bgcolor VALUE=#FFFFFF>

```

Interessant ist dagegen die Einbindung des Flash-Films für den Netscape Navigator in der Version 4 und früher. Dort wurde `<object>` ja nur unzureichend unterstützt, weswegen hier `<embed>` zum Einsatz kommen muss:

```

<EMBED src="fscommand.swf" quality=high bgcolor=#FFFFFF
WIDTH="125" HEIGHT="175" swLiveConnect=true ID="fscommand"
NAME="fscommand" ALIGN=""
    TYPE="application/x-shockwave-flash" PLUGINSOURCE="http://
www.macromedia.com/go/getflashplayer"></EMBED>

```

Im `<embed>`-Element ist der Parameter `swLiveConnect="true"` gesetzt (oder genauer gesagt, `swLiveConnect=true`, also ganz weit weg von

XHTML). Diese Angabe ermöglicht es dem Flash-Film erst, auf JavaScript-Code zuzugreifen; andernfalls wäre das nicht möglich. Wenn Sie also die resultierende HTML-Seite von Hand erstellen sollten (oder zumindest den Flash-Film von Hand einbinden), vergessen Sie diesen Parameter nicht, sonst rührt sich im Netscape Navigator nichts.

```
</OBJECT>
</BODY>
</HTML>
```

Damit endet die Seite, es ist an der Zeit, dass wir den Code um eigene Funktionalitäten erweitern.

Zuvor noch ein gut gemeinter Rat: Achten Sie penibel darauf, im Dateinamen der **.fla**-Datei keine Sonderzeichen zu verwenden. Der Grund: Beim Veröffentlichen des Films wird der Dateiname vor **.fla** wie gesehen als Namensteil für gewisse Funktionen verwendet. An dieser Stelle sind in JavaScript jedoch keine Sonderzeichen erlaubt, was aber den Veröffentlichen-Mechanismus wenig stört. Hätten wir den vorherigen Film, **fscommand-standalone.flas**, veröffentlicht, hätte das unter anderem zu einer Funktion `fscommand-standalone_DoFSCommand()` geführt – und zu einer JavaScript-Fehlermeldung, da das Minuszeichen nicht erlaubt ist.



Aber zurück zum Code. Wie Sie sicherlich schon vermuten, werden in der Funktion `xxx_DoFSCommand()` (bei uns: `fscommand_DoFSCommand()`) Aufrufe von `fscommand()` innerhalb von ActionScript ausgeführt. Beachten Sie bitte: Die JavaScript-Funktion erwartet zwei Parameter, also genauso viele wie die ActionScript-Funktion `fscommand()`.

Um dies zu testen, fügen wir in die Funktion eine Anweisung zur Ausgabe der übergebenen Parameter ein:

```
function fscommand_DoFSCommand(command, args) {
    alert("Kommando: " + command + "\nArgument" + args);
}
```

Sie finden eine angepasste HTML-Datei mit diesem Code unter dem Dateinamen **fscommand-1.html**.



Testen Sie diesen Code – laden Sie die HTML-Seite in Ihren Webbrowser und klicken Sie eine der Schaltflächen an. Sie erhalten ein modales Warnfenster mit den entsprechenden Angaben, also dem Kommando und – sofern vorhanden – dem Argument.

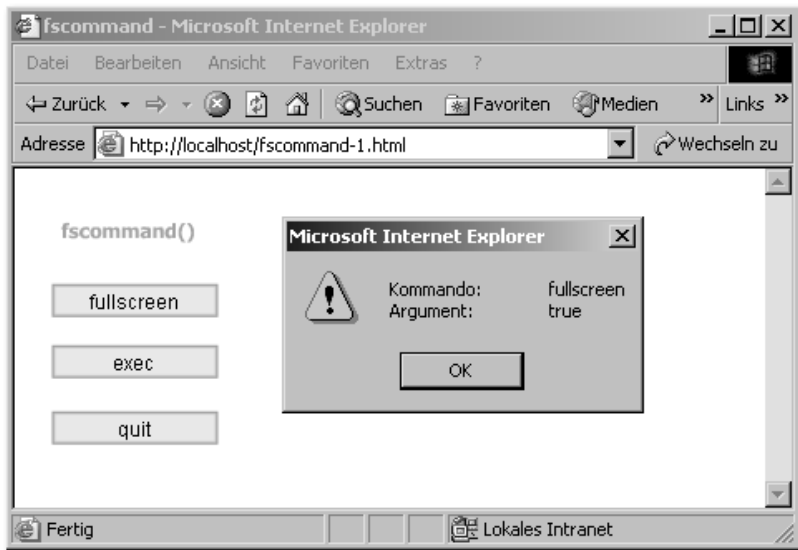


Abbildung 17.6 Kommando und Argument beim Anklicken von **fullscreen**

Der weitere Ablauf ist nun eher einfach: Der erste Parameter, also das eigentliche Kommando, wird ausgelesen und in Abhängigkeit davon eine Aktion gestartet. Dabei versuchen wir, die drei gewünschten Aktionen – Vollbildmodus, Ausführen einer Datei und Schließen des Browsers – durch JavaScript zu emulieren.

Beginnen wollen hier hierbei mit dem Vollbildmodus. Die JavaScript-Bordmittel erlauben leider keine browserübergreifende Vollbildansicht, weswegen wir uns mit einem kleinen Kniff behelfen müssen, den Sie bereits aus dem ersten Teil dieses Buches kennen. Dazu wird die maximale Breite und Höhe des verfügbaren Bildschirmbereichs ermittelt:

```
var breite = screen.availWidth;
var hoehe = screen.availHeight;
```

Die Dimensionen des Fensters werden dann auf diesen Wert gesetzt und das Fenster wird die linke obere Ecke geschoben:

```
window.resizeTo(breite, hoehe);
window.moveTo(0, 0);
```

Dies ist jedoch nur der Weg, ein Fenster zu maximieren. Wenn es dagegen wieder in den Normalmodus zurückgeschaltet werden soll, müssen die ursprünglichen Maße wiederhergestellt werden. Deswegen müssen

Sie beim ersten Laden des Dokuments diese Originalmaße abspeichern und gegebenenfalls wieder zuweisen (dann nämlich, wenn `fscommand("fullscreen", "false")` aufgerufen wird).

Je nachdem, welcher Browser eingesetzt wird, geht das entweder über das `window`-Objekt (und dann die Eigenschaften `outerWidth` und `outerHeight`) oder über `document.body` (und die Eigenschaften `clientWidth` und `clientHeight`).

Zusätzlich sollten Sie auch noch die Koordinaten der linken oberen Ecke des Browserfensters ermitteln, um das Fenster später wieder an seine ursprüngliche Position zurück verschieben zu können. Beim Netscape-Browser geht das via `window.screenX` und `window.screenY`, beim Internet Explorer heißen diese Eigenschaften `window.screenLeft` und `window.screenTop`; es gibt sie dort allerdings erst ab Version 5 und offensichtlich nur in der Windows-Variante des Browsers.

```
var originalbreite, originalhoehe, x, y;
if (window.outerWidth) {
    originalbreite = window.outerWidth;
    originalhoehe = window.outerHeight;
    x = window.screenX;
    y = window.screenY;
} else if (document.body) {
    originalbreite = document.body.clientWidth;
    originalhoehe = document.body.clientHeight;
    x = window.screenLeft;
    y = window.screenTop;
}
```

Dieser Code muss **außerhalb** der Funktion `fscommand_DoFSCommand()` stehen, damit er nur einmal beim Laden der Seite ausgeführt wird!



Der zugehörige Code innerhalb der Funktion `fscommand_DoFSCommand()` sieht dann folgendermaßen aus:

```
if (command == "fullscreen") {
    if (args == "true") {
        var breite = screen.availWidth;
        var hoehe = screen.availHeight;
        if (breite && hoehe) {
            window.resizeTo(breite, hoehe);
            window.moveTo(0, 0);
        }
    }
}
```

```

    } else {
        if (originalbreite && originalhoehe)
            window.resizeTo(originalbreite, originalhoehe);
        if (x && y)
            window.moveTo(x, y);
    }
}

```

Mit diesem Code ist der Fullscreen-Modus von Flash auch im Webbrowser implementiert.



Die Bestimmung von Browserbreite und -höhe muss nach dem `<body>`-Tag vorgenommen werden, andernfalls liefert der JavaScript-Interpreter falsche Werte zurück. Beim Internet Explorer übrigens scheint nur die Höhe des Browsers ohne Leisten ermittelt zu werden; das Browserfenster ist also nach dem Umschalten in den Vollbild-Modus und zurück niedriger als zuvor.

Die nächste Schaltfläche ist **exec**, zur Ausführung einer Datei. Auch dies kann in JavaScript zumindest ungefähr nachgebaut werden, indem einfach ein neues Fenster mit der angegebenen Datei oder dem Programm geöffnet wird. Entweder der Internet Explorer führt das Programm sofort aus (dies tut er beispielsweise in vielen Versionen, wenn die HTML-Datei und auch das Programm jeweils lokal aufgerufen werden), oder es erscheint eine Warnmeldung wie in Abbildung 171, der Benutzer kann aber das Programm ausführen oder die Datei ansehen.

Bleibt ein einziges Problem, das es noch zu überwinden gilt: Unter Umständen wurde dem Dateinamen noch ein "start\t" vorangestellt – unter Windows ist das notwendig, beim Macintosh nicht. Dieses "start\t" sollte natürlich herausgefiltert werden. Ansonsten bringt der Code nicht viel Aufsehen Erregendes; es wird ein neues Fenster geöffnet und darin die Datei angezeigt oder eben ausgeführt, je nach Gusto und Konfiguration des Browsers:

```

if (command == "exec") {
    if (args.indexOf("start\t") == 0)
        args = args.substring(6); //alles nach "start\t"
    window.open(args, "exec");
}

```

Wenn Sie diese Funktion der `fscommand_DoFSCommand()` hinzufügen und die Seite im Webbrowser öffnen, können Sie den Effekt bewundern. Die Datei wird in einem neuen Browserfenster geöffnet. Da es sich hierbei um eine Text-Datei gehandelt hat, wird sie direkt angezeigt, bei anderen Dateitypen kommt unter Umständen eine Warnmeldung oder (bei Programmen) die Datei wird direkt ausgeführt. In Abbildung 17.7 sehen Sie das Ergebnis für unseren Beispielfilm.

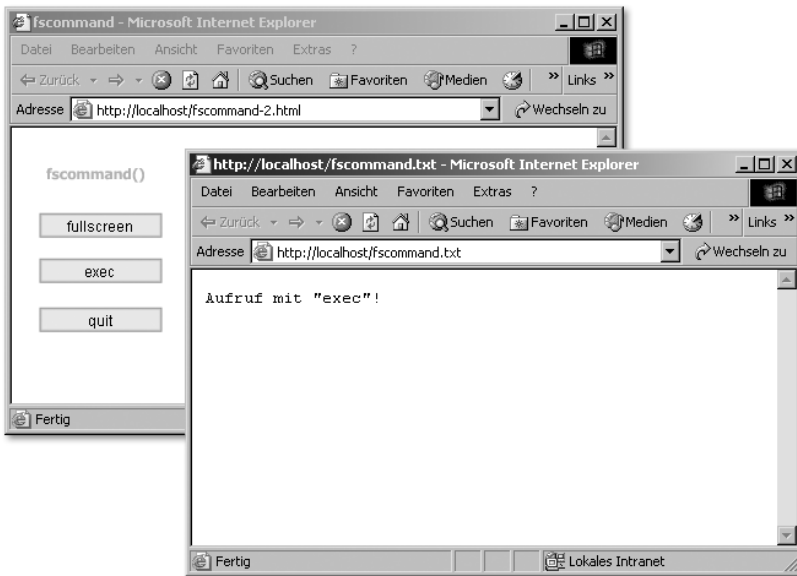


Abbildung 17.7 Die Datei wird in einem neuen Fenster geöffnet.

Beim Netscape Navigator funktioniert dies leider nicht.



Was nun noch fehlt, ist das Schließen des Players, in diesem Fall gleichbedeutend mit dem Schließen des Webbrowsers. Wie Sie bereits wissen, hilft hier die JavaScript-Methode `window.close()` weiter. Bauen Sie also diesen Code in die Funktion `fscommand_DoFSCommand()` ein:

```
if (command == "quit")
    window.close();
```

Allerdings erscheint dann beim Anklicken der Schaltfläche **quit** eine Warnmeldung des Browsers (siehe Abbildung 17.8):



Abbildung 17.8 Die Warnmeldung des Browsers

Diese erscheint nicht ohne Grund – das wäre ja noch schöner, wenn ein JavaScript-Programm willkürlich Browserfenster schließen könnte. Eine Ausnahme gibt es jedoch zu dieser Regel: Mit JavaScript geöffnete Fenster können auch wieder per JavaScript geschlossen werden. Eine Möglichkeit, dies auszunutzen (und die Warnmeldung aus Abbildung 17.5 zu vermeiden) besteht darin, gleich auf der Startseite der Applikation ein neues Fenster zu öffnen:

```
<html>
<head>
<title>ActionScript mit JavaScript</title>
<script type="text/javascript"><!--
window.open("fscommand-2.html", "Flash",
            "width=150,height=225");
//--></script>
</head>
<body>
<noscript>
Diese Anwendung funktioniert leider nur mit JavaScript ...
</noscript>
</body>
</html>
```

Listing 17.3 Der Flash-Film wird in einem neuen Fenster angezeigt.

Sie finden diese HTML-Datei unter dem Dateinamen **start.html** auf der CD-ROM zum Buch. Wenn Sie diese Datei aufrufen, wird der Flash-Film in einem neuen Fenster geöffnet. Beim Anklicken der Schaltfläche **quit** wird dieses Fenster ohne Rückfrage des Browsers geschlossen.



Nachfolgend noch einmal der komplette Code für dieses Beispiel; Sie finden es auch auf der CD-ROM zum Buch unter dem Dateinamen **fscommand-2.html**. Wir haben neben dem Hinzufügen der JavaScript-Funktionalität noch einige kosmetische Änderungen vorgenommen:

- ▶ Das Dokument wurde weitestgehend XHTML-konform gemacht:
 - ▶ alle Elemente und Attribute/Parameter in Kleinbuchstaben,
 - ▶ jedes Tag wurde abgeschlossen,
 - ▶ statt `<script language="JavaScript">` besser `<script type="text/javascript">`,
 - ▶ alle Attribut- und Parameterwerte in Anführungszeichen.
- ▶ Wir haben einige kosmetische Änderungen bei Zeilenumbrüchen, Einrückungen etc. vorgenommen.

Sie sehen also, die Nachbearbeitung des von Flash automatisch generierten HTML-Codes ist durchaus sinnvoll; der Code ist nicht optimal.

```
<html>
<head>
<meta http-equiv=Content-Type content="text/html; charset=ISO-
8859-1" />
<title>fscommand</title>
</head>
<body bgcolor="#ffffff">
<script type="text/javascript">
<!--
var originalbreite, originalhoehe, x, y;
if (window.outerWidth) {
    originalbreite = window.outerWidth;
    originalhoehe = window.outerHeight;
    x = window.screenX;
    y = window.screenY;
} else if (document.body) {
    originalbreite = document.body.clientWidth;
    originalhoehe = document.body.clientHeight;
    x = window.screenLeft;
```

```

        y = window.screenTop;
    }

    // Handle all the FSCommand messages in a Flash movie
    function fscommand_DoFSCommand(command, args) {

        if (command == "fullscreen") {
            if (args == "true") {
                var breite = screen.availWidth;
                var hoehe = screen.availHeight;
                if (breite && hoehe) {
                    window.resizeTo(breite, hoehe);
                    window.moveTo(0, 0);
                }
            } else {
                if (originalbreite && originalhoehe)
                    window.resizeTo(originalbreite, originalhoehe);
                if (x && y)
                    window.moveTo(x, y);
            }
        }

        if (command == "exec") {
            if (args.indexOf("start\t") == 0)
                args = args.substring(6); //alles nach "start\t"
            window.open(args, "exec");
        }

        if (command == "quit")
            window.close();

    }

    // Hook for Internet Explorer
    if (navigator.appName && navigator.appName.indexOf("Microsoft")
        != -1 &&
        navigator.userAgent.indexOf("Windows") != -1 && navigator.userAgent.
        indexOf("Windows 3.1") == -1) {
        document.write('<script language="VBScript"> \n');
        document.write('on error resume next \n');
        document.write('Sub fscommand_FSCommand(ByVal command, ByVal
args)\n');
    }

```

```

        document.write('  call fscommand_DoFSCommand(command, args)\n');
        document.write('end sub\n');
        document.write('</script> \n');
    }
//-->
</script>
<!-- URLs used in the movie-->
<!-- text used in the movie-->
<!--fscommand()--><object classid="clsid:D27CDB6E-AE6D-11cf-
96B8-444553540000"
    codebase="http://download.macromedia.com/pub/shockwave/cabs/
flash/swflash.cab#version=6,0,0,0"
    id="fscommand" width="125" height="175">
    <param name="movie" value="fscommand.swf">
    <param name="quality" value="high">
    <param name="bgcolor" value="#ffffff">
    <embed src="fscommand.swf" quality="high" bgcolor="#ffffff"
    width="125" height="175" swLiveConnect="true" id="fscommand"
name="fscommand"
    type="application/x-shockwave-flash" pluginspage="http://
www.macromedia.com/go/getflashplayer">
    </embed>
</object>
</body>
</html>

```

Listing 17.4 Die komplette HTML-Seite mit JavaScript-Code

Tipps und Tricks

Hier folgen noch ein paar Hinweise, wie der Code innerhalb von `fscommand_DoFSCommand()` alternativ strukturiert werden könnte:

- ▶ Sie können anstelle von mehreren Abfragen der Machart `if (command == "xxx")` auch die `switch`-Anweisung verwenden. Hier bereiten allerdings einige Netscape-Versionen Probleme.
- ▶ Die `if`-Abfrage kann auch komplett eingespart werden, wenn Sie als Kommando JavaScript-Code übergeben und diesen dann mit der JavaScript-Funktion `eval()` auswerten. Innerhalb von Flash/ActionScript verwenden Sie also beispielsweise folgende Anweisung:

```
fscommand("alert('Es geht auch mit eval()');");
```

- und dann folgenden Code innerhalb der JavaScript-Funktion `fscommand_DoFSCommand()`:

```
eval(command);
```

Um die Funktion `fscommand_DoFSCommand()` (oder wie auch immer sie bei Ihnen heißt, das Schema ist `filmlname_DoFSCommand()`) übersichtlich zu halten, sollten Sie den eigentlichen Code in eigene Funktionen auslagern und dann in `fscommand_DoFSCommand()` nur noch diese Funktionen aufrufen:

```
var originalbreite, originalhoehe, x, y;
if (window.outerWidth) {
    originalbreite = window.outerWidth;
    originalhoehe = window.outerHeight;
    x = window.screenX;
    y = window.screenY;
} else if (document.body) {
    originalbreite = document.body.clientWidth;
    originalhoehe = document.body.clientHeight;
    x = window.screenLeft;
    y = window.screenTop;
}

function fullscreen_click(args) {
    if (args == "true") {
        var breite = screen.availWidth;
        var hoehe = screen.availHeight;
        if (breite && hoehe) {
            window.resizeTo(breite, hoehe);
            window.moveTo(0, 0);
        }
    } else {
        if (originalbreite && originalhoehe)
            window.resizeTo(originalbreite, originalhoehe);
        if (x && y)
            window.moveTo(x, y);
    }
}
```



```

function exec_click(args) {
    if (args.indexOf("start\t") == 0)
        args = args.substring(6);
    window.open(args, "exec");
}

function quit_click() {
    window.close();
}

// Handle all the FSCommand messages in a Flash movie
function fscommand_DoFSCommand(command, args) {
    switch (command) {
        case "fullscreen": fullscreen_click(args);
                               break;
        case "exec":         exec_click(args);
                               break;
        case "quit":         quit_click();
                               break;
    }
}

```

Listing 17.5 Der Code unter Verwendung von switch

Dieser Code ist natürlich nur ein Teil der HTML-Datei; wir haben beispielsweise den kleinen Kniff für den Internet Explorer beiseite gelassen. Das vollständige Listing finden Sie unter dem Dateinamen **fscommand-3.html** auf der CD-ROM zum Buch.

Eine wichtige Beobachtung noch: Sie haben gesehen, dass die »reservierten« Kommandos wie `fullscreen`, `exec` und `quit` nur innerhalb eines Stand-alone-Players gelten; im Webbrowser sind sie nicht vorbelegt. Das bringt bei einer cross-medialen Entwicklung für CD-ROM und Web Vorteile: Für die Desktop-Applikation verwenden Sie die vorgefertigten Kommandos, für den Web-Einsatz müssen Sie keinen neuen Film erstellen. Exportieren Sie einfach den alten Film wie oben gezeigt und belegen Sie die verwendeten Kommandos neu. Sie sparen so Entwicklungszeit und natürlich auch -kosten.

17.1.2 Die Funktion `getURL()`

Es gibt jedoch noch eine zweite Möglichkeit, JavaScript-Code aus einem Flash-Film heraus aufzurufen. Die (globale) Funktion, von der hier die Rede ist, heißt `getURL()`. Sie wurde bereits in der Flash-Version 2 eingeführt, aber insbesondere in Version 4 deutlich erweitert. In Kapitel 21, Serverseitige Technologien, werden Sie diese Funktion wieder finden.

Vom Prinzip her ist der Einsatz von `getURL()` sehr einfach. Sie übergeben als Parameter die URL, die geöffnet werden soll:

```
getURL("http://www.galileocomputing.de/");
```

Normalerweise würde dann die neue Seite in dem Browserfenster/Frame geöffnet werden, in dem der Film abgespielt wird (damit wird gleichzeitig die Filmausführung beendet). Als zweiten, optionalen Parameter können Sie jedoch den Namen des Fensters oder Frames übergeben. Beispielsweise würde folgende Anweisung die Homepage von Galileo Computing in einem neuen Fenster anzeigen:

```
getURL("http://www.galileocomputing.de/", "_blank");
```



Der zweite Parameter von `getURL()` entspricht also dem `target`-Parameter des HTML-`<a>`-Elements.

Was hat dies nun mit JavaScript zu tun? Wie Sie bereits wissen, kennen JavaScript-fähige Browser das Pseudo-Protokoll `javascript:`, mit dem per URL JavaScript-Kommandos ausgegeben werden können.

Mit `getURL()` können Sie auch diesen Weg gehen. Die folgende Anweisung ruft eine URL mit dem Pseudo-Protokoll `javascript:` auf und sorgt für die Ausführung des dort angegebenen Codes:

```
getURL("javascript:alert(\"Mit getURL()!\");");
```

Wie zu erwarten, öffnet hier der Browser ein modales Warnfenster.

Der eigentliche Clou an der Funktion `getURL()` ist jedoch ein anderer. Sie können damit nicht nur banale JavaScript-Kommandos aufrufen, die in keinem Zusammenhang mit der aktuellen HTML-Seite oder dem Flash-Film stehen. Der JavaScript-Code wird im Kontext der HTML-Seite mit dem eingebetteten Flash-Film ausgeführt. Sie können also auf alle in der HTML-Seite deklarierten Funktionen zugreifen und diese ausführen.

Um dies zu erreichen, müssen wir zunächst die Skripte im Flash-Film leicht verändern. Alle Aufrufe von `fscommand()` werden durch die Aufrufe der entsprechenden, im HTML-Dokument deklarierten Funktionen

ersetzt. Als Beispiel verwenden wir wiederum die drei Schaltflächen; der zugehörige JavaScript-Code existiert ja bereits.

Hier also die drei Handlerfunktionen, diesmal unter Verwendung von `getURL()`:

```
var fullscreen = false;

function push_fullscreen_click() {
    fullscreen = !fullscreen;
    getURL("javascript:fullscreen_click(\"" +
        fullscreen + "\\");");
}

function push_exec_click() {
    getURL("javascript:exec_click(\"fscommand.txt\");");
}

function push_quit_click() {
    getURL("javascript:quit_click();");
}
```

Der so modifizierte Film befindet sich auf der CD-ROM zum Buch unter dem Dateinamen **geturl.fl**.



Sie könnten nun wieder den Weg über die Veröffentlichen-Funktion von Flash gehen, um den Film einzusetzen; in den vorangegangenen Abschnitten haben wir aber schon eine Menge Vorarbeit geleistet. Wir ändern ja nur die Aufrufmethode (von `fscommand()` in `getURL()`), der Rest des Codes bleibt gleich. Aus diesem Grund kommen Sie am schnellsten ans Ziel, wenn Sie den letzten Stand der HTML-Datei (**fscommand-3.html**) verwenden und dort einfach den Namen der .swf-Datei abändern: von **fscommand.swf** in **geturl.swf**. Denken Sie daran, diese Ersetzung zweimal durchzuführen, einmal im `<object>`-Element, einmal im `<embed>`-Element. Wenn Sie sich die Tipparbeit sparen möchten, verwenden Sie die Datei **geturl.html** von der CD-ROM zum Buch (wo wir zusätzlich noch den Seitentitel angepasst und vor allem die nun unnötige Funktion `fscommand_DoFSCommand()` samt VBScript-Code für den Internet Explorer entfernt haben).

Sie müssen also für Netscape-Browser auch bei dieser Zugriffsmethode im `<embed>`-Tag `swLiveConnect="true"` setzen, damit der Zugriff funktioniert.



Wie Sie im Übrigen erfreut feststellen werden, funktioniert bei dieser Variante im Netscape-Browser sogar das Skript zum Öffnen des neuen Fensters mit der Datei oder dem Programm (siehe Abbildung 17.9).

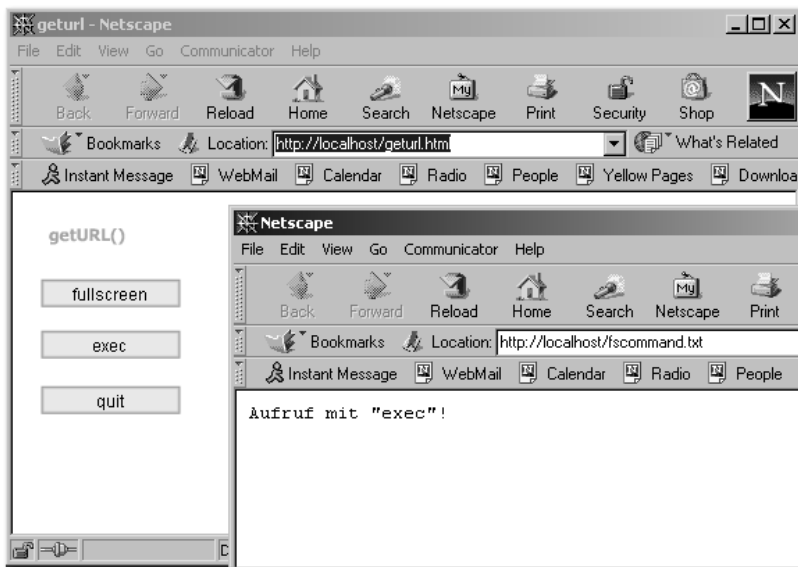


Abbildung 17.9 Jetzt wird auch im Netscape Navigator das neue Fenster geöffnet.

Bleibt nur noch zu klären, wann welche Funktion eingesetzt werden soll. Der Weg über `fscommand()` ist aufwändiger, da dort der umständliche Weg für den Internet Explorer über VBScript-Code und `xxx_DoFSCommand()` nicht mehr gegangen werden muss. Bei der Verwendung von `getURL()` können Sie sich diesen Weg sparen und müssen auch nicht die Unmenge an Code durch die Veröffentlichen-Funktion generieren lassen (aber trotzdem noch daran denken, für Netscape `swLiveConnect="true"` zu setzen, siehe oben). Von der JavaScript-Seite her ist es also ein wenig bequemer, wenn `getURL()` eingesetzt wird. Wenn Sie das Ganze jedoch aus dem Blickwinkel des ActionScript-Programmierers betrachten, ist dort der Einsatz von `fscommand()` der etwas bequemere Weg. Sie können zwei Parameter übergeben, während Sie bei `getURL()` recht mühsam und auch fehleranfällig den Aufrufstring aus `javascript:`, dem Funktions- oder Befehlsnamen und mehreren Argumenten (möglicherweise noch Strings, also müssen Anführungszeichen besonders behandelt werden etc.) zusammensetzen müssen. Unser Code zum Öffnen des neuen Fensters funktioniert nicht bei der Verwendung von `fscommand()`, mit `getURL()` aber sehr wohl. Es gibt wiederum andere Programme, bei

denen es sich andersherum verhält. Zudem dürfen von `getURL()` aufgerufene JavaScript-Funktionen keinen Rückgabewert liefern. Falls doch, würde dieser Rückgabewert (beispielsweise eine Zeichenkette, eine Referenz auf ein Fenster oder `null`) im Browserfenster ausgegeben werden. Wenn Sie beispielsweise in der Funktion `exec_click()` vor das `window.open` ein `return` stellen, hat die Funktion einen Rückgabewert, der im aktuellen Fenster ausgegeben wird. Der Rückgabewert ist vom Typ `window`, allgemeiner noch `object` (denn `window.open()` liefert eine Referenz auf das geöffnete Fenster zurück), die String-Repräsentation davon ist `[object]`. In Abbildung 17.10 sehen Sie den Beleg, einen Screenshot im Internet Explorer.

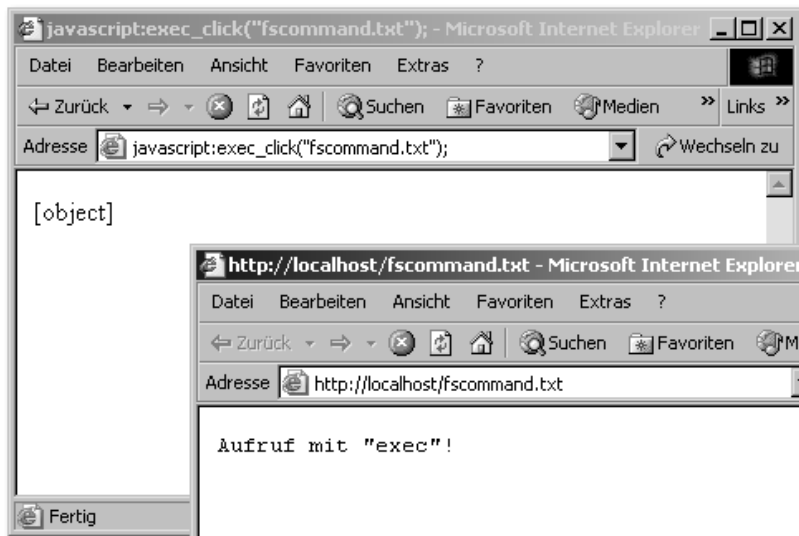


Abbildung 17.10 Achten Sie bei `getURL()` auf Rückgabewerte!

Und die Moral von der Geschichte' – es ist reine Geschmackssache. Testen Sie auf jeden Fall ausgiebig.

17.1.3 Anwendung: PHP-Session-Management

Nun möchten wir noch eine Anwendung mit großer Praxisrelevanz erstellen. Sie benötigen dazu allerdings Zugriff auf einen Server mit PHP-Unterstützung. PHP-Kenntnisse sind von Vorteil, aber nicht unbedingt notwendig.

Worum geht es? HTTP ist, wie hinlänglich bekannt, ein statusloses Protokoll; es ist also nicht möglich, Daten zwischen zwei Seitenabrufen zwi-

schenzuspeichern. Ein Ausweg aus dieser Misere ist der Einsatz von Cookies, kleinen Textinformationen, die auf dem Client (also im Webbrowser) zwischengespeichert werden. Allerdings sind insbesondere im deutschsprachigen Raum Cookies sehr ungern gesehen, obwohl bekannt ist, dass ein Großteil der vorgetragenen Befürchtungen kompletter Mümpitz sind.

Die meisten serverseitigen Programmiersprachen bieten hierzu ein so genanntes Session-Management an. Wenn ein Anwender eine Website aufruft und dann dort einige Links anklickt, spricht man von einer **Sitzung**, englisch **session**. In so einer Sitzung können bei den meisten serverseitigen Technologien Daten (z.B. Variablen) zwischengespeichert werden. Jede dieser Sitzungen erhält eine eindeutige Kennzeichnung, die so genannte **Session-ID**, meist eine kryptische Zeichenkette. Anhand dieser Kennung werden die einzelnen Sitzungen voneinander unterschieden.

Fast alle dieser Technologien regeln das über Cookies (die dann die Session-ID enthalten), bieten dazu aber alternativ noch die Möglichkeit an, einen Weg ohne Cookies zu beschreiten. Das sieht dann in der Regel so aus, dass die Session-ID per URL weitergegeben wird. Also wird ein Skript nicht mehr wie folgt aufgerufen:

```
http://servername/skriptname.xyz
```

sondern folgendermaßen:

```
http://servername/skriptname.xyz?Session-ID=abc123def
```

Wir wollen im Folgenden eine kleine, dreiseitige PHP-Applikation erstellen. Innerhalb dieser Applikation soll mitgezählt werden, wie viele Seiten sich der Benutzer schon angesehen hat. Normalerweise ist das eher trivial, denn PHP lässt sich so konfigurieren, dass die Session-ID automatisch an jeden Link auf einer Seite angehängt wird. Das Problem: Die Navigation erfolgt vollständig in Flash. Aus diesem Grund gehen wir einen anderen Weg. Vom Flash-Film aus rufen wir eine JavaScript-Funktion auf, die dann wiederum den angegebenen Link in den Browser lädt, vorher aber die Session-ID anhängt.

Bevor wir uns an die Programmierung machen, sollten Sie noch Ihre PHP-Installation überprüfen. In der zentralen Konfigurationsdatei **php.ini** finden Sie in der Rubrik [Session] die für das Session-Management zuständigen Konfigurationseinstellungen. Sorgen Sie mit der folgenden Anwei-

sung dafür, dass keine Cookies bei den Sessions verwendet werden, sondern alles über die URL abläuft:

```
session.use_cookies = 0
```

Außerdem sollten Sie verifizieren, dass das Verzeichnis, in dem Sessions abgelegt werden, existiert und der PHP-Prozess darauf Schreibrechte besitzt:

```
session.save_path = /tmp
```

Beginnen wir nun zunächst mit der JavaScript-Funktion selbst. Diese erhält eine URL als Parameter und ruft diese URL auf, hängt aber die Session-ID an. Dazu müssen Sie zunächst das Session-Management im Kopf der PHP-Seite starten:

```
<?php
session_start();
?>
```

Sie müssen `session_start()` (und alle Schreibzugriffe auf die Session) ausführen, bevor Sie den ersten HTML-Code an den Client schicken!



Nun erhalten Sie über die Funktion `session_id()` die Session-ID zurück, `session_name()` liefert den Namen der Session (standardmäßig "PHPSESSID"); damit lässt sich die JavaScript-Funktion schreiben:

```
function ladeURL(adresse) {
    location.href = adresse +
        "?<? echo(session_name() . "=" . session_id()); ?>";
}
```

Lautet die Session-ID beispielsweise "abc123def", so wird folgender Code vom PHP-Interpreter erzeugt:

```
function ladeURL(adresse) {
    location.href = adresse +
        "?PHPSESSID=abc123def";
}
```

Denn `session_id()` wird durch `abc123def`, `session_name()` durch `PHPSESSID` ersetzt.

Erstellen Sie nun drei Seiten **session1.php**, **session2.php** und **session3.php**, die alle nach demselben Muster aufgebaut sind. Im Kopf

der Seite wird versucht, aus der Session die Anzahl der besuchten Seiten auszulesen und diese um eins zu erhöhen:

```
<?php
session_start();
if ($_SESSION["besuche"]) {
    $besuche = $_SESSION["besuche"] + 1;
} else {
    $besuche = 1;
}
session_register("besuche");
?>
```

Weiter unten auf der Seite wird dann die Anzahl der Besuche ausgegeben:

```
<p>Dies ist die <?php echo($besuche); ?>. Seite in
    diesem Angebot, die Sie betrachten!</p>
```

Außerdem müssen Sie noch den Film einbauen, den wir im Folgenden erstellen werden; der Dateiname wird **session.swf** sein. Ansonsten sind die drei Seiten identisch, nur der Seitentitel ändert sich jeweils (damit die Seiten unterschieden werden können).

Nachfolgend finden Sie den kompletten Code für Seite 1 abgedruckt (**session1.php**); die Dateien **session2.php** und **session3.php** können auf dieser Basis erstellt werden (und befinden sich natürlich auch auf der CD-ROM zum Buch).

```
<?php
session_start();
if ($_SESSION["besuche"]) {
    $besuche = $_SESSION["besuche"] + 1;
} else {
    $besuche = 1;
}
session_register("besuche");
?>

<html>
<head>
<title>Sessions - Seite 1</title>
<script type="text/javascript"><!--
function ladeURL(adresse) {
```



```

        location.href = adresse +
            "?<?php echo(session_name()). "=" . session_id()); ?>";
    }
//--></script>
</head>
<body>
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
    codebase="http://download.macromedia.com/pub/shockwave/cabs/
flash/swflash.cab#version=6,0,0,0"
    id="session" width="125" height="175">
    <param name="movie" value="session.swf">
    <param name="quality" value="high">
    <param name="bgcolor" value="#ffffff">
    <embed src="session.swf" quality="high" bgcolor="#ffffff"
        width="125" height="175" swLiveConnect="true" id="session"
name="session"
        type="application/x-shockwave-flash" pluginspage="http://
www.macromedia.com/go/getflashplayer">
    </embed>
</object>
<p>Dies ist die <?php echo($besuche); ?>. Seite in diesem Ange-
bot,
    die Sie betrachten!</p>
</body>
</html>

```

Listing 17.6 Der Quellcode der ersten von drei PHP-Seiten

Auch hier ist es (für den Netscape-Browser) wieder wichtig, bei `<embed>` das `swLiveConnect`-Attribut zu setzen.



Als Nächstes muss noch der Film, `session fla` (und auf dieser Basis **session.swf**), erzeugt werden. Als Ausgangsbasis dient der Flash-Film aus den vorangegangenen Abschnitten.

Zunächst wird die Überschrift ersetzt, aus **fscommand()** bzw. **getURL()** wird **PHP-Sessions**. Dann ändern sich die Beschriftungen und Handler-Funktionen der drei Schaltflächen in die folgenden Werte:

Bezeichnung	Beschriftung	Click-Handler
seite1	Seite 1	seite1_click
seite2	Seite 2	seite2_click
seite3	Seite 3	seite3_click

Tabelle 17.3 Bezeichnungen, Beschriftungen und Click-Handler

Fehlen nur noch die drei Handlerfunktionen, die unsere JavaScript-Funktion `ladeURL()` aufrufen:

```
function seite1_click() {
    getURL("javascript:ladeURL(\"session1.php\");");
}

function seite2_click() {
    getURL("javascript:ladeURL(\"session2.php\");");
}

function seite3_click() {
    getURL("javascript:ladeURL(\"session3.php\");");
}
```

Listing 17.7 Der ActionScript-Code im Film **session.fla**

Erzeugen Sie aus diesem Film die Datei **session.swf** (sie befindet sich natürlich auch auf der CD-ROM zum Buch) und rufen Sie das Skript `session1.php` im Webbrowser auf. Klicken Sie nun einige der Schaltflächen an. Sie werden feststellen, dass (korrekte PHP-Installation und -Konfiguration vorausgesetzt) sich der Zähler bei jedem Seitenaufruf um eins erhöht (siehe Abbildung 17.11).



Rufen Sie die Seite über den Webserver auf, also über **`http://server-name/session1.php`** – auf keinen Fall über das Dateisystem, etwa **`c:\inet-pub\wwwroot\session1.php`**. In letzterem Fall nämlich würde der PHP-Interpreter außen vor bleiben und der eingebettete PHP-Code nicht zur Ausführung kommen.

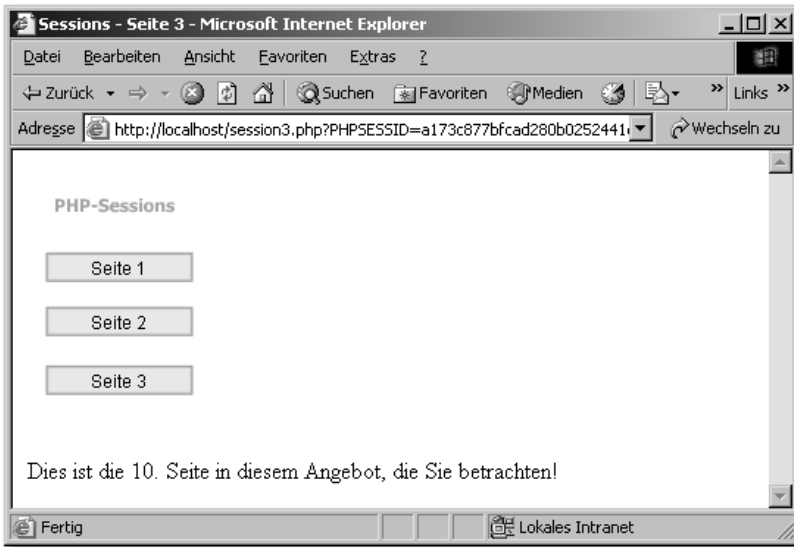


Abbildung 17.11 Die fertige Applikation

Wenn Sie dieses Beispiel auf ASP portieren möchten, lassen Sie es lieber – das Session-Management von ASP funktioniert **nur** mit Cookies. Wenn Sie es auf ASP.NET umschreiben möchten, benötigen Sie den JavaScript-Code nicht. Bei entsprechender Konfiguration legt der ASP.NET-Prozess beim Starten einer Session ein virtuelles Verzeichnis an, so dass Sie eine Seite `http://servername/(abc123def)/seite1.aspx` aufrufen, wobei `abc123def` die Session-ID ist. Sie können dann direkt auf `seite2.aspx` linken, denn bei relativen Links wird ja das Verzeichnis mit der Session-ID beibehalten, Sie verlieren also die Session-Informationen nicht.



Sie haben in diesem Beispiel gesehen, wie Sie ActionScript, JavaScript und eine serverseitige Skriptsprache wie etwa PHP miteinander verbinden und kombinieren können: Mehr zum Zusammenspiel von ActionScript und serverseitigen Technologien erfahren Sie in Kapitel 21, Serverseitige Technologien.

17.2 Von JavaScript nach ActionScript

Wir wollen nun den Weg in die andere Richtung beschreiten, also von der HTML-Seite aus mit JavaScript auf den Flash-Film zugreifen. Sie können beispielsweise mit HTML-Schaltflächen den Flash-Film steuern.

Zunächst einmal werfen wir noch einen Blick auf die Einbau-Syntax für Flash-Filme in ein HTML-Dokument:

17.2.1 Zugriff

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
  codebase="http://download.macromedia.com/pub/shockwave/cabs/
  flash/swflash.cab#version=6,0,0,0"
  id="flashfilm" width="125" height="175">
  <param name="movie" value="film.swf">
  <param name="quality" value="high">
  <param name="bgcolor" value="#ffffff">
  <embed src="film.swf" quality="high" bgcolor="#ffffff"
    width="125" height="175" swLiveConnect="true" id="flashfilm"
    name="flashfilm"
    type="application/x-shockwave-flash" pluginspage="http://
    www.macromedia.com/go/getflashplayer">
  </embed>
</object>
```

Es gibt drei Stellen, an denen Sie dem Flash-Film einen Namen geben können:

- ▶ im `id`-Parameter des `<object>`-Tags,
- ▶ im `name`-Parameter des `<object>`-Tags,
- ▶ im `id`-Parameter des `<embed>`-Tags (eigentlich unnötig, wird aber von Macromedia so propagiert, eventuell in Hinblick auf spätere Browser-versionen?).

Um ein browserübergreifendes Funktionieren Ihrer Skripte zu ermöglichen, müssen Sie diese drei Parameter auf denselben Wert setzen; am besten verwenden Sie dabei nur Buchstaben und Ziffern, keine Sonderzeichen.

Je nach Browsertyp gestaltet sich der Zugriff auf den Flash-Film anders. Wir betrachten hierbei nur den Microsoft Internet Explorer und den Netscape Navigator 4; der neue Netscape 6 unterstützt, wie bereits erwähnt wurde, den Zugriff auf den Flash-Film nicht.

Zuerst müssen Sie ermitteln, ob überhaupt ein Internet Explorer oder ein Netscape Navigator 4 vorliegt. Die vorgegebene Abfrage im von der Veröffentlichung-Funktion von Flash erzeugten HTML-/JavaScript-Code kann nur als »suboptimal« bezeichnet werden, weswegen wir hier einen eigenen Weg gehen. Zunächst einmal wird festgestellt, ob ein Internet Explorer vorliegt. Dann ist im Identifikationsstring des Browsers (`navigator.userAgent`) auf jeden Fall die Zeichenkette `MSIE` enthalten:

```
var IE = (navigator.userAgent.indexOf("MSIE") > -1);
```

Wenn die Zeichenkette vorkommt, liefert `indexOf()` einen Wert größer oder gleich null (die Zählung bei Zeichenketten beginnt bei 0 und nicht bei 1).

Beim Netscape Navigator ist es schon etwas schwieriger. Er identifiziert sich als Mozilla, allerdings tun das andere Browser wie eben der Internet Explorer auch. Aus diesem Grund findet hier die Überprüfung zweigleisig statt:

- ▶ `navigator.appName` muss den Wert »Netscape« haben (dann ist es ein Netscape-Browser),
- ▶ das erste Zeichen in `navigator.appVersion` muss eine 4 sein, dann ist es ein Netscape Navigator 4.x.

```
var N4 = (navigator.appName == "Netscape" &&  
         navigator.appVersion.charAt(0) == "4");
```

Die beiden Browser ermöglichen den Zugriff auf den eingebetteten Flash-Film auf unterschiedliche Art und Weise. Beim Internet Explorer ist der Film direkt unterhalb des `window`-Objekts verfügbar, Sie können also direkt über `window.filmname` oder gar nur über `filmname` darauf zugreifen. Der Netscape Navigator geht einen etwas logischer anmutenden Weg: Der Film ist ja in das Dokument eingebettet, also kann auf ihn als Unterobjekt des `document`-Objekts zugegriffen werden. Sie können hier also über `document.filmname` auf den Flash-Film zugreifen (`filmname` ist hierbei jeweils der Name des Films aus den drei `id`- bzw. `name`-Parametern).

In Wirklichkeit ist `document.filmname` eine Kurzform. Im Netscape Navigator 4 werden alle per `<embed>` eingebundenen Dateien im Array `document.embeds[]` abgelegt. Anstelle von `document.filmname` können Sie also auch `document.embeds["filmname"]` verwenden.



Und noch ein Hinweis: Der Internet Explorer unterstützt ebenfalls den Zugriff via `document.filmname`. Es ist zwar nicht geplant, dies in zukünftigen Versionen zu verändern, aber da die Browserunterscheidung nicht aufwändig ist, sollten Sie sie trotzdem vornehmen.



Das folgende Codefragment erstellt eine Variable namens `film`, der eine Referenz auf den Flash-Film zugewiesen wird:

```
var film = false;  
if (IE)
```

```

    film = filename;
else if (N4)
    film = document.filename;

```



Überprüfen Sie erst dann auf das Vorhandensein des Films, wenn das Dokument bereits vollständig geladen worden ist!

Sie können dann im weiteren Verlauf des Films über die Variable `film` auf den Film zugreifen. Um aber bei anderen Browsern als dem Internet Explorer und dem Netscape Navigator 4 einen Fehlermeldung zu verhindern, sollten Sie zunächst überprüfen, ob die Variable überhaupt gesetzt worden ist:

```

if (film) {
    // Zugriff auf Film, wie im Folgenden beschrieben
}

```

17.2.2 Funktionen und Methoden

Sie können nun von JavaScript aus über die Variable `film` eine Reihe von Methoden aufrufen, die den Flash-Film steuern können. In Tabelle 17.4 finden Sie eine alphabetische Übersicht.

Methode	Beschreibung	Ab Flash-Version ...
<code>GetVariable(Name)</code>	Liefert den Wert einer Flash-Variablen zurück	4
<code>GotoFrame(Nummer)</code>	Geht zum Frame mit der angegebenen Nummer. Im Gegensatz zu ActionScript beginnt hier jedoch die Zählung bei 0, <code>GotoFrame(9)</code> geht also zum zehnten Frame. Der Film wird danach angehalten.	2
<code>IsPlaying()</code>	Gibt als booleschen Wert zurück, ob der Film gerade abgespielt wird (<code>true</code>) oder nicht (<code>false</code>)	2
<code>LoadMovie(Ebene, Datei)</code>	Lädt die angegebene Datei in die Dokumentenebene (die Ebenennummer muss angegeben werden)	3

Tabelle 17.4 Die Methoden des Flash-Plug-Ins und -ActiveX-Controls

Methode	Beschreibung	Ab Flash-Version ...
Pan(x, y, Modus)	Führt Panning aus (Schwenken eines gezoomten Bereichs). Wenn Modus den Wert 0 hat, sind x und y Koordinaten, beim Wert 1 Prozentwerte.	2
PercentLoaded()	Gibt an, wieviel Prozent des Films bereits geladen worden sind. Achtung, der Rückgabewert liegt zwischen 0 und 100, nicht zwischen 0 und 1.	2
Play()	Spielt den Film ab	2
Rewind()	Spult den Film zum ersten Bild zurück; der Abspielkopf wird angehalten	2
SetVariable(Name, Wert)	Setzt eine Flash-Variable auf den angegebenen Wert. Der Wert wird auf jeden Fall in einen String konvertiert.	4
SetZoomRect(links, oben, rechts, unten)	Ein rechteckiger Bereich im Film wird vergrößert. Die Einheit, in der die Koordinaten angegeben werden müssen, ist twips – 1440 twips entsprechen einem Inch (Zoll), also 2,54 Zentimetern.	2
StopPlay()	Hält den Film an	2
TotalFrames()	Gesamtzahl der Bilder im Film	2
Zoom(Wert)	Verkleinert oder vergrößert den Film. Im Parameter wird angegeben, um wie viel in den Film hinein gezoomt wird; der Wert 20 etwa vergrößert den Film um das Fünffache, 500 verkleinert ihn um ein Fünftel.	2

Tabelle 17.4 Die Methoden des Flash-Plug-Ins und -ActiveX-Controls (Forts.)

Daneben gibt es noch eine Reihe von undokumentierten Methoden. Diese existieren offensichtlich, befinden sich aber beispielsweise nicht in der offiziellen Macromedia-Dokumentation. Dazu gehören unter anderem:



- ▶ `CurrentFrame()`, das die Nummer des aktuellen Bilds zurückliefert,
- ▶ `FrameLoaded(Nummer)`, womit Sie – genau wie innerhalb von ActionScript – feststellen können, ob ein bestimmtes Bild bereits geladen worden ist,
- ▶ die Eigenschaft `FlashVersion`, über die Sie die aktuelle Version des Flash-Plug-Ins/-ActiveX-Controls ermitteln und beispielsweise bei zu alten Plug-Ins/Controls eine Warnmeldung ausgeben können.

Wenn Sie das ActiveX-Control einsetzen, das heißt Ihre Anwender einen Microsoft Internet Explorer einsetzen, stehen noch zusätzliche Eigenschaften und Methoden zur Verfügung. Tabelle 17.3 können Sie die drei wohl wichtigsten entnehmen.

Methoden/Eigenschaft	Beschreibung	Ab Flash-Version ...
<code>Back()</code>	Geht ein Bild zurück	2
<code>Forward()</code>	Geht ein Bild vor	2
<code>ReadyState</code>	Status des Flash-Films: ▶ 0 = wird geladen ▶ 1 = noch nicht initialisiert ▶ 2 = geladen ▶ 3 = im Interaktiv-Modus (z. B.: wartet auf Anwendereingabe) ▶ 4 = komplett vorhanden	2

Tabelle 17.5 Einige Eigenschaften und Methoden des ActiveX-Controls



Auch, wenn der Microsoft Internet Explorer mit großem Abstand Marktführer ist: Wenn nur irgendwie möglich, verzichten Sie auf die ActiveX-Control-Erweiterungen und programmieren Sie browserunabhängig (denken Sie an den und testen Sie mit dem Netscape Navigator 4)! In neueren Netscape-Plug-Ins wurden zudem die Methoden `Back()` und `Previous()` implementiert. Verlassen Sie sich hier nicht auf die Dokumentation von Macromedia, die ist in dieser Hinsicht (zurzeit) nicht auf dem aktuellen Stand.

Zusätzlich existieren noch die so genannten Tell-Target-Methoden. Damit können Sie einzelne Filmsequenzen (in Flash MX: Movieclips) steuern. All diese Methoden beginnen mit einem `T`. Als ersten Parameter übergeben Sie immer die Sequenz, im Format "DokumenteEbene/Sequenzname". Wenn Sie als Parameter nur `"/` angeben, greifen Sie direkt auf die Zeit-
leiste zu.

Methode	Beschreibung	Ab Flash-Version ...
TCallFrame (Sequenz, Nummer)	Ruft das Bild mit der angegebenen Nummer auf (Achtung, Zählung beginnt wieder bei 0!)	4
TCallLabel (Sequenz, Name)	Ruft das Bild (samt zugehörigen Aktionen) mit der angegebenen Bezeichnung auf	4
TCurrentFrame (Sequenz)	Gibt die aktuelle Bildnummer zurück, Zählung beginnt bei 0	3
TCurrentLabel (Sequenz)	Gibt die Bezeichnung des aktuellen Bildes zurück, falls vorhanden	3
TGetProperty (Sequenz, Nummer)	Gibt die gewünschte Eigenschaft zurück. Als Parameter wird dabei ein numerischer Wert übergeben, der jeweils einer bestimmten Eigenschaft entspricht: ▶ 0: _x ▶ 1: _y ▶ 2: _xscale ▶ 3: _yscale ▶ 4: _currentframe ▶ 5: _totalframes ▶ 6: _alpha ▶ 7: _visible ▶ 8: _width ▶ 9: _height ▶ 10: _rotation ▶ 11: _target ▶ 12: _framesloaded ▶ 13: _name ▶ 14: _droptarget ▶ 15: _url ▶ 16: _highquality ▶ 17: _focusrect ▶ 18: _soundbuftime Der Rückgabewert ist immer ein String.	4
TGetProperty-AsNumber (Sequenz, Nummer)	Arbeitet wie TGetProperty(), allerdings ist der Rückgabewert numerisch, kein String	4
TGotoFrame (Sequenz, Nummer)	Springt zum angegebenen Bild, Zählung beginnt bei 0	3

Tabelle 17.6 Die Tell-Target-Methoden von Flash

Methoden	Beschreibung	Ab Flash-Version ...
TGotoLabel (Sequenz, Name)	Springt zum Bild mit der angegebenen Bezeichnung	3
TPlay(Sequenz)	Spielt die angegebene Sequenz ab	3
TSetProperty (Sequenz, Nummer, Wert)	Setzt die durch ihren Integer-Wert angegebene Eigenschaft (siehe Erläuterung zu TGetProperty()) auf den angegebenen Wert	4
TStopPlay (Sequenz)	Hält die angegebene Sequenz an	3

Tabelle 17.6 Die Tell-Target-Methoden von Flash (Forts.)

Mithilfe dieser Methoden können Sie nun von JavaScript aus den Film innerhalb gewisser Parameter steuern. Ein Tipp noch zum Schluss: Überprüfen Sie stets vor jedem Aufruf einer Funktion, ob der Film geladen worden ist. Zusätzlich sollten Sie noch überprüfen, ob die Methode auch existiert (damit verschiedene Versionen des Plug-Ins bzw. des ActiveX-Controls und auch unvollständige Implementierungen nicht zu Fehlermeldungen führen):

```
if (film && film.Methodenname) {
    film.Methodenname();
}
```



Diese Vorgehensweise führt beim Internet Explorer leider nicht zum richtigen Ergebnis. Dort wird selbst bei `if (film.Methodenname)` eine Fehlermeldung ausgegeben. Beim Netscape Navigator funktioniert die Abfrage wie geplant. Eine browserunabhängige Lösung ist:

```
if (IE || film.Methodenname) {
    film.Methodenname();
}
```

17.2.3 Beispiel: Preloader

Im Kapitel 16, Preloader, haben wir Ihnen bereits gezeigt, wie Sie mit ActionScript einen Preloader erstellen können. Diesmal gehen wir einen anderen Weg. Der Film wird mehr oder minder autark geladen, per JavaScript wird allerdings sein Ladestand permanent überprüft.

Die HTML-Seite besteht aus zwei `<div>`-Elementen; dabei handelt es sich um die beste browserunabhängige Möglichkeit, »DHTML«-Effekte zu erzeugen. Das erste `<div>`-Element enthält den Flash-Film, das zweite den Hinweistext mit dem Ladestatus:

```
<div id="flash" style="position:absolute">
  <object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
    codebase="http://download.macromedia.com/pub/shockwave/cabs/
flash/swflash.cab#version=6,0,0,0"
    id="flashfilm" width="125" height="175">
    <param name="movie" value="js-preloader.swf">
    <param name="quality" value="high">
    <param name="bgcolor" value="#ffffff">
    <embed src="js-preloader.swf" quality="high" bgcolor="#ffffff"
      width="125" height="175" swLiveConnect="true"
      id="flashfilm" name="flashfilm"
      type="application/x-shockwave-flash" pluginspage="http://
www.macromedia.com/go/getflashplayer">
    </embed>
  </object>
</div>
<div id="ladestatus"
style="position:absolute;x:25;y:200;left:25;top:200">
Ladestatus: 0%
</div>
```

Sie müssen für das `<div>`-Element mit dem Ladestatus auf jeden Fall die exakten Koordinaten angeben. Wenn Sie dies unterlassen, wird der Flash-Film über den Ladestatus gelegt, Sie bekommen also nichts zu sehen.



Nun benötigen wir noch einen Flash-Film, der nicht allzu klein ist. Dazu schießen wir einige Screenshots (wir haben die Website von Galileo Press verwendet) und blenden diese Screenshots in einem Flash-Film ein. Der so erzeugte Flash-Film (auf der CD-ROM unter dem Dateinamen **js-pre-loader fla**) ist zwar als komprimierte **.swf**-Datei unter 200 KB groß, über langsamere Leitungen ist der Effekt jedoch schon zu beobachten. Notfalls müssen Sie eben den Film deutlich größer machen.

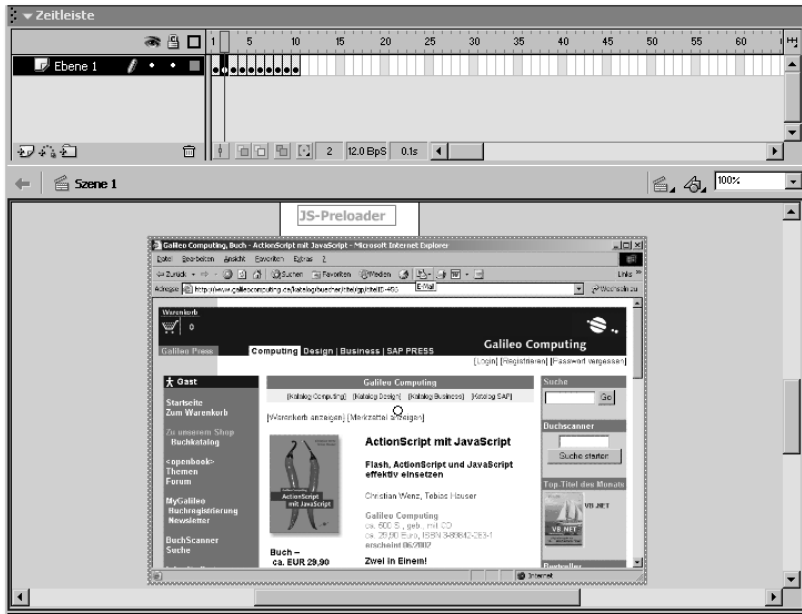


Abbildung 17.12 Der Beispiel-Flashfilm

Machen wir uns nun an die JavaScript-Programmierung. Zunächst einmal muss wieder für die beiden Browsertypen ein Verweis auf den eingebetteten Flash-Film erstellt werden. Dies ist auch schon die erste Hürde. Der Netscape Navigator behandelt alle Daten innerhalb eines `<div>`-Elements als eigenes Dokument. Sie können also nicht via `document.filmname` auf den Film zugreifen, sondern müssen folgenden Weg gehen:

```
var film =
    document.layers["flash"].document.embeds["flashfilm"];
```



Der Zugriff über `document.layers["flash"].document.flashfilm` funktioniert hier nicht (offensichtlich ein Bug). Wenn Sie auf ähnliche Fehler stoßen, wählen Sie beim Netscape Navigator immer den »ordentlichen« Zugriff über das Array `embeds[]`.

Der Zugriff auf den Film erfolgt innerhalb einer Funktion `init()`. Diese Funktion wird beim Laden des Dokuments aufgerufen:

```
window.onload = init;
```

In dieser Funktion wird neben der Bestimmung der Variablen `film` auch noch eine Funktion `fortschritt()` per Intervall alle 50 Millisekunden aufgerufen:

```

var ID;
function init() {
    if (IE)
        film = flashfilm;
    else if (N4)
        film = document.layers["flash"].document
            .embeds["flashfilm"];
    ID = setInterval("fortschritt()", 50);
}

```

Die Variable `ID`, welche die Intervall-ID darstellt, wird außerhalb der Funktion aufgerufen. Damit ist sie global gültig und das Intervall kann auch von einer anderen Funktion aus beendet werden.



Bleibt eigentlich nur noch die Funktion `fortschritt()` übrig. In dieser Funktion wird zunächst der aktuelle Ladestand ermittelt, sofern verfügbar:

```

var prozent = 0;
if (film)
    prozent = film.PercentLoaded();

```

Der Ladestatus wird dann im dem `<div>`-Element ausgegeben. Je nach Browser geht das unterschiedlich vonstatten:

- ▶ Beim Internet Explorer müssen Sie den Prozentwert in die Eigenschaft `document.all("ladestatus").innerHTML` schreiben.
- ▶ Beim Netscape Navigator ist es etwas aufwändiger; Sie müssen auf `document.layers["ladestatus"].document` zugreifen und dort `open()`, `write()` und `close()` durchführen.

Hier der zugehörige Code:

```

if (document.all)
    document.all("ladestatus").innerHTML =
        "Ladestatus: " + prozent + "%";
else if (document.layers)
    with (document.layers["ladestatus"].document) {
        open();
        write("Ladestatus: " + prozent + "%");
        close();
    }

```

Am Ende wird noch überprüft, ob der Film komplett geladen worden ist (dann wäre der Ladestatus 100%). Falls ja, wird die Ladeanzeige unsichtbar gemacht, denn sie ist nun nicht mehr nötig. Außerdem sollten Sie den Intervall-Aufruf von `fortschritt()` deaktivieren, da Sie sonst nur Rechenressourcen verschwenden.

```
if (prozent == 100) {
    clearInterval(ID);
    if (document.all)
        document.all("ladestatus").style.visibility="hidden";
    else if (document.layers)
        document.layers["ladestatus"].visibility="hidden";
}
```

Hier noch einmal der komplette Code im Überblick; auf der CD-ROM zum Buch finden Sie ihn unter dem Dateinamen **js-preloader.html**:

```
<html>
<head>
<title>JavaScript-Preloader</title>
<script type="text/javascript"><!--
window.onload = init;

var IE = (navigator.userAgent.indexOf("MSIE") > -1);
var N4 = (navigator.appName == "Netscape" &&
    navigator.appVersion.charAt(0) == "4");
var film = false;
var ID;

function fortschritt() {
    var prozent = 0;
    if (film)
        prozent = film.PercentLoaded();
    if (document.all)
        document.all("ladestatus").innerHTML =
            "Ladestatus: " + prozent + "%";
    else if (document.layers)
        with (document.layers["ladestatus"].document) {
            open();
            write("Ladestatus: " + prozent + "%");
            close();
        }
    if (prozent == 100) {
```

```

        clearInterval(ID);
        if (document.all)
            document.all("ladestatus").style.visibility="hidden";
        else if (document.layers)
            document.layers["ladestatus"].visibility="hidden";
    }
}

function init() {
    if (IE)
        film = flashfilm;
    else if (N4)
        film = document.layers["flash"].document.embeds["flash-
film"];
    ID = setInterval("fortschritt();", 50);
}
//--></script>
</head>
<body>
<div id="flash" style="position:absolute">
    <object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
        codebase="http://download.macromedia.com/pub/shockwave/cabs/
flash/swflash.cab#version=6,0,0,0"
        id="flashfilm" width="125" height="175">
        <param name="movie" value="js-preloader.swf">
        <param name="quality" value="high">
        <param name="bgcolor" value="#ffffff">
        <embed src="js-preloader.swf" quality="high"
            bgcolor="#ffffff"
            width="125" height="175" swLiveConnect="true"
            id="flashfilm" name="flashfilm"
            type="application/x-shockwave-flash" pluginspage="http://
www.macromedia.com/go/getflashplayer">
        </embed>
    </object>
</div>
<div id="ladestatus"
    style="position:absolute;x:25;y:200;left:25;top:200">
Ladestatus: 0%
</div>
</body>
</html>

```

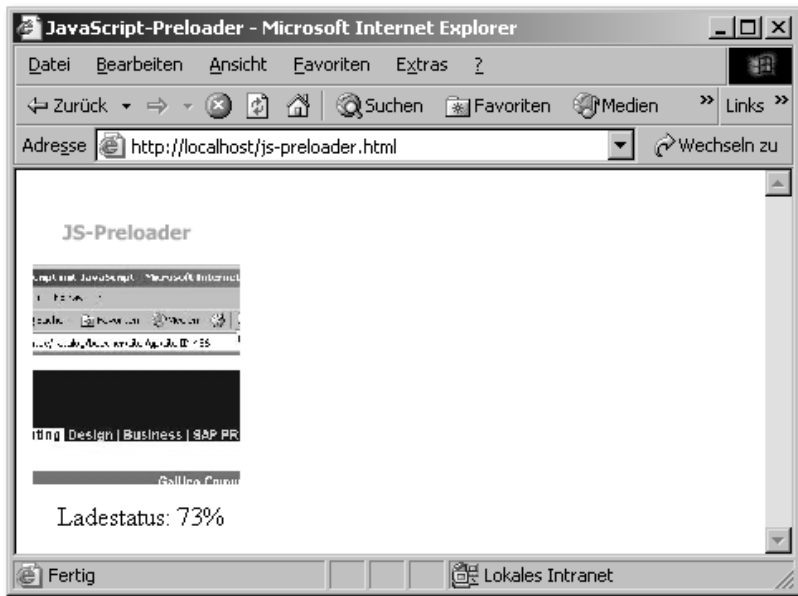


Abbildung 17.13 Der Film mit der JavaScript-Fortschrittsanzeige

17.2.4 Beispiel: Filmsteuerung

Im nächsten Beispiel wollen wir einige der Steuerungsoptionen für den Flash-Film im Einsatz zeigen. Dazu erstellen wir zunächst einen Film **steuerung fla**, der aus neun Bildern besteht. In jedem dieser Bilder befindet sich ein Textfeld mit der Nummer des aktuellen Bildes (also von 1 bis 9). Damit der Film auch nicht allzu schnell abläuft, stellen wir die Ablaufgeschwindigkeit auf **3 BpS** (Bilder pro Sekunde). Dies funktioniert über **Modifizieren • Dokument** oder über **[Strg] + [J]** (siehe Abbildung 17.14).

Der Film wird dann wie gewohnt in ein HTML-Dokument eingebunden. Zusätzlich erstellen wir wieder ein `<div>`-Element für den Ladefortschritt:

```
<div id="ladestatus" style="position:absolute;
x:25;y:200;left:25;top:200">
Ladestatus: 0%
</div>
```

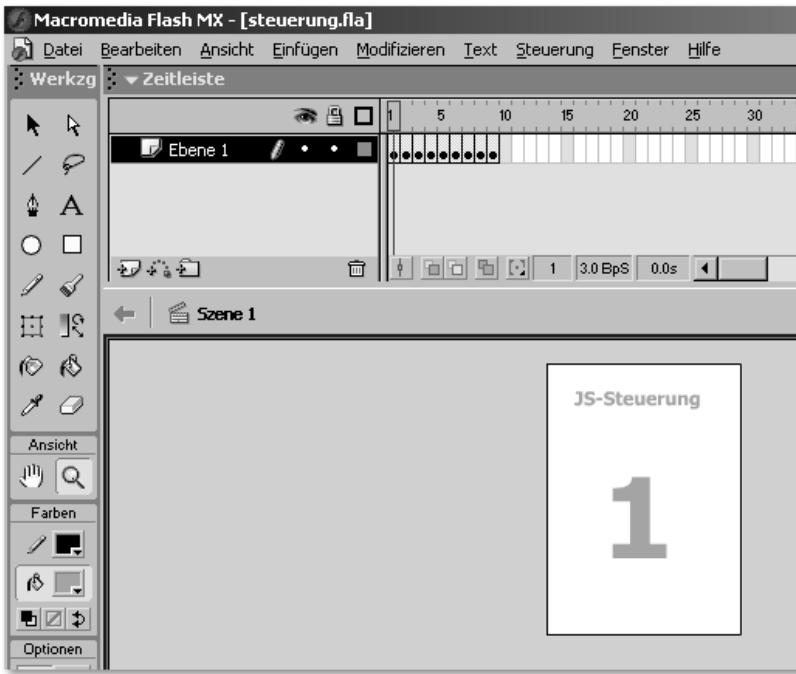



Abbildung 1714 Der Film besteht aus neun Bildern.

Die Filmsteuerung wird über ein HTML-Formular geregelt, über das die wichtigsten Steuerungsfunktionen von Flash aufgerufen werden können. Für zwei der verwendeten Funktionen, `GotoFrame()` und `Zoom()`, wird ein Textfeld zur Verfügung gestellt, in das der zugehörige Parameter für die Funktion eingegeben werden kann:

```
<form onsubmit="return false;">
  <input type="button" onclick="flash_play();" value="Play" />
  <input type="button" onclick="flash_stop();"
    value="Stop" />
  <input type="button" onclick="flash_rewind();"
    value="Rewind" />
  <br />
  <input type="button" onclick="flash_back();"
    value="Back" />
  <input type="button" onclick="flash_forward();"
    value="Forward" />
  <br />
  <input type="text" name="goto" size="1" />
```

```

        <input type="button" onclick="flash_goto(this.form);"
            value="GotoFrame" />
        <br />
        <input type="text" name="zoom" size="3" />
        <input type="button" onclick="flash_zoom(this.form);"
            value="Zoom" />
    </form>

```

Der Skriptbereich beginnt wieder mit dem Setzen der Variablen `film` auf den eingebetteten Flash-Film in der Funktion `init()`:

```

window.onload = init;
function init() {
    if (IE)
        film = flashfilm;
    else if (N4)
        film = document.flashfilm;
    ID = setInterval("fortschritt();", 100);
}

```

In der Funktion `fortschritt()` wird wieder der Ladestand ermittelt und ausgegeben, wie schon beim vorhergehenden Beispiel. Besonderheit diesmal: Beim Flash-ActiveX-Control gibt es die Eigenschaft `ReadyState`; diese wird gleich mit ausgegeben:

```

var prozent = 0;
if (film)
    prozent = film.PercentLoaded();
if (document.all)
    document.all("ladestatus").innerHTML =
        "Ladestatus: " + prozent + "%<br />" +
        "Abspielstatus: " + film.ReadyState;
else if (document.layers)
    with (document.layers["ladestatus"].document) {
        open();
        write("Ladestatus: " + prozent + "%");
        close();
    }

```

Der Rest besteht aus den Funktionen, die beim Anklicken der Schaltflächen per JavaScript aufgerufen werden. Für die meisten Schaltflächen sind diese Funktionen ziemlich einfach zu erstellen. Es wird überprüft, ob

der Film und (sofern kein Internet Explorer vorliegt) die Methode existieren. Falls ja, wird die Flash-Methode aufgerufen:

```
function flash_play() {
    if (film && (IE || film.Play))
        film.Play();
}
function flash_stop() {
    if (film && (IE || film.StopPlay))
        film.StopPlay();
}
function flash_rewind() {
    if (film && (IE || film.Rewind))
        film.Rewind();
}
function flash_back() {
    if (film && (IE || film.Back))
        film.Back();
}
function flash_forward() {
    if (film && (IE || film.Forward))
        film.Forward();
}
```

Nur minimal schwieriger sind die Funktionen, die Parameter erwarten. Wie Sie im HTML-Code des Formulars gesehen haben, wird an diese Funktionen praktischerweise eine Referenz auf das Formular übergeben:

```
<input type="button" onclick="flash_zoom(this.form);"
      value="Zoom" />
```

Damit können Sie direkt auf die Eingabe im zugehörigen Textfeld zugreifen. Diese wird mit `parseInt()` in einen Zahlenwert umgewandelt. Mit `isNaN()` wird überprüft, ob die Umwandlung funktioniert hat (`true`) oder ob in dem Feld gar keine Zahl stand.

```
function flash_goto(f) {
    var framenummer = parseInt(f.elements["goto"].value);
    if (film && (IE || film.GotoFrame)
        && !isNaN(framenummer))
        film.GotoFrame(framenummer);
}
function flash_zoom(f) {
```

```

var zoomlevel = parseInt(f.elements["zoom"].value);
if (film && (IE || film.Zoom)
    && !isNaN(zoomlevel))
    film.Zoom(zoomlevel);
}

```

Damit steht nun eine voll funktionsfähige Steuerungskonsole für den Flash-Film zur Verfügung. In Abbildung 17:15 sehen Sie, wie das im Browser aussieht.

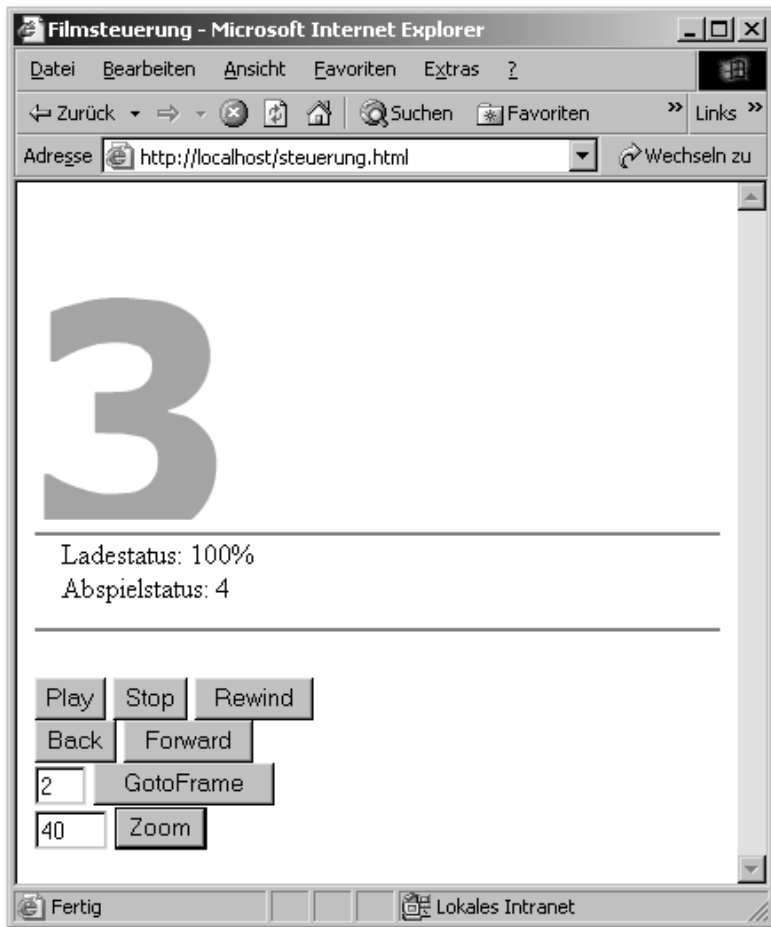


Abbildung 17:15 Die Steuerungskonsole im Webbrowser

Nachfolgend noch einmal der komplette Quellcode; Sie finden ihn zusätzlich auf der CD-ROM zum Buch unter dem Dateinamen **steuerung.html**.

```

<html>
<head>
<title>Filmsteuerung</title>
<script type="text/javascript"><!--
window.onload = init;

var IE = (navigator.userAgent.indexOf("MSIE") > -1);
var N4 = (navigator.appName == "Netscape" &&
    navigator.appVersion.charAt(0) == "4");
var film = false;
var ID;

function flash_play() {
    if (film && (IE || film.Play))
        film.Play();
}

function flash_stop() {
    if (film && (IE || film.StopPlay))
        film.StopPlay();
}

function flash_rewind() {
    if (film && (IE || film.Rewind))
        film.Rewind();
}

function flash_back() {
    if (film && (IE || film.Back))
        film.Back();
}

function flash_forward() {
    if (film && (IE || film.Forward))
        film.Forward();
}

function flash_goto(f) {
    var framenummer = parseInt(f.elements["goto"].value);
    if (film && (IE || film.GotoFrame)
        && !isNaN(framenummer))

```

```

        film.GotoFrame(framenumber);
    }

function flash_zoom(f) {
    var zoomlevel = parseInt(f.elements["zoom"].value);
    if (film && (IE || film.Zoom)
        && !isNaN(zoomlevel))
        film.Zoom(zoomlevel);
}

function fortschritt() {
    var prozent = 0;
    if (film)
        prozent = film.PercentLoaded();
    if (document.all)
        document.all("ladestatus").innerHTML =
            "Ladestatus: " + prozent + "%<br />" +
            "Abspielstatus: " + film.ReadyState;
    else if (document.layers)
        with (document.layers["ladestatus"].document) {
            open();
            write("Ladestatus: " + prozent + "%");
            close();
        }
    if (prozent == 100)
        clearInterval(ID);
}

function init() {
    if (IE)
        film = flashfilm;
    else if (N4)
        film = document.flashfilm;
    ID = setInterval("fortschritt();", 100);
}
//--></script>
</head>
<body>
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
    codebase="http://download.macromedia.com/pub/shockwave/cabs/

```

```

flash/swflash.cab#version=6,0,0,0"
  id="flashfilm" width="125" height="175">
    <param name="movie" value="steuerung.swf">
    <param name="quality" value="high">
    <param name="bgcolor" value="#ffffff">
    <embed src="steuerung.swf" quality="high" bgcolor="#ffffff"
      width="125" height="175" swLiveConnect="true" name="flashfilm"
      type="application/x-shockwave-flash" pluginspage="http://
www.macromedia.com/go/getflashplayer">
    </embed>
  </object>
<hr noshade />
<div id="ladestatus" style="position: absolute;
x:25;y:200;left:25;top:200">
Ladestatus: 0%
</div>
<br /><br />
<hr noshade />
<form onsubmit="return false;">
  <input type="button" onclick="flash_play();" value="Play" />
  <input type="button" onclick="flash_stop();" value="Stop" />
  <input type="button" onclick="flash_rewind();" value="Rewind"
/>
  <br />
  <input type="button" onclick="flash_back();" value="Back" />
  <input type="button" onclick="flash_forward();" value="Forward" />
  <br />
  <input type="text" name="goto" size="1" />
  <input type="button" onclick="flash_goto(this.form);"
value="GotoFrame" />
  <br />
  <input type="text" name="zoom" size="3" />
  <input type="button" onclick="flash_zoom(this.form);"
value="Zoom" />
</form>
</body>
</html>

```

Listing 17.8 Der vollständige Code für die Flash-Steuerungskonsole

17.2.5 Beispiel: Flash-Variablen setzen

Zum Abschluss dieses Kapitels noch ein kleines Beispiel, wie Sie die Methode `SetVariable()` einsetzen können. Wir wollen dazu eine einfache Form der Personalisierung vorstellen. Der Benutzer kann seinen Namen in ein Textfeld eingeben und wird in Zukunft auf der Webseite persönlich begrüßt. Damit die Informationen nicht verloren gehen, werden sie zusätzlich in einem Cookie gespeichert (aber nur in einem temporären Cookie, das nach dem Schließen des Browsers wieder gelöscht wird).

Wir beginnen mit dem Flash-Film. Dieser besteht im Wesentlichen aus einem dynamischen Textfeld und etwas schmückendem Beiwerk. Rufen Sie für das dynamische Textfeld den Eigenschafteninspektor auf und geben sie als Variablennamen `Name` ein (im Feld **Var**, siehe Abbildung 17.16). Das erlaubt es Ihnen, später auf dieses Textfeld zuzugreifen.

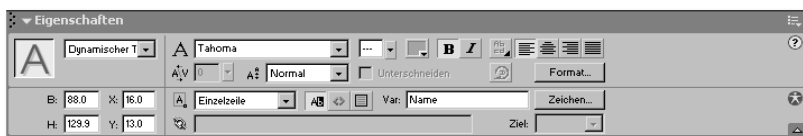


Abbildung 17.16 Der Eigenschafteninspektor für das Textfeld

Sie finden den kompletten Film unter dem Dateinamen **personalisierung.fla** auf der CD-ROM zum Buch; die daraus erzeugte **.swf**-Datei hat naheliegenderweise den Namen **personalisierung.swf**.

Beginnen wir nun mit der HTML-Datei. Wie der Flash-Film eingebaut wird, zeigen wir dieses Mal nicht explizit; denken Sie jedoch wie immer an das `swLiveConnect="true"` für den Netscape-Browser. Nach dem Film folgt das Formular, in das der Benutzer seinen Namen eintragen kann:

```
<div id="Formular" style="position:absolute">
  <hr noshade />
  <form onsubmit="NameSpeichern(this); return false;">
    <input type="text" name="Name" />
    <input type="submit" value="Speichern" />
  </form>
</div>
```

Beachten Sie, dass das Formular in einen `<div>`-Container gesteckt worden ist. Wenn nämlich der Benutzer bereits einen Namen eingegeben

hat, wollen wir ihn nicht mehr mit der Abfrage belästigen und können das Formular verstecken.

Beginnen wir mit einer Hilfsfunktion. Der Benutzername soll in einem Cookie namens `Name` abgespeichert werden. Die Funktion `CookieBenutzername()` liest dieses Cookie aus. Es wird prinzipiell in `document.cookie` nach der Zeichenkette `"Name="` gesucht. Alles, was darauf folgt, ist der Wert des Cookies, außer, es wird ein Semikolon gefunden, dann gehört alles bis zum Semikolon zum Wert des Cookies:

```
function CookieBenutzername() {
    var dc = document.cookie;
    if (dc.indexOf("Name=") > -1) {
        var name = dc.substring(dc.indexOf("Name=") + 5);
        if (name.indexOf(";") > -1)
            name = name.substring(0, name.indexOf(";") - 1);
        return unescape(name);
    }
}
```

Beim Laden des HTML-Dokuments wird diese Funktion aufgerufen. Wenn im Cookie ein Benutzername steht, werden zwei Dinge erledigt:

- Das Formular zur Eingabe des Benutzernamens wird ausgeblendet (die `visibility`-Eigenschaft wird auf `"hidden"` gesetzt).
- Der Name wird im Flash-Film ausgegeben.

```
var name = CookieBenutzername();
if (name != "") {
    if (document.all)
        document.all("Formular").style.visibility = "hidden";
    else if (document.layers)
        document.layers["Formular"].visibility = "hidden";
    NameSchreiben(name);
}
```

Fehlt eigentlich nur noch die Funktion `NameSchreiben()`. Diese ist aber besonders einfach. Es wird die Methode `SetVariable()` des Flash-Films aufgerufen. Die Variable, die gesetzt wird, heißt `Name` – so nämlich haben wir das Textfeld bezeichnet.

```
function NameSchreiben(name) {
    if (film && (IE || film.SetVariable))
```

```

        film.SetVariable("Name", name);
    }

```

Die einzige Funktionalität, die nun noch fehlt, ist das Speichern des Namens in dem Cookie. Diese Funktion wird aufgerufen, wenn das Formular verschickt worden ist:

```
<form onsubmit="NameSpeichern(this); return false;">
```



Beachten Sie das `return false` – dadurch wird sichergestellt, dass die Formulardaten nicht an den Webserver verschickt werden, sondern nur der JavaScript-Code ausgeführt wird.

Hier folgt nun der Code der Funktion `NameSpeichern()`:

```

function NameSpeichern(f) {
    var name = f.elements["Name"].value;
    document.cookie = "Name=" + escape(name);
    if (name != "")
        NameSchreiben(name);
}

```

Und das war es auch schon! Wenn Sie die Seite zum ersten Mal aufrufen, werden Sie zur Angabe Ihres Namens aufgefordert. Nach der Eingabe des Namens erscheint dieser sofort, ohne Neuladen, im Flash-Film. Beim nächsten Aufruf der Seite sehen Sie den Namen sofort, das Formular ist (scheinbar) verschwunden.

Sie können auf Basis dieses Codes alle Seiten des Webangebots analog aufbauen. Der eingegebene Name wird dann auf allen Seiten so angezeigt, da er ja im Cookie abgelegt worden ist.



Wenn Sie Zugriff auf eine serverseitige Programmiersprache haben, können Sie den Namen auch in einer Session-Variablen speichern. Dadurch können Sie auf Cookies verzichten, was Ihre Anwendung noch ein wenig unabhängiger von der Browserkonfiguration Ihrer Benutzer macht. JavaScript wird aber weiterhin benötigt.


```

function CookieBenutzername() {
    var dc = document.cookie;
    if (dc.indexOf("Name=") > -1) {
        var name = dc.substring(dc.indexOf("Name=") + 5);
        if (name.indexOf(";") > -1)
            name = name.substring(0, name.indexOf(";") - 1);
        return unescape(name);
    }
}

function NameSchreiben(name) {
    if (film && (IE || film.SetVariable))
        film.SetVariable("Name", name);
}

function NameSpeichern(f) {
    var name = f.elements["Name"].value;
    document.cookie = "Name=" + escape(name);
    if (name != "")
        NameSchreiben(name);
}

function init() {
    if (IE)
        film = flashfilm;
    else if (N4)
        film = document.flashfilm;
    var name = CookieBenutzername();
    if (name && name != "") {
        if (document.all)
            document.all("Formular").style.visibility = "hidden";
        else if (document.layers)
            document.layers["Formular"].visibility = "hidden";
        NameSchreiben(name);
    }
}

//--></script>
</head>
<body>
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
    codebase="http://download.macromedia.com/pub/shockwave/cabs/

```

```

flash/swflash.cab#version=6,0,0,0"
  id="flashfilm" width="125" height="175">
  <param name="movie" value="personalisierung.swf">
  <param name="quality" value="high">
  <param name="bgcolor" value="#ffffff">
  <embed src="personalisierung.swf" quality="high"
    bgcolor="#ffffff"
    width="125" height="175" swLiveConnect="true"
    name="flashfilm"
    type="application/x-shockwave-flash" pluginspage="http://
www.macromedia.com/go/getflashplayer">
  </embed>
</object>
<p>
<div id="Formular" style="position:absolute">
  <hr noshade />
  <form onsubmit="NameSpeichern(this); return false;">
    <input type="text" name="Name" />
    <input type="submit" value="Speichern" />
  </form>
</div>
</p>
</body>
</html>

```

Listing 17.9 Personalisierung mit Flash, JavaScript und Cookies

17.3 Fazit

In diesem Kapitel haben Sie gesehen, dass sich Flash und JavaScript nicht ausschließen; in der Kombination sind diese beiden Technologien sogar noch mächtiger als jede einzelne für sich. Wir konnten hier natürlich nur einen Bruchteil der schier unendlichen Einsatzmöglichkeiten vorführen; aufgrund der Methodenübersichten in diesem Kapitel sowie Ihrer Erfahrungen mit JavaScript und ActionScript in diesem Buch können Sie aber auch Ihre eigenen Ideen schnell verwirklichen.

Beachten Sie bei der ganzen möglichen Begeisterung, dass Sie beim Einsatz von JavaScript den Kreis derjenigen Besucher, die alle Effekte Ihrer Programmierung sehen können, einschränken. Insbesondere auf der Macintosh-Plattform funktioniert, wie gezeigt, nicht alles wie gewünscht. Überlegen Sie sich also auf alle Fälle, ob und wie Sie eine Alternativlösung

ohne JavaScript – und am besten ganz ohne Flash – anbieten können. Damit sieht ein Großteil Ihrer Besucher alle Effekte, der kleinere Teil, der aber oft der entscheidende sein kann, freut sich aber, dass er alle Informationen auf Ihrer Seite abrufen kann, wenn auch nicht so schick, als wenn er das Flash-Plug-In bzw. -ActiveX-Control installieren würde.

18 XML

Gott gab uns die Sprache, damit wir aneinander vorbeireden können.

– Friedhelm Franken

XML war lange Zeit ein Modethema – nur in Sachen Praxisanwendungen tat sich nicht viel. Heute hat sich die Situation ein wenig verbessert. Die meisten besseren Datenbanken bieten die Möglichkeit eines XML-Exports ihrer Daten, immer mehr Softwareprodukte verwenden XML zumindest an der einen oder anderen Stelle. Beispielsweise hat sich Microsoft bei den Komponenten für das neue **.NET Framework** von den zeilenbasierten INI-Dateien verabschiedet und arbeitet jetzt mit Dateien auf XML-Basis.

XML ist ein W3C-Standard, XML ist weit verbreitet – ist XML auch die Eier legende Wollmilchsau? Wir meinen: nein. So schön das Format XML auch ist, ein binäres Dateiformat ist beinahe immer effizienter. Besonders für kleine, einfache Anwendungen ist es wie das sprichwörtliche Schießen mit Kanonen auf Spatzen, hier großspurig auf XML zu setzen – meist gibt es ein anderes, proprietäres und improvisiertes Dateiformat, welches die Aufgabe viel schneller löst, wenngleich dann das Ego des Entwicklers nicht so sehr befriedigt worden ist. Wer's braucht ...

Seit Version 5 unterstützt Flash XML direkt (d.h. ohne externe Zusatzkomponenten) und die `XML`-Klasse ist im Lieferumfang enthalten. Somit ist es möglich, mit ActionScript auf XML-Datenquellen zuzugreifen und auch XML zu erzeugen. In Flash MX hat sich insofern eine Neuerung ergeben, als die XML-Unterstützung jetzt **nativ** ist. Das bedeutet, die XML-Unterstützung ist direkt in den Flash-Player beziehungsweise den ActionScript-Interpreter eingebaut. Bei Flash 5 war das noch nicht der Fall (dort waren Funktionen im Hintergrund, die das XML-Verhalten mit ActionScript-Mitteln nachgebildet haben). Ergebnis: Unter Flash MX läuft der Zugriff auf XML-Daten deutlich schneller ab als unter Flash 5. Bei den eher geringen Datenmengen, die in Flash-Filmen üblich sind, ist das eine nette, aber keine entscheidende Neuerung.

Wir werden in diesem Kapitel **nicht** großartig erzählen, was XML ist und was nicht. Dies überlassen wir anderen Büchern zu diesem Thema. Stattdessen stellen wir zunächst ganz kurz das XML-Konzept vor und zeigen Ihnen dann, wie Sie XML einlesen, verarbeiten und ausgeben können.

Als Praxisbeispiel für dieses Kapitel haben wir uns etwas ganz Besonderes einfallen lassen. Mit RSS steht im World Wide Web ein etabliertes XML-basiertes Dateiformat zum Austausch von Newsmeldungen zur Verfügung. Wir implementieren eine kleine Anwendung, welche RSS-Dateien einliest, interpretiert und daraus ein wechselndes Werbefbanner erstellt.

18.1 Was ist XML?

Das Akronym XML steht für **eXtensible Markup Language**. Die Standardhüter des World Wide Web, das World Wide Web Consortium (W3C; <http://w3.org/>), wachen über den Standard. Auf der Projekt-Homepage, <http://w3.org/XML/>, wird XML als »universelles Format für strukturierte Dokumente und Daten im Web« beschrieben. Das trifft die Intention des Formats ziemlich genau und ist gleichermaßen der Grund für die hohe Anerkennung von XML und die Ursache für einige Probleme, die das Format hat.

XML selbst verwendet Tags, genauso wie HTML und XHTML. Innerhalb dieser Tags können Attribute und Parameter verwendet werden. Sie kennen das bereits von HTML her:

```
<IMG SRC="grafik.gif" BORDER=0>
```

XML ist jedoch sehr strengen Regeln unterworfen. Beispielsweise müssen alle Attributwerte in Anführungszeichen eingeschlossen sein, zu jedem Tag muss es ein Abschlusstag geben. Während alle Webbrowser die obige Syntax akzeptieren, wäre es jedoch kein gültiges XML und auch kein gültiges XHTML (ein Versuch, HTML XML-konform zu machen). Korrekt wäre Folgendes:

```
</img>
```



Wir haben den Tag- und die Attributnamen in Kleinbuchstaben umgewandelt, wie in XHTML üblich. Außerdem wird als Dateiformat PNG anstelle des patentgeschützten GIF verwendet, dies nur ganz nebenbei.

Das `<img>`-Element ist aber auch ein gutes Beispiel für ein leeres HTML-Element; denn alle wichtigen Angaben stehen schon in den Attributen des Tags. Für einen solchen Fall benötigen Sie kein explizites Abschlusstag, ein Schrägstrich vor dem `>` reicht aus:

```

```



Das Leerzeichen vor dem Schrägstrich ist eigentlich nicht nötig, einige Webbrowser haben allerdings Probleme, wenn das Leerzeichen fehlt.

So viel zu HTML/XHTML; Sie sehen hier aber schon einige der Regeln, die für XML gelten:

- ▶ Jedes Element muss wieder geschlossen werden – zu einem Tag gehört immer ein End-Tag.
- ▶ Elemente dürfen verschachtelt werden, die Schließ-Reihenfolge ist jedoch genau umgekehrt zur Reihenfolge des Auftretens der Tags. Somit ist `<a><b>c</a></b>` falsch, `<a><b>c</b></a>` dagegen richtig.
- ▶ Attributwerte müssen in Anführungszeichen gesetzt werden.
- ▶ Zwischen Groß- und Kleinschreibung wird unterschieden, `<a>` und `<A>` sind zwei verschiedene Elemente.
- ▶ Auf der obersten Ebene der Elementhierarchie darf es nur ein Element geben. Das heißt das erste Tag im XML-Dokument ist gleichzeitig dasjenige Tag, das ein einziges Mal, ganz am Ende des Dokuments, geschlossen wird. Korrekt ist also `<a>b</a>`, falsch ist `<a>b</a><a>c</a>`. Dieses oberste Element wird auch als **Root-Element** bezeichnet (engl. root = dt. Wurzel).
- ▶ Ganz zu Beginn jedes XML-Dokuments wird das Dokument überhaupt erst als XML ausgezeichnet:

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

XML selbst besteht komplett aus Text. Nachfolgend ein XML-Dokument, gefunden unter <http://www.cnn.com/cnn.rss>, etwas umformatiert und gekürzt:

```
<?xml version="1.0"?>
<!DOCTYPE rss PUBLIC "-//Netscape Communications//DTD RSS 0.91//EN"
    "http://my.netscape.com/publish/formats/rss-0.91.dtd">
<rss version="0.91">
  <channel>
    <title>News from CNN.com</title>
    <link>http://cnn.com/index.html</link>
    <description>The world's news leader</description>
    <language>en-us</language>
    <item>
      <title>CNN.com</title>
      <link>http://cnn.com</link>
    </item>
  </channel>
</rss>
```

Listing 18.1 Eine exemplarische XML-Datei

Sie sehen in diesem Beispiel schon einige der Konzepte:

- Das Dokument beginnt wie erwartet mit `<?xml version="1.0"?>`.
- Es gibt nur ein Root-Element, nämlich `<rss>`.
- Jedes Element wird auch geschlossen, alle Attribute stehen in Anführungszeichen.

Bei den meisten dieser Elemente ist Ihnen auch klar, was sie bedeuten. `<title>CNN.com</title>` scheint einen Titel oder eine Beschreibung (wofür auch immer) zu beinhalten, `<link>http://cnn.com</link>` ist ein Link auf eine Website. Aber: Eines der erklärten Ziele von XML ist es, dass – im Gegensatz zu HTML – nicht versucht werden muss zu interpretieren, was der Autor mit seinen Tags eigentlich ausdrücken wollte. Dafür gibt es indes strenge Regeln, so genannte **DTDs**. Wieder so eine Abkürzung, und dieses Mal ist nicht eindeutig zu beantworten, wofür das Kürzel steht. Am gebräuchlichsten ist **Document Type Definition**, im World Wide Web finden sich aber eine ganze Reihe von Quellen, die von **Document Type Declaration** sprechen.

Es handelt sich bei einer DTD um eine formale Beschreibung, welche Elemente erlaubt sind und an welcher Stelle welches Element stehen darf. Im obigen Beispiel haben Sie beispielsweise gesehen, dass die Elemente `<title>` und `<link>` innerhalb des `<item>`-Elements stehen, welches wiederum innerhalb des `<channel>`-Elements untergebracht worden ist. Dieser Aufbau ist mandatorisch und wird in einer DTD festgelegt.

Es ist keineswegs verpflichtend, so eine DTD zu verwenden; allerdings ist eine der Stärken von XML, dass Daten durch eine solche DTD validiert werden können. Sie können also nicht nur feststellen, ob eine Datei alle XML-Regeln befolgt hat (man spricht hier von einer **wohlgeformten** XML-Datei), sondern auch, ob diese Daten in der richtigen Struktur angeordnet sind. Je nach Anwendung ist die »richtige Struktur« eine andere und kann in einer DTD festgehalten werden.

Die DTD wird – sofern sie verwendet werden soll – ebenfalls zu Beginn des XML-Dokuments angegeben, direkt nach `<?xml version="1.0"?>`:

```
<!DOCTYPE rss PUBLIC "-//Netscape Communications//DTD RSS 0.91//
EN"
    "http://my.netscape.com/publish/formats/rss-0.91.dtd">
```

Im obigen Beispiel wird also auf eine auf dem Netscape-Server liegende DTD verwiesen. Hier eine Zeile aus dieser DTD:

```
<!ELEMENT item (title | link | description)*>
```

Hier passiert Folgendes: `<item>` wird als Element definiert, das (übrigens beliebig viele) Unterelemente `<title>`, `<link>` und `<description>` haben kann.

XML selbst enthält nur die Daten, kümmert sich aber auf keinen Fall um deren Darstellung. XML ist also ein reines Daten-Format, die Repräsentation obliegt anderen Mechanismen (beispielsweise XSLT, eine Möglichkeit, XML-Daten in andere, anzeigbare Formate wie etwa (X)HTML umzuwandeln). Dennoch können Sie XML-Dateien grafisch anzeigen lassen. Wenn Sie eine Datei mit der Endung `.xml` im Internet Explorer öffnen, werden die Elemente angezeigt (siehe Abbildung 18.1). Sie können sogar per Mausklick einzelne Elemente samt Unterelementen ein- und ausklappen. Dies ist vor allem zum Testen ziemlich praktisch, denn der Internet Explorer kann zwar nicht DTDs überprüfen, findet aber Syntax-Fehler in der XML-Anweisung. Wenn Sie beispielsweise die letzte Zeile des Dokuments, `</rss>`, entfernen, erhalten Sie eine Fehlermeldung (siehe Abbildung 18.2).

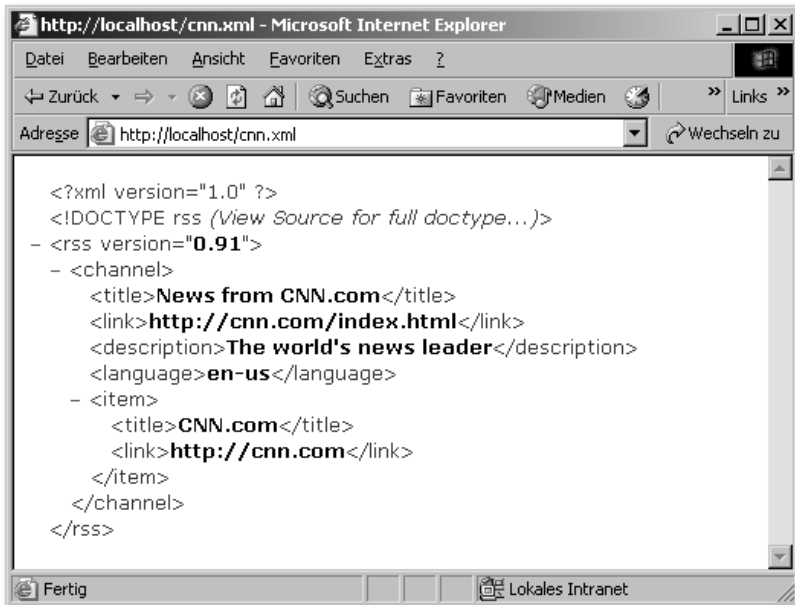


Abbildung 18.1 Die zuvor beschriebene XML-Datei im Webbrowser

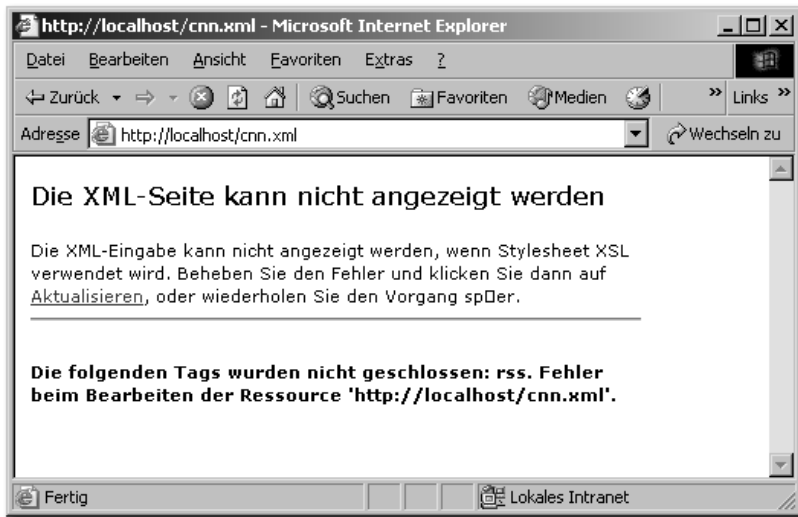


Abbildung 18.2 Fehlermeldung, wenn `</rss>` fehlt

Damit wollen wir unseren Ausflug in die Welt von XML aber auch schon beenden. Für weitere Informationen verweisen wir Sie auf die bereits erwähnte offizielle XML-Seite beim W3C sowie auf die offiziellen XML-FAQ unter <http://www.ucc.ie/xml/>.

18.2 Das XML-Objekt

Flash ist nun seit Version 5 in der Lage, XML-Daten zu verarbeiten. Dazu gibt es das XML-Objekt.

Es gibt zwei Möglichkeiten, dieses Objekt zu instanzieren:

1. Mit `var xmldoc = new XML();` – es wird einfach ein neues XML-Objekt ohne Inhalt erstellt.
2. Mit `var xmldoc = newXML(XML-Code);` hier übergeben Sie als Parameter gültigen XML-Code (ohne `<?xml ... ?>`, beispielsweise `"<rss></rss>"`).

Nach der Objektinstanzierung steht Ihnen das XML-Objekt mit all seinen Methoden und Eigenschaften zur Verfügung. Diese sind im Folgenden alle aufgeführt; danach folgt eine Reihe von Beispielen. Zunächst aber möchten wir noch einmal ein XML-Dokument visualisieren, damit die im Folgenden verwendeten Begriffe – Knoten, Kinder etc. – verständlicher werden.

Als Beispiel verwenden wir eine nochmals gekürzte Version des beschriebenen XML-Dokuments:

```
<?xml version="1.0"?>
<rss version="0.91">
  <channel>
    <item>
      <title>CNN.com</title>
      <link>http://cnn.com</link>
    </item>
  </channel>
</rss>
```

Listing 18.2 Die verkürzte Version der XML-Datei

Das `<rss>`-Element ist das Root-Element des XML-Dokuments, man spricht auch vom **Root-Knoten**. Das nächste in der Hierarchie ist das `<channel>`-Element. Man sagt auch, das `<channel>`-Element ist der erste Kindknoten des `<rss>`-Knotens. Analog hat `<channel>` auch einen Kindknoten, `<item>`. Dieser Knoten hat wiederum zwei Kindknoten, `<title>` und `<link>`.

Jetzt wird es interessant: `<title>` und `<link>` haben jeweils auch einen Kindknoten! Der Text innerhalb eines Elements ist ebenfalls ein Knoten, ein **Textknoten**, um genau zu sein.

In Abbildung 18.3 wird diese Struktur visualisiert.

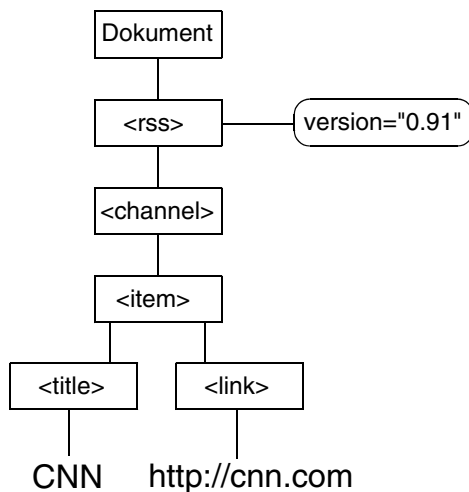


Abbildung 18.3 Die Baumstruktur des XML-Dokuments



In Flash gibt es eigentlich zwei XML-Klassen: einmal die XML-Klasse selbst, die für ein XML-Dokument steht; außerdem gibt es noch die Klasse XMLnode, die einen XML-Knoten repräsentiert. Die Handhabung der beiden Klassen ist praktisch identisch; allerdings stehen einige Methoden und Eigenschaften nur bei der XML-Klasse zur Verfügung (beispielsweise die Eigenschaft zum Laden eines neuen XML-Dokuments). Alle Eigenschaften und Methoden, die sowohl beim XML-Dokument als auch bei jedem einzelnen Knoten verfügbar sind, werden mit einem Sternchen (*) gekennzeichnet.

Wir beginnen mit den Methoden, die hauptsächlich der Erweiterung und Modifikation des vorliegenden XML-Dokuments dienen:

Methoden	Beschreibung	Beispiel
appendChild (Knoten) *	Hängt einen Knoten an einen anderen Knoten als Kindknoten an. Wenn der Knoten schon Kindknoten hat, wird der neue Knoten zum letzten Kindknoten.	<pre>/* k sei ein Knoten */ var x = new XML("<a>"); x.appendChild(k); /* XML-Dokument ist jetzt <a> */</pre>
cloneNode (rekursiv) *	Erstellt eine Kopie des angegebenen Knotens. Wenn als Parameter rekursiv der Wert true übergeben wird, werden Unterobjekte mit »geklont«, bei false nicht.	<pre>/* k sei ein Knoten <a>b */ var x = new XML(); l = k.cloneNode(false); x.appendChild(k); x.appendChild(l); /* XML-Dokument ist jetzt <a>b<a> - Unterknoten (Typ Text) "b" wird nicht mitkopiert*/</pre>
createElement (Text)	Erstellt einen neuen Knoten vom angegebenen Typ	<pre>var x = new XML(); var k = x.createElement("a"); x.appendChild(k); trace(x); /* Ausgabe <a /> */</pre>

Tabelle 18.1 Die Methoden des XML-Objekts

Methoden	Beschreibung	Beispiel
<code>createTextNode(Text)</code>	Erstellt einen neuen Textknoten mit dem angegebenen Text	<pre> var x = new XML(); var k = x.createElement("a"); var l = x.createTextNode("b"); k.appendChild(l); x.appendChild(k); trace(x); /* Ausgabe: <a>b */ </pre>
<code>hasChildNodes()</code> *	Gibt an, ob ein Knoten Kindknoten hat (Rückgabewert <code>true</code>) oder nicht (Rückgabewert <code>false</code>)	<pre> var x = new XML(); trace(x.hasChildNodes()); /* Ausgabe: false */ </pre>
<code>insertBefore(neu, alt)</code> *	Fügt in ein XML-Dokument den Knoten <code>neu</code> direkt vor <code>alt</code> ein	<pre> var x = new XML(); var k = x.createElement("a"); var l = x.createElement("b"); x.appendChild(l); x.insertBefore(k, l); trace(x); /* Ausgabe: <a /> */ </pre>
<code>load(URL)</code>	Lädt eine XML-Datei	<pre> var x = new XML(); x.load("datei.xml"); </pre>
<code>parseXML(Text)</code>	Lädt XML-Daten aus einem String in das XML-Objekt und ersetzt die vorherigen Daten	<pre> var x = new XML("<a>b"); x.parseXML("<c>d</c>"); trace(x); /* Ausgabe: <c>d</c> */ </pre>
<code>removeNode()</code> *	Entfernt den angegebenen Knoten	<pre> var x = new XML(); var k = x.createElement("a"); var l = x.createElement("b"); x.appendChild(k); x.appendChild(l); k.removeNode(); trace(x); /* Ausgabe: */ </pre>

Tabelle 18.1 Die Methoden des XML-Objekts (Forts.)

Methode	Beschreibung	Beispiel
<code>send(URL, Ziel)</code>	Wandelt die XML-Daten in einen String um und sendet diese an eine URL – innerhalb des Projekts per GET, innerhalb des Browsers per POST. Als <code>Ziel</code> kann der Name eines Fensters oder Frames angegeben werden.	<pre>var x = new XML("<a>b"); x.send("skript.asp", "_blank"); /* Skriptausgabe wird in einem neuen Fenster dargestellt */</pre>
<code>sendAndLoad(URL, Zielobjekt)</code>	Sendet wie <code>send()</code> Daten an ein Skript. Das Ergebnis des Skripts wird allerdings nicht angezeigt, sondern in einem XML-Objekt gespeichert. Es müssen also XML-Daten zurückgeliefert werden.	<pre>var x = new XML("<a>b"); var y = new XML(); x.sendAndLoad("skript.asp", y); /* y enthält den (XML-)Rückgabewert des Skripts */</pre>
<code>toString() *</code>	Wandelt die XML-Daten des Knotens in eine Zeichenkette um. Der Aufruf ist normalerweise unnötig, die Funktion wird automatisch z. B. bei Ausgabeversuchen mittels <code>trace()</code> aufgerufen.	<pre>var x = new XML("<a>b"); trace(x.toString()); /* Ausgabe: <a>b */</pre>

Tabelle 18.1 Die Methoden des XML-Objekts (Forts.)



Noch ein Hinweis zur Methode `send()`: Je nach Skriptsprache können Sie anders auf die Daten zugreifen, die per POST an das Skript geschickt worden sind. Bei ASP beispielsweise geht das mit `Request.BinaryRead()`.

Wir fahren mit den Eigenschaften des XML-Objekts fort:

Eigenschaft	Lesen/Schreiben	Beschreibung	Beispiel
<code>attributes[] *</code>	L/S	Alle Attribute des aktuellen Knotens	<pre>var x = new XML(); var k = x.createElement("a"); k.attributes["b"] = "c"; x.appendChild(k); trace(x); /* Ausgabe: /> */</pre>

Tabelle 18.2 Die Eigenschaften des XML-Objekts

Eigenschaft	Lesen/Schreiben	Beschreibung	Beispiel
childNodes[] *	L	Array aus allen Kindknoten eines Knotens	<pre> var x = new XML("<a>b"); var k = x.childNo- des[0]; var l = k.childNo- des[0]; trace(l); /* Ausgabe: b (denn der erste Kindknoten ist <a />, und dessen erster Kind(text)kno- ten ist "b") */ </pre>
contentType	L/S	MIME-Typ der XML-Daten, wenn sie per send() oder sendAndLoad() verschickt werden	<pre> var x = new XML(); trace(x.contentType); /* Ausgabe: applica- tion/x-www-form- urlencoded */ </pre>
docTypeDecl	L/S	DOCTYPE-Deklara- tion des XML-Doku- ments	<pre> var x = new XML(); x.docTypeDecl = "<!DOCTYPE x SYSTEM \ 'y.dtd\">"; </pre>
firstChild *	L/S	Erster Kindknoten; entspricht childNo- des[0]	<pre> var x = new XML("<a>b"); var k = x.firstChild; var l = k.firstChild; trace(l); /* Ausgabe: b (siehe Beispiel zu childNo- des[]) */ </pre>
ignoreWhite	L/S	Legt fest, ob Kno- ten, die nur aus »Whitespace« (Leer- zeichen, Zeilen- sprüngen etc.) bestehen, ignoriert werden sollen (true, empfohlen) oder nicht (false)	<pre> var x = new XML(); x.ignoreWhite = true; </pre>

Tabelle 18.2 Die Eigenschaften des XML-Objekts (Forts.)

Eigenschaft	Lesen/Schreiben	Beschreibung	Beispiel
lastChild *	L	Letzter Kindknoten eines Knotens; entspricht childNodes[childNodes.length - 1] – denn die Array-Zählung beginnt bei 0	<pre>var x = new XML("<a><c />"); var k = x.firstChild; var l = k.lastChild; trace(l); /* Ausgabe: <c /> */</pre>
loaded	L	Hat einen von drei Werten: normalerweise undefined; beim Aufruf von load() oder loadAndSend() false, nach dem Vollständigen Laden durch load() oder loadAndSend() true	<pre>/* Bild 1 */ var x = new XML(); x.load("datei.xml"); /* Bild 3 */ if (x.loaded) trace("ok"); else gotoAndPlay(2);</pre>
nextSibling *	L	Der Knoten neben dem aktuellen Knoten (auf derselben Hierarchieebene)	<pre>var x = new XML("<a><c />"); var k = x.firstChild; var l = k.firstChild; var m = l.nextSibling; trace(m); /* Ausgabe: <c /> */</pre>
nodeName *	L/S	Name des Knotens, sofern es ein Tag ist und kein Textknoten	<pre>var x = new XML("<a>b"); var k = x.firstChild; var l = k.firstChild; trace(k.nodeName); /* Ausgabe: a */ trace(l.nodeName); /* Ausgabe: null */</pre>
nodeType *	L	Typ des Knotens als Integerwert. Bei 1 ist es ein Element (also ein Tag), bei 3 ein Textknoten.	<pre>var x = new XML("<a>b"); var k = x.firstChild; var l = k.firstChild; trace(k.nodeType); /* Ausgabe: 1 */ trace(l.nodeType); /* Ausgabe: 3 */</pre>

Tabelle 18.2 Die Eigenschaften des XML-Objekts (Forts.)

Eigenschaft	Lesen/Schreiben	Beschreibung	Beispiel
nodeValue *	L/S	Der Wert (Text) in einem Textknoten; null bei einem Element	<pre>var x = new XML("<a>b"); var k = x.firstChild; var l = k.firstChild; trace(k.nodeValue); /* Ausgabe: null */ trace(l.nodeValue); /* Ausgabe: b */</pre>
parentNode *	L	Der Elternknoten des aktuellen Knotens (oder null, falls es sich um das Root-Element handelt)	<pre>var x = new XML("<a />"); var k = x.firstChild; trace(x.parentNode); /* Ausgabe: null */ trace(k.parentNode); /* Ausgabe: <a /> */</pre>
previousSibling *	L	Der dem aktuellen Knoten auf derselben Hierarchieebene vorangehende Knoten	<pre>var x = new XML("<a><c />"); var k = x.firstChild; var l = k.lastChild; var m = l.previousSibling; trace(m); /* Ausgabe: */</pre>
status	L	Statuscode des XML-Parsers (der Parser wird aufgerufen bei load(), sendAndLoad(), new XML() und parseXML()). 0 bedeutet kein Fehler, -2 bis -10 Fehler.	<pre>var x = new XML("<a>"); trace(x.status); /* Ausgabe: -9 */</pre>
xmlDecl	L/S	Die XML-Deklaration des Dokuments	<pre>var x = new XML(); x.xmlDecl = "<?xml version='1.0'?'>";</pre>

Tabelle 18.2 Die Eigenschaften des XML-Objekts (Forts.)

Damit haben Sie einen beinahe kompletten Überblick über die Funktionalität des XML-Objekts erhalten.

18.3 Häufige Anwendungen

In diesem Abschnitt werden wir die bereits vorgestellten Funktionen in der Praxis einsetzen. Dazu werfen wir einen Blick auf Anwendungen, die Sie beim Arbeiten mit XML-Dokumenten häufig benötigen. Ganz nebenbei demonstrieren wir damit die Methoden und Eigenschaften des XML-Objekts von Flash im Einsatz.

Damit wir auch etwas Handfestes zum Experimentieren haben, benötigen wir einen beispielhaften XML-Datensatz. Wir erweitern dazu die XML-Datei, die uns schon im vorangegangenen Abschnitt gute Dienste geleistet hat. Damit wir nicht dauernd über Gebühr Schleichwerbung für CNN machen, verweisen wir diesmal auf die Internet-Angebote zweier großer deutscher Nachrichtenmagazine, die montags erscheinen:

```
<?xml version="1.0"?>
<rss version="0.91">
  <channel>
    <item>
      <title>DER SPIEGEL</title>
      <link>http://www.spiegel.de/</link>
    </item>
    <item>
      <title>FOCUS</title>
      <link>http://www.focus.de/</link>
    </item>
  </channel>
</rss>
```

Tabelle 18.3 Die etwas erweiterte Beispieldatei



Sie finden diese Datei auf der CD-ROM zum Buch unter dem Dateinamen **test.xml**.

18.3.1 Daten laden

Bei der Vorstellung der Methode `load()` haben wir bereits kurz angedeutet, wie Sie den **asynchronen** Ladevorgang programmtechnisch erfassen können. Sie laden den Film per `load()` und warten dann in einer Schleife, bis die Eigenschaft `loaded` den Wert `true` hat. Hier noch einmal der Aufbau eines solchen Beispielfilms, diesmal auch unter Berücksichtigung der Eigenschaft `status` (es kann ja auch ein Fehler in der XML-Datei vorliegen).

In Bild 1 des Films wird das XML-Objekt erzeugt und die Methode `load()` aufgerufen:

```
var x = new XML();  
x.load("test.xml");
```

Bild 4 enthält eine Statusmeldung, dass alles geklappt hat; Bild 5 eine Fehlermeldung. Die Bilder 2 und 3 sind für eine Schleife reserviert, die den Ladevorgang überprüft beziehungsweise diesen abwartet. Dazu ist Bild 2 leer (d.h. es enthält keinen Skriptcode), Bild 3 ist ein neues Schlüsselbild und enthält den folgenden Code:

```
if (x.load) {  
    if (x.status == 0) {  
        gotoAndStop(4); // ok  
    } else {  
        gotoAndStop(5); // Fehler  
    }  
} else {  
    gotoAndPlay(2);  
}
```

In Bild 4 wird außerdem noch der Inhalt des XML-Objekts in einem dynamischen Textfeld ausgegeben:

```
XMLFeld = x.toString();
```

Sie finden den Film auf der CD-ROM zum Buch unter dem Dateinamen **load-1 fla**. Wie der Film bei ordnungsgemäßigem Laden ausgegeben wird, können Sie Abbildung 18.4 entnehmen.

Eine zweite Möglichkeit besteht darin, den `onLoad`-Event-Handler des XML-Objekts zu setzen. Damit können Sie eine Funktion angeben, die ausgeführt wird, sobald die XML-Daten geladen worden sind:

```
stop();  
  
var x = new XML();  
x.ignoreWhite = true;  
x.onLoad = geladen;  
x.load("test.xml");
```



Abbildung 18.4 Die XML-Daten wurden eingelesen.

Der Film wird also angehalten, dann werden die XML-Daten geladen – Whitespace wird dabei ignoriert. Die Funktion `geladen()` wird ausgeführt, sobald die Daten alle angekommen sind. Hier wird wieder verzweigt: Bei Erfolg springt die Funktion zu Bild 2 (positive Statusmeldung nebst Ausgabe des XML-Codes), bei Misserfolg zu Bild 3 (Fehlermeldung):

```
function geladen(status) {
    if (status)
        gotoAndStop(2);
    else
        gotoAndStop(3);
}
```



An die Funktion, die durch `onLoad` aufgerufen wird, wird als Parameter der Ladestatus übergeben. Sie müssen also nicht extra `x.status` abfragen, der Wert steht direkt als Parameter zur Verfügung.



Um genau zu sein: `onLoad` wird ausgeführt, sobald die Daten geladen und geparkt worden sind. Direkt nach dem Laden kommt zunächst `onData` zum Zug, was aber in der Praxis eher selten eingesetzt wird.

In Abbildung 18.5 sehen Sie die Ausgabe bei erfolgreichem Laden. Sie erkennen hier auch die Wirkung von `ignoreWhite = true`: Zeilensprünge und Einrückungen durch Leerzeichen sind verschwunden.

Auch allgemein ist `ignoreWhite = true` zu empfehlen. Wenn Sie beispielsweise die Elemente `<a />` `<b />` haben, wäre `nextSibling` von `<a />` keineswegs `<b />`, sondern ein Textknoten, der aus einem Leerzeichen besteht.



Abbildung 18.5 Die Daten werden ausgegeben – ohne Whitespace.

Dieser Film – auf der CD-ROM zum Buch unter dem Dateinamen **load-2.fla** – benötigt deutlich weniger Code und auch zwei Bilder weniger. Der Weg über `onLoad` ist also empfehlenswert: Sie müssen nicht von Hand eine Flash-Schleife produzieren und auf das Ergebnis warten, denn das erledigt Flash im Hintergrund für Sie.

18.3.2 Elemente finden

Oftmals wissen Sie zwar, welches Element Sie benötigen, aber nicht, wo es sich befindet. Nehmen wir zum Beispiel unsere Datei **test.xml**. Sie wollen – etwa für ein Werbefbanner – alle `<text>`-Elemente oder alle `<link>`-Elemente aus der XML-Datei ermitteln. Sie können nun einen Blick auf die Struktur werfen und Folgendes feststellen:

- Nach dem Laden haben Sie ein XML-Dokument, wie zuvor wohl wieder in einer Variablen `x`.
- `x.firstChild` ist der erste (und einzige) Kindknoten, also das `<rss>`-Element.
- `x.firstChild.firstChild` ist dann das erste Element innerhalb von `<rss>`, also `<channel>`.
- Das `<channel>`-Element hat zwei `<item>`-Elemente als Unterelemente, auf die Sie über `x.firstChild.firstChild.childNodes[0]` und `x.firstChild.firstChild.childNodes[1]` zugreifen können. Allgemein: Sie greifen auf alle Elemente des Arrays `x.firstChild.firstChild.childNodes[]` zu.
- Für jedes der `<item>`-Elemente ist `childNodes[0]` das `<title>`-Element, `childNodes[1]` das `<link>`-Element. Bei jedem dieser Elemente haben Sie über `firstChild` Zugriff auf den Textknoten und damit auf den Inhalt des Elements.

Wenn Sie also beispielsweise alle `<title>`-Knoten ausgeben möchten, müssen Sie in einer Schleife das XML-Dokument durchlaufen:

```
var items = x.firstChild.firstChild.childNodes;
for (var i=0; i<items.length; i++)
    trace(items[i].childNodes[0].firstChild);
```

Sie sehen – das ist ganz schön aufwändig. Und wir haben sogar das `toString()` weggelassen! Nun sind wir aber ganz strikt davon ausgegangen, dass `<title>` das erste Element innerhalb von `<item>` ist. Am Inhalt der XML-Datei ändert sich allerdings nichts, wenn beispielsweise die Reihenfolge von `<text>` und `<link>` vertauscht wird. Sie müssen also über `nodeName` überprüfen, ob Sie den richtigen Kindknoten von `<item>` haben:

```
var items = x.firstChild.firstChild.childNodes;
for (var i=0; i<items.length; i++)
    for (var j=0; j<items[i].childNodes.length; j++)
        if (items[i].childNodes[j].nodeName == "title")
            trace(items[i].childNodes[j].firstChild);
```

Das Verfahren wird immer komplizierter und damit auch fehleranfälliger; ein Versehen ist bei den ganzen Ansammlungen von `childNodes[]` und `firstChild` schnell passiert.

Aus diesem Grund ist es eine gute Idee, hierfür eine eigene Funktion zu schreiben. Hierzu setzen wir einen **rekursiven** Programmierungsansatz ein. Die Funktion ruft sich nämlich bei Bedarf selbst wieder auf.

Die Vorgehensweise ist recht einfach. Wir verwenden ein globales Array, das später alle Elemente mit dem gewünschten Namen (`nodeName`) enthalten wird.

```
var e = new Array();
```

Nun zur Funktion. Beginnend beim Hauptknoten werden alle Kindknoten untersucht. Als Parameter werden der gewünschte Knotenname sowie eine Referenz auf den Knoten an die Funktion übergeben:

```
function suche(name, knoten) {
```

Wenn einer der Knoten den gewünschten Namen hat, wird er in das Array gelegt:

```
    for (var i=0; i<knoten.childNodes.length; i++)
        if (knoten.childNodes[i].nodeName == name)
            e.push(knoten.childNodes[i]);
```

Hat keiner der Knoten den gewünschten Namen, wird die Funktion erneut aufgerufen, damit alle Kindknoten des aktuell betrachteten Knotens ebenfalls untersucht werden können:

```
        else
            suche(name, knoten.childNodes[i]);
    }
```

Um nun beispielsweise alle `<link>`-Elemente zu finden, muss diese Funktion wie folgt aufgerufen werden:

```
suche("link", x);
```

Dabei gehen wir davon aus, dass `x` das XML-Dokument ist. Das Array `e` wird nun gefüllt und Sie können später das Array ausgeben:

```
XMLFeld = e.join("\n");
```

Hier sehen Sie noch einmal das komplette Skript:

```
var e = new Array();
```

```
function suche(name, knoten) {
    for (var i=0; i<knoten.childNodes.length; i++)
        if (knoten.childNodes[i].nodeName == name)
            e.push(knoten.childNodes[i]);
        else
            suche(name, knoten.childNodes[i]);
}
```

```
}
```

```
suche("link", x);  
XMLFeld = e.join("\n");
```

Listing 18.3 Das Bildskript für Bild 2

In dem Film **elementsuche.fla** ist dieses Verfahren einmal umgesetzt. Auf der Basis des vorherigen Beispiels, **load-2.fla**, wird die XML-Datei geladen. Anstatt aber den gesamten Dateiinhalt auszugeben, werden im zweiten Bild mit der oben gezeigten Funktion alle `<link>`-Elemente ausgegeben (siehe Abbildung 18.6).



Abbildung 18.6 Alle gesuchten Elemente werden ausgegeben.



Wenn Sie nur den Textinhalt ausgeben möchten, müssen Sie folgenden Code verwenden:

```
for (var el in e)  
    XMLFeld += el.firstChild.toString() + "\n";
```

18.3.3 XML formatiert ausgeben

Als letztes kleines Beispiel wollen wir einen XML-Viewer basteln. Dazu wird eine XML-Datei eingelesen und formatiert ausgegeben. »Formatiert« bedeutet hier Folgendes:

- Eine anzugebende XML-Datei wird eingelesen, Whitespace dabei ignoriert.
- Die Datei wird dann formatiert ausgegeben. Dabei wird jede Ebene um zwei Leerzeichen in Bezug auf die übergeordnete Ebene eingerückt.

Der praktische Nutzen dieses Beispiels ist ziemlich begrenzt; wir können jedoch an diesem Beispiel einige wichtige Konzepte für den Umgang mit XML-Daten und auch den einen oder anderen Kniff in Flash vorführen.

Beginnen wir mit dem Code zum Einlesen. Zunächst wird wie in den vorangegangenen Beispielen ein XML-Objekt instanziiert und der Programmcode zum Abwarten des Ladevorgangs untergebracht:

```
stop();

var x = new XML();
x.ignoreWhite = true;
x.onLoad = geladen;

function geladen(status) {
    if (status)
        gotoAndStop(2);
    else
        gotoAndStop(3);
}
```

Im ersten Bild wird ein Eingabetextfeld mit dem Variablennamen `Dateiname` erstellt; in diesem kann der Benutzer den Namen der gewünschten XML-Datei angeben (wir geben aus Bequemlichkeitsgründen **test.xml** vor). Zusätzlich definieren wir eine Schaltfläche mit der Beschriftung **Laden**, die dann den Ladevorgang auslösen wird.

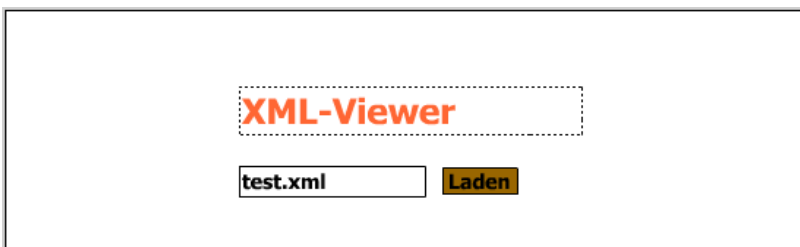


Abbildung 18.7 So sieht das erste Bild im Film aus.

Mit dieser Schaltfläche (Vorgehensweise: statisches Textfeld im Hauptfilm, gefülltes Rechteck im Hintergrund als neues Schaltflächensymbol) verbinden Sie die folgende Aktion:

```
on (release) {  
    x.load(Dateiname);  
    Ueberschrift = "Lade Daten ...";  
}
```

Nach dem Anklicken der Schaltfläche laden Sie die angegebene Datei und geben eine entsprechende Meldung in der Überschrift (die zurzeit noch mit »XML-Viewer« vorbelegt ist) aus.

So weit, so gut. Jetzt kommen wir allerdings zum komplexen Teil der Anwendung. Ziel ist es, eine Funktion zu schreiben, die XML-Daten formatiert ausgibt. Auch hier verfolgen wir wieder einen rekursiven Programmieransatz. Für jede Ebene in der XML-Datei wird die Funktion extra aufgerufen.

Wir beginnen mit dem Funktionskopf. Dass als Parameter eine Referenz auf den aktuellen Knoten übergeben werden muss, ist klar. Es wird jedoch zusätzlich noch ein zweiter Parameter benötigt, der angibt, auf welcher Ebene in der Hierarchie sich das aktuelle Element gerade befindet. Sie erinnern sich – pro Ebene soll um eine bestimmte Anzahl von Leerzeichen eingerückt werden.

```
function ausgabe(knoten, level) {
```

Beginnen wir gleich mit der Einrückung – pro Ebene zwei Leerzeichen:

```
    var einrueckung = "";  
    for (var i=1; i<=level; i++)  
        einrueckung += "  ";
```

In der Variablen `a` soll der Rückgabewert der Funktion gespeichert werden. Zunächst muss die bereits definierte Einrückung erfolgen:

```
    var a = einrueckung;
```

Nun ist es wichtig festzustellen, ob ein Textknoten (`nodeType` ist 3) oder ein Element (`nodeType` ist 1) vorliegt. Beginnen wir mit dem Element. In diesem Fall muss der Elementname (`nodeName`) in spitzen Klammern ausgegeben werden:

```
    if (knoten.nodeType == 1) {  
        a += "<" + knoten.nodeName;
```

Die spitze Klammer wird allerdings noch nicht geschlossen; zunächst müssen wir überprüfen, ob das Element Attribute hat. Falls ja, werden diese mit ausgegeben (per for-in-Schleife). Erst dann darf das Tag geschlossen werden.

```
for (var j in knoten.attributes)
    a += " " + j + "=\"" + knoten.attributes[j] + "\"";
a += ">\n";
```

Wenn dagegen ein Textknoten vorliegt, wird dessen Inhalt (`nodeValue`) in `a` abgelegt:

```
else
    a += knoten.nodeValue + "\n";
```

Nun kommt die Rekursion ins Spiel. Bis jetzt hat `a` ungefähr folgenden Wert:

```
<tagname attribut1="wert1" attribut2="wert2">
```

Nun müssen alle Kindknoten des aktuellen Knotens betrachtet werden; für jeden wird `ausgabe()` aufgerufen. Das Hierarchielevel wird dabei um 1 erhöht, damit die Kindelemente um zwei Leerzeichen mehr eingerückt werden. Die Rückgabewerte dieser Aufrufe von `ausgabe()` werden an `a` angehängt:

```
for (i=0; i<knoten.childNodes.length; i++)
    a += ausgabe(knoten.childNodes[i], level+1);
```

Damit sind wir beinahe fertig. Wenn der aktuelle Knoten ein Element ist, muss das Tag noch geschlossen werden. Am Ende wird `a` zurückgegeben:

```
if (knoten.nodeType == 1)
    a += einrueckung + "</" + knoten.nodeName + ">\n";
return a;
}
```

Ein Spezialfall wurde bis dato noch völlig außer Acht gelassen. Wenn die Funktion `ausgabe()` aufgerufen und dabei ein XML-Objekt (also kein `XMLNode`-Objekt) übergeben wird, befinden wir uns auf der obersten Ebene des XML-Dokuments. Das heißt `nodeType` ist 1, aber `nodeName` ist `null`. In diesem Fall müssen Sie die folgenden Schritte ausführen:

- Zunächst wird ermittelt, ob das XML-Dokument eine XML-Deklaration (`<?xml ... ?>`) enthält: Falls ja, wird diese ausgegeben.

```
if (knoten.xmlDecl != undefined)
    a += knoten.xmlDecl + "\n";
```

- Gleiches gilt für eine DOCTYPE-Deklaration; auch diese wird – sofern vorhanden – ausgegeben.

```
if (knoten.docTypeDecl != undefined)
    a += knoten.docTypeDecl + "\n";
```

- Danach wird `ausgabe()` für den ersten (und einzigen) Kindknoten aufgerufen, dieser ist dann das Root-Element des XML-Dokuments. Als Level wird weiterhin 0 verwendet.

```
return a + ausgabe(x.firstChild, 0);
```



Sie müssen im letzten Schritt unbedingt `return` verwenden; andernfalls würde der Rest der Funktion (Durchlaufen aller Kindknoten) ebenfalls ausgeführt werden, was beim Root-Element nicht erwünscht ist.

Nun müssen Sie nur noch feststellen, wann Sie ein Root-Element vor sich haben. Nun, dies ist nur der Fall, wenn der aktuelle Knoten keinen übergeordneten Knoten besitzt:

```
if (knoten.parentNode == null) {
    // Code von oben
}
```

Sie müssen nun im zweiten Bild des Films nur noch die Funktion `ausgabe()` aufrufen und eine Referenz auf das XML-Objekt übergeben. Den Rückgabewert weisen Sie einem mehrzeiligen dynamischen Textfeld zu:

```
XMLFeld = ausgabe(x, 0);
```

Nachfolgend finden Sie noch einmal den kompletten Code für das zweite Bild im Zusammenhang; der Film befindet sich auf der CD-ROM zum Buch unter dem Dateinamen **ausgabe fla**.

```
function ausgabe(knoten, level) {
    var einrueckung = "";
    for (var i=1; i<=level; i++)
        einrueckung += "  ";

    var a = einrueckung;
    if (knoten.parentNode == null) {
        if (knoten.xmlDecl != undefined)
            a += knoten.xmlDecl + "\n";
        if (knoten.docTypeDecl != undefined)
            a += knoten.docTypeDecl + "\n";
        return a + ausgabe(x.firstChild, 0);
    }
```

```

    }

    if (knoten.nodeType == 1) {
        a += "<" + knoten.nodeName;
        for (var j in knoten.attributes)
            a += " " + j + "=\"" + knoten.attributes[j] + "\"";
        a += ">\n";
    }
    else
        a += knoten.nodeValue + "\n";

    for (i=0; i<knoten.childNodes.length; i++)
        a += ausgabe(knoten.childNodes[i], level+1);

    if (knoten.nodeType == 1)
        a += einrueckung + "</" + knoten.nodeName + ">\n";
    return a;
}

XMLFeld = ausgabe(x, 0);

```

Listing 18.4 Das Bildskript für Bild 2

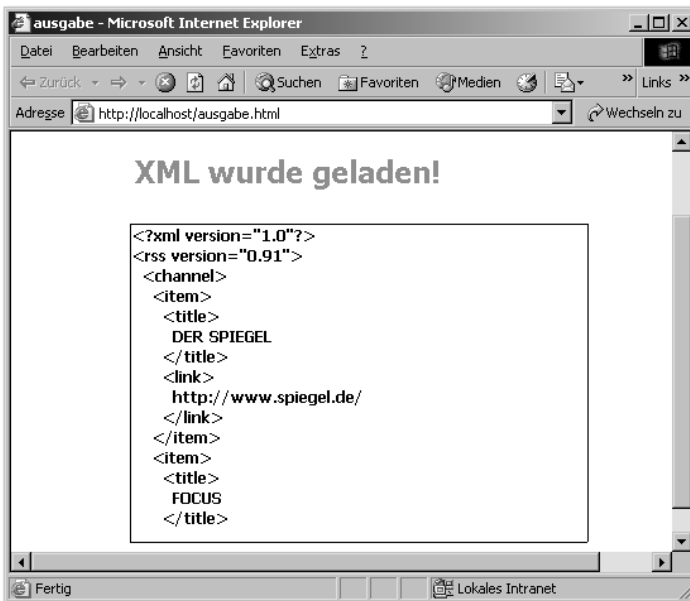


Abbildung 18.8 Die XML-Datei wird formatiert ausgegeben.

Ein kleiner Nachteil dieses Skripts wird in Abbildung 18.8 offenkundig. Bei zu großen XML-Dateien muss gescrollt werden, es wird aber kein Scrollbalken angezeigt. Bis einschließlich Flash 5 war das nur mit Mühen oder externen Komponenten möglich. In Flash MX wurden jedoch die so genannten Flash UI Components eingeführt. Damit lassen sich bequem (unter anderem) Scrollbalken erstellen, allerdings können diese nur mit Flash MX bearbeitet und mit dem Flash-MX-Plug-In angezeigt werden.

Um dies zu realisieren, sind einige kleinere Umbauarbeiten im zweiten Bild des Films notwendig. Rufen Sie zunächst die Komponentenbibliothek von Flash auf (**Fenster • Komponenten** oder **Strg + F7**). Ziehen Sie das mit **ScrollBar** beschriftete Element auf die Bühne, in das dynamische Textfeld für die Ausgabe hinein. Der Scrollbalken passt sich automatisch an die Größe des Textfeldes an.

Nun müssen Sie dem dynamischen Textfeld (im Eigenschafteninspektor) einen Instanznamen geben; verwenden Sie beispielsweise `FeldInstanz`. Bei den Eigenschaften des Scrollbalkens geben Sie unter **Target Text-Field** den gerade gewählten Instanznamen, bei uns `FeldInstanz`, an. Damit ist der Scrollbalken fest mit dem Textfeld verbunden. Wenn Sie jetzt den Film starten, sehen Sie – nichts. Der Scrollbalken erscheint zwar, ist aber offensichtlich inaktiv (siehe Abbildung 18.9).



Abbildung 18.9 Der Scrollbalken ist inaktiv.

Des Rätsels Lösung: Da der Scrollbalken an den Instanznamen des dynamischen Textfelds gebunden ist, müssen Sie auch den Text an den Instanznamen binden. Sie dürfen also nicht `XMLFeld` den formatierten XML-Code übergeben, sondern müssen auf die Instanz `FeldInstanz` zugreifen. Deren Eigenschaft `text` erlaubt den Zugriff auf den Inhalt des Textfelds.

Die Anweisung, die notwendig ist, um den Text zuzuweisen und die Scrollbalken zu aktivieren, lautet also folgendermaßen:

```
FeldInstanz.text = ausgabe(x, 0);
```

Den Effekt dieser Anweisung – nämlich den plötzlich funktionsfähigen Scrollbalken – sehen Sie in Abbildung 18.10; den zugehörigen Film finden Sie auf der CD-ROM zum Buch unter dem Dateinamen **ausgabe-mx.flx**; er läuft aber nur unter Flash MX.



Abbildung 18.10 Jetzt funktioniert der Scrollbalken.

In diesem Abschnitt haben Sie einige häufige Aufgabenstellungen beim Umgang mit XML gesehen und erfahren, wie Sie diese lösen können. Dabei haben wir Ihnen auch die wichtigsten Eigenschaften und Funk-

tionen der XML-Objekte von Flash im Einsatz gezeigt. Höchste Zeit, ein komplettes praxisrelevantes Projekt durchzuführen!

18.4 Beispiel: RSS-Newsbanner

Wir haben in diesem Kapitel als Beispiel für eine XML-Datei nicht umsonst zunächst eine Datei **cnn.rss** vom CNN-Server und später Modifikationen und Erweiterungen dieser Datei verwendet. Es handelt sich hierbei nicht um irgendeine Datei, die »zufällig« wohlgeformtes XML enthält. Statt dessen ist XML ein eigenes, standardisiertes Dateiformat, mit dem **Content Syndication** betrieben wird.

Hinter diesem Schlagwort verbirgt sich ein Konzept, mithilfe dessen neueste Schlagzeilen leicht verbreitet werden können. RSS ist ein Dateiformat, in dem solche Meldungen samt zugehörigen Links abgelegt werden können. Viele Newsdienste im World Wide Web bieten ihre Nachrichten parallel im RSS-Format an. Damit können zwei weitere Zielgruppen von dem Rechercheaufwand der jeweiligen Redaktion profitieren:

- ▶ Spezielle Softwareprodukte, die RSS-Dateien auslesen können, ermöglichen einen bequemen, softwaregesteuerten Zugriff auf diese Dateien; ohne Webbrowser können schnell die wichtigsten Neuigkeiten abgegriffen werden.
- ▶ Andere Site-Betreiber können diese Nachrichten in ihre Seiten integrieren, sofern der Betreiber der Site mit den Original-News dies gestattet. Oftmals muss für dieses Nutzungsrecht auch ein Obolus entrichtet werden. Mit einem Wort (beziehungsweise zwei Wörtern): **Content Syndication**.

Das RSS-Format wurde ursprünglich von der Firma **Netscape** entwickelt; mittlerweile tüfteln zwei Institutionen parallel daran weiter: Netscape und eine Firma **UserLand**, von deren Dateiformat **<scriptingNews>** Netscape bei der ersten Überarbeitung der RSS-Spezifikation einiges übernommen hatte. Es gibt mittlerweile eine eigenständige Entwicklergemeinschaft RSS-DEV, die inzwischen RSS Version 1.0 veröffentlicht hat, während die aktuellste UserLand-Version die Nummer 0.92 trägt. Am weitesten verbreitet ist allerdings immer noch ein vom Juli 1999 datierender Netscape-Entwurf mit der Versionsnummer 0.91.



Weitere Informationen zu RSS finden Sie in den RSS-FAQ unter <http://www.voidstar.com/rssfaq>.

Werfen wir noch einmal einen Blick auf die RSS-Datei von CNN.com:

```
<?xml version="1.0"?>
<!DOCTYPE rss PUBLIC "-//Netscape Communications//DTD RSS 0.91//EN"
    "http://my.netscape.com/publish/formats/rss-0.91.dtd">
<rss version="0.91">
  <channel>
    <title>News from CNN.com</title>
    <link>http://cnn.com/index.html</link>
    <description>The world's news leader</description>
    <language>en-us</language>
    <item>
      <title>CNN.com</title>
      <link>http://cnn.com</link>
    </item>
  </channel>
</rss>
```

Diese Datei ist nicht gerade ein Vorbild in Sachen mannigfaltiger, sorgfältig recherchierter News, sie enthält lediglich einen Link auf die CNN-Homepage. Das war früher einmal anders, als Basis für die Analyse der Daten genügt diese Datei allerdings.

Wie wir bereits festgestellt haben, entspricht ein `<item>`-Element einer Meldung; über die Elemente `<title>` und `<link>` können der Meldungs(kurz)text sowie die dazugehörige URL ermittelt werden. Allerdings hat auch jedes `<channel>`-Element ein `<link>`-Element; zumeist die Homepage des Anbieters, oder, wenn es mehrere Newskategorien gibt, die Homepage der jeweiligen Kategorie.

Die Hauptschwierigkeit beim Interpretieren einer RSS-Datei besteht also darin, die richtigen `<link>`-Elemente zu finden. Mit der in Abschnitt 9 entwickelten Funktion `suche()` ist das mit relativ wenig Aufwand möglich. Folgende Vorgehensweise erscheint sinnvoll:

- ▶ Zunächst werden alle `<item>`-Elemente ermittelt.
- ▶ Innerhalb dieser `<item>`-Elemente werden dann `<title>` und `<link>` herausgesucht und weiterverarbeitet.

Programmtechnisch kann das dann in etwa wie folgt aussehen. Zunächst werden alle `<item>`-Elemente ermittelt:

```
var e = new Array();
suche("item", x);
```

Dieses Ergebnis wird in einer Variablen zwischengespeichert, damit die Variable `e` wieder frei ist für erneute Aufrufe von `suche()`. Jedes der `<item>`-Elemente wird dann nach `<title>` und `<link>` durchsucht:

```
var texte = new Array();
var links = new Array();
var items = e; // Ergebnis sichern
for (var j=0; j<items.length; j++) {
    e = new Array(); // Array e leeren
    suche("title", new XML(items[j]));
    texte.push((new XML(e[0])).firstChild.firstChild.nodeValue);
    e = new Array(); // Array e leeren
    suche("link", new XML(items[j]));
    links.push((new XML(e[0])).firstChild.firstChild.nodeValue);
}
```

Am Ende enthält das Array `texte[]` alle Newsmeldungen, `links[]` die dazugehörigen Links, und zwar in derselben Reihenfolge.

Der größte Aufwand in obigem Listing beruht auf der Tatsache, dass in den Arrays die entsprechenden XML-Knoten als Strings vorliegen und deswegen erst mit `new XML()` in ein XML-Objekt umgewandelt werden müssen. Die gefundenen `<title>`- und `<link>`-Elemente müssen zunächst ebenfalls in ein XML-Objekt konvertiert werden und dann muss `firstChild.firstChild.nodeValue` aufgerufen werden. Mit `firstChild` erhalten Sie das eigentliche Element, mit erneut `firstChild` den Textknoten und mit `nodeValue` dessen Inhalt.

Wir wollen nun als Beispielapplikation einen Film entwickeln, der ein dynamisches Werbefbanner darstellt. Aus einer RSS-Datei werden alle Mitteilungen ausgelesen und dann in einer Schleife angezeigt. Wir beginnen wie gehabt mit dem Einlesen der Datei in ein Flash-XML-Objekt:

```
stop();

var x = new XML();
x.ignoreWhite = true;
x.onLoad = geladen;
x.load("news.rss");
```

Nun wird eine Reihe von Hilfsvariablen deklariert:

```

var texte = new Array();
var links = new Array();
var e = new Array();

```

Sobald die XML-Daten geladen worden sind, wird der oben entwickelte Code ausgeführt (Funktion `geladen()`):

```

function geladen(status) {
    if (status) {
        suche("item", x);
        var items = e;
        for (var j=0; j<items.length; j++) {
            e = new Array(); // Array e leeren
            suche("title", new XML(items[j]));
            texte.push((new XML(e[0])).firstChild.firstChild.nodeValue);
            e = new Array(); // Array e leeren
            suche("link", new XML(items[j]));
            links.push((new XML(e[0])).firstChild.firstChild.nodeValue);
        }
        ZeigeNews();
        setInterval(ZeigeNews, 3000);
    }
}

```

Beachten Sie die letzten beiden Anweisungen: Eine Funktion `ZeigeNews()` wird ausgeführt; per Intervall wird dies alle drei Sekunden wiederholt. Bevor wir uns aber dieser Funktion zuwenden, werfen wir zunächst einen Blick auf den eigentlichen Film.

Da es sich um ein Werbefbanner handeln soll, verwenden wir als Filmgröße 468 * 60 Pixel, das ist ein Standard-Bannerformat im Web.

Der Film selbst besteht im Wesentlichen aus einem dynamischen Textfeld namens `Newsfeld`, welches die Tickermeldungen anzeigen wird. Achten Sie darauf, dass der Text nicht selektierbar ist, denn sonst können Sie später den Text nicht anklicken und damit einen Link aufrufen.

Definieren Sie eine Symbol-Schaltfläche in derselben Größe wie das Textfeld. Damit haben Sie später die Möglichkeit, Mausklicks auf das Textfeld per Skript abzufangen und dann den entsprechenden Link in einem Browserfenster zu öffnen.

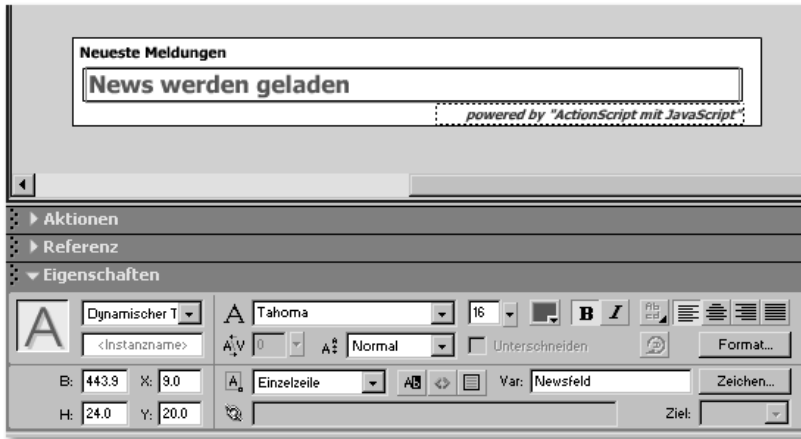


Abbildung 18.11 Das dynamische Textfeld im Eigenschafteninspektor

Doch nun zur Funktion `ZeigeNews()`, die alle drei Sekunden aufgerufen wird. In einer Variablen `pos` wird die aktuelle Position in den Arrays `texte/links` abgespeichert; dies ermöglicht es, auf die jeweils aktuelle Meldung zuzugreifen:

```
var pos = -1;
```

In der Funktion `ZeigeNews()` passiert nun Folgendes:

- Die Variable `pos` wird um 1 erhöht. Sollte damit das Ende des Arrays überschritten werden, wird sie auf 0 zurückgesetzt (das erste Element im Array hat den Index 0).
- Nun wird im Textfeld die entsprechende Meldung ausgegeben, also `texte[pos]`.

```
function ZeigeNews() {
    pos++;
    if (pos >= texte.length)
        pos = 0;
    Newsfeld = texte[pos];
}
```

Das Newsfeld – beziehungsweise die dahinter liegende Schaltfläche – wird nun mit einer Aktion verknüpft:

```
on (release) {
    FolgeLink();
}
```

Die Funktion `FolgeLink()` greift ebenfalls auf `pos` zu und ermittelt damit den entsprechenden Link aus dem Array `texte`; der Link wird mittels `getURL()` in einem neuen Fenster geöffnet:

```
function FolgeLink() {  
    if (pos >= 0)  
        getURL(links[pos], "_blank");  
}
```

Und damit ist der Film fertig. Nachfolgend sehen Sie noch einmal das komplette Bildskript für Bild 1 im Zusammenhang. Auf der CD-ROM zum Buch finden Sie den Film unter dem Dateinamen **rssbanner fla** sowie eine exemplarische RSS-Datei unter dem Namen **news.rss**.

```
stop();  
  
var x = new XML();  
x.ignoreWhite = true;  
x.onLoad = geladen;  
x.load("test.xml");  
  
var pos = -1;  
var texte = new Array();  
var links = new Array();  
var e = new Array();  
  
function geladen(status) {  
    if (status) {  
        suche("item", x);  
        var items = e;  
        for (var j=0; j<items.length; j++) {  
            e = new Array(); // Array e leeren  
            suche("title", new XML(items[j]));  
            texte.push((new XML(e[0])).firstChild.firstChild.nodeValue);  
            e = new Array(); // Array e leeren  
            suche("link", new XML(items[j]));  
            links.push((new XML(e[0])).firstChild.firstChild.nodeValue);  
        }  
        ZeigeNews();  
        setInterval(ZeigeNews, 3000);  
    }  
}
```

```

}

function ZeigeNews() {
    pos ++;
    if (pos >= texte.length)
        pos = 0;
    Newsfeld = texte[pos];
}

function FolgeLink() {
    if (pos >= 0)
        getURL(links[pos], "_blank");
}

function suche(name, knoten) {
    for (var i=0; i<knoten.childNodes.length; i++)
        if (knoten.childNodes[i].nodeName == name)
            e.push(knoten.childNodes[i]);
        else
            suche(name, knoten.childNodes[i]);
}

```

Das war jetzt zwar ein wenig aufwändig, dafür ist dieser Flash-Film aber in der Lage, beliebige RSS-Dateien einzulesen und anzuzeigen. Testen Sie doch einfach einmal eine der in den RSS-FAQ angegebenen Dateien. Aufgrund von Sicherheitsbeschränkungen im Flash-Player müssen Sie unter Umständen zunächst die RSS-Datei auf Ihren eigenen Server kopieren, um sie einsetzen zu können – probieren Sie es einfach aus. Beispielsweise rufen Sie mit folgendem Kommando den Newsfeed des bekannten Nachrichtenportals **Slashdot** auf:

```
x.load("http://slashdot.org/slashdot.rss");
```

Eine derart angepasste **.swf**-Datei finden Sie auf der CD-ROM zum Buch unter dem Dateinamen **rssbanner-slashdot.swf**. Im Flash-Player können Sie diese bei bestehender Internetverbindung abspielen.



Bevor Sie fremde RSS-Dateien auf Ihrer eigenen Website wiederverwenden, sollten Sie zunächst bei den Site-Inhabern deren Erlaubnis einholen!



Abbildung 18.12 Das Werbebanner im Flash-Player

Sie sehen also – XML ist ein sehr umfangreiches Thema, denn XML ist schier überall. Mit ein wenig Übung gewöhnen Sie sich jedoch schnell an den Zugriff auf die XML-Baumstruktur und können dann Beispiele wie den RSS-Parser schnell erstellen.

19 Spiele programmieren

*Das Spiel ist das einzige, was Männer wirklich ernst nehmen.
Deshalb sind Spielregeln älter als alle Gesetze der Welt.
– Peter Bamm*

Eines der wohl größten Probleme, dem sich Macromedia Flash gegenübergestellt sieht, ist das »Skip Intro«-Image. Zurzeit wird Flash hauptsächlich für nichtssagende Intros eingesetzt, aber nicht für sinnvolle Anwendungen. Oftmals besteht die einzige Interaktion, die hier möglich ist, in einem mit **Skip Intro** beschrifteten Link, der das Intro überspringt.

Dabei gibt es doch für Flash viel mehr Einsatzgebiete. Eines davon ist die Spieleprogrammierung. In den vorangegangenen Kapiteln haben Sie bereits die wichtigsten Elemente kennen gelernt:

- ▶ die Reaktion auf Tastaturereignisse,
- ▶ das Abspielen von Soundeffekten,
- ▶ den Einsatz von Movieclips.

In diesem Kapitel werden wir ein kleines Spiel entwickeln. Dabei zeigen wir Ihnen alle wichtigen Konzepte noch einmal auf. Damit haben Sie die Basis dafür, das Spiel zu erweitern oder auch selbst Flash-Spiele zu entwickeln.

Wir stellen den auf der CD-ROM befindlichen Beispielfilm **meteoritenha-gel.fl** vor und zeigen daran, wie wir die verschiedenen Flash-Techniken eingesetzt und zusammengeführt haben.

19.1 Grundlagen

An dieser Stelle wollen wir noch einmal die Grundlagen kurz zusammentragen, die wir für dieses Beispiel benötigen.

19.1.1 Tastatur abfragen

Wie Sie wissen, gibt es zwei Möglichkeiten, Tastatureingaben des Benutzers abzufragen:

1. über ein Symbol und die Aktion `on keypress`,
2. über das `Key`-Objekt.

Im Allgemeinen wird dem `Key`-Objekt der Vorzug gegeben. Damit können nämlich mehrere Tasten gleichzeitig abgefragt werden und auch Sondertasten wie etwa  oder  erfasst werden. Sie müssen dazu einfach eine der Methoden des `Key`-Objekts aufrufen, etwa folgende:

```
if (Key.isDown(Key.LEFT))  
    trace("Linke Pfeiltaste gedrückt");
```

In diesem Beispiel jedoch wenden wir die zweite Möglichkeit an, da wir in dem Spiel keine Sondertasten brauchen. Es wird darum gehen, mit einem kleinen Raumschiff einem Meteoritenhagel zu entgehen; als einzige Steuertasten dienen  und .

Die angesprochene zweite Möglichkeit verwendet die Aktion `on keypress`. Aktionen können nur mit Symbolen verknüpft werden, weswegen ein neues Symbol erstellt werden muss.

Es genügt hier völlig, ein unsichtbares Symbol zu verwenden, dessen einzige Aufgabe es ist, Tastatureingaben abzufragen. Befolgen Sie dabei die folgenden Schritte:

1. Erstellen Sie am besten eine eigene Ebene für das Symbol (optional).
2. Erstellen Sie ein neues Symbol (**Einfügen • Neues Symbol** oder  + ). Sie müssen dabei als Typ (**Verhalten**) die Option **Schaltfläche** auswählen (siehe Abbildung 19.1).
3. Sie landen im Editiermodus für die Schaltfläche. Ändern Sie nichts, sondern springen Sie wieder zum eigentlichen Film zurück.
4. Öffnen Sie die Bibliothek und ziehen Sie das gerade erstellte Symbol auf einen leeren, unbenutzten Bereich des Films.
5. Da die Schaltfläche keinen Inhalt hat, sehen Sie auf der Bühne keine Veränderung; im Eigenschafteninspektor jedoch bemerken Sie, dass die Schaltfläche aktiviert ist.
6. Gehen Sie zum Aktionenfenster; als Überschrift sehen Sie vermutlich etwas wie **Aktionen für [kein Instanzname zugewiesen] (Tastaturcheck)**.
7. Geben Sie das Skript zum Erfassen der Tastatureingaben an:

```
on (keypress "<Left>")  
    trace("Linke Pfeiltaste gedrückt");
```

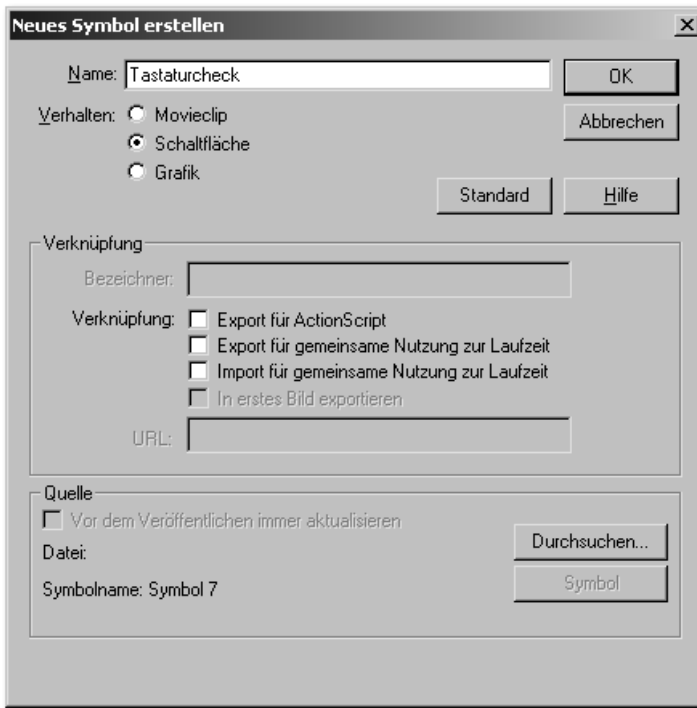


Abbildung 19.1 Die Einstellungen für das Symbol

19.1.2 Sounds abspielen

Sie können in Flash nicht nur Sound einbinden, sondern diesen auch je nach Bedarf abspielen. Wenn Sie nicht exklusiv für Flash MX (und das zugehörige, aber noch nicht so weit verbreitete Plug-In) entwickeln möchten, setzen Sie am besten das Sound-Objekt ein.

Dazu befolgen Sie folgende Schritte:

1. Importieren Sie die Sounddatei in die Bibliothek (**Datei • In Bibliothek importieren**).
2. Klicken Sie in der Bibliothek mit der rechten Maustaste den Sound an und wählen Sie den Menübefehl **Verknüpfung**.
3. Achten Sie darauf, dass **Export für ActionScript** aktiviert ist (siehe Abbildung 19.2).
4. Führen Sie an geeigneter Stelle im Film den folgenden Code aus, um die Sounddatei abzuspielen:

```
var soundobj = new Sound();
soundobj.attachSound("Soundbezeichner");
soundobj.start(0, 1);
```

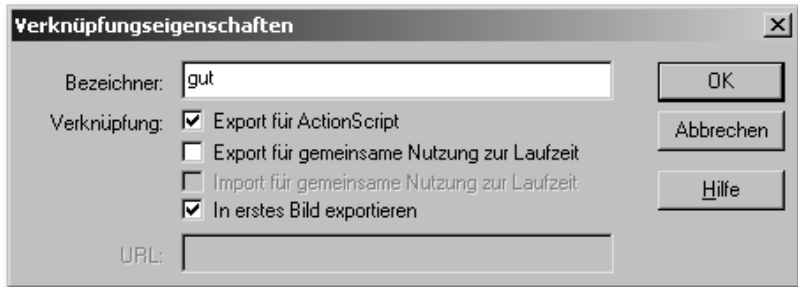


Abbildung 19.2 Export für ActionScript muss aktiviert werden.

19.1.3 Kollisionstest

Wichtiger Bestandteil von vielen Flash-basierten Spielen ist ein so genannter Kollisionstest. Sie können damit feststellen, ob ein Objekt mit einem anderen kollidiert, also zusammenstößt.

Jeder Movieclip besitzt die Methode `hitTest()`, mit der eine solche Überprüfung stattfinden kann. Sie müssen also auch die Objekte, mit denen Sie eine Kollision überprüfen möchten, in Movieclips stecken.

Es gibt zwei Möglichkeiten, diese Methode aufzurufen:

- Entweder Sie rufen die Methode auf und übergeben den Instanznamen des Movieclips, mit dem auf eine Kollision getestet werden soll:

```
if (this.hitTest(_root.Instanz))
    trace("Kollision festgestellt!");
```

- oder Sie übergeben drei Parameter an `hitTest()`:
 - die x-Koordinate des Kollisionsbereichs,
 - die y-Koordinate des Kollisionsbereichs,
 - den booleschen Wert, ob nur die Umrandung des Objekts auf Kollision überprüft werden soll (`false`) oder das gesamte Objekt (`true`).

```
if (this.hitText(_root._xmouse, _root._ymouse, true))
    trace("Maus befindet sich über dem Objekt!");
```

Vergessen Sie niemals die korrekte Referenz auf den Movieclip; im obigen Beispiel etwa müssen Sie mit `_root` auf den Hauptfilm zugreifen.



Der wohl beste Platz für diesen Kollisionstest ist in der Aktion `enterFrame` des Movieclips:



```
onClipEvent (enterFrame) {  
    // Aufruf von hitTest()  
}
```

In unserem Film, der ja das Raumschiff im Meteoritenhagel beinhaltet, können Sie nun feststellen, ob der Raumgleiter von einem Meteoriten getroffen worden ist.

19.2 Der Meteoritenhagel

Im Folgenden beschreiben wir nun die einzelnen Aspekte des Meteoritenhagels. Den kompletten Film finden Sie, wie eingangs erwähnt, auf der CD-ROM zum Buch unter dem Dateinamen **meteoritenhagel fla**.

19.2.1 Startbildschirm

Im ersten Bild des Films befindet sich ein eher profaner Startbildschirm. Dort werden die einfachen Regeln erklärt (das heißt die Tasten zur Steuerung aufgelistet). Außerdem muss der Benutzer per Mausklick das eigentliche Spiel starten.

Dieser letzte Schritt ist besonders wichtig, wenn das Spiel in einem Webbrowser ablaufen soll. Wenn der Flash-Film nicht angeklickt wird, werden Tastatureingaben nur an den Webbrowser, nicht jedoch an den integrierten Flash Player geschickt. Durch den Zwang, zum Spielstart einen Mausklick auszuführen, bringen Sie Ihre Besucher dazu, den Flash-Film anzuklicken. Noch viel wichtiger: Sie verpacken diese Notwendigkeit recht geschickt. Ein »Klicken Sie den (schon laufenden) Film an, damit Sie das Raumschiff steuern können« ist da die deutlich schlechtere Alternative.

In Abbildung 19.3 sehen Sie den Startbildschirm. Hinter dem Textfeld »Los« liegt eine Schaltfläche, die mit folgender Aktion verknüpft ist:

```
on (release) {  
    gotoAndPlay(2);  
}
```



Abbildung 19.3 Der Startbildschirm des Flash-Films

19.2.2 Die einzelnen Movieclips

Der Film selbst enthält eine Reihe von Movieclips. Da ist zunächst einmal das Raumschiff, dann der Meteorit. Das Raumschiff werden wir später einmal instanzieren, die Meteoriten mehrmals. Der dritte Movieclip ist eine kleine Grafik, welche für ein Leben steht. Am Anfang des Spiels hat der Spieler noch drei verfügbare Leben, nach jeder Kollision mit einem Meteoriten wird eines davon abgezogen. Rein grafisch äußert sich das darin, dass dann ein Lebenssymbol ausgeblendet wird. Dies wird möglich, wenn wir daraus einen Movieclip machen, denn dieser hat die Eigenschaft `_visible`. Der vierte Movieclip ist eine kleine Animation, die den Antrieb des Raumschiffs darstellen soll.

Nachfolgende Tabelle enthält die drei wichtigsten Movieclips in der Übersicht, inklusive der Abbildungen:

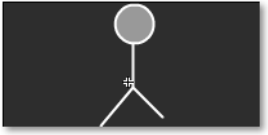
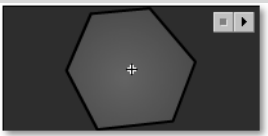
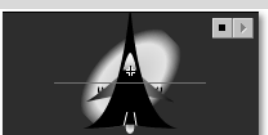
Name	Beschreibung	Abbildung
Leben	Grafik für die Anzeige der verbleibenden Leben	
Meteor	Grafik für die Meteoriten, denen das Raumschiff ausweichen muss	
Raumschiff	Grafik für das Raumschiff selbst	

Tabelle 19.1 Die Movieclips im Flash-Film

Beachten Sie insbesondere die Breite des Raumschiffs; aufgrund des Antriebs ist dieses recht breit, der Kollisionstest schlägt später also auch Alarm, wenn ein Meteorit vermeintlich dicht am Raumschiff vorbeifliegt.



Nun müssen diese Movieclips noch eingebunden werden. Erweitern Sie Bild 2 in der Zeitleiste nach rechts (klicken Sie ein Bild weiter rechts an und drücken Sie **[F5]**) und platzieren Sie drei Instanzen des Leben-Icons. Als Instanznamen vergeben Sie `Leben0`, `Leben1` und `Leben2`.

Im Film selbst platzieren Sie das Raumschiff, und zwar innerhalb eines Rechtecks, das die Begrenzungen des Spielfelds (oder in unserem Fall: des Weltalls) definiert.

Von den Meteoriten erstellen Sie fünf Instanzen zu je zehn Bildern. Erstellen Sie ein Bewegungs-Tween: Dazu platzieren Sie den Meteoriten im ersten Bild eines »Zehnerblocks« am oberen Ende des Rechtecks, im zehnten Bild am unteren Ende, und fügen dann ein Tweening ein (**Einfügen • Bewegungs-Tween erstellen**). Verwenden Sie für die fünf Instanzen (benannt von `Meteor1` bis `Meteor5`) jeweils horizontal versetzte Start- und Endpositionen. Achten Sie insbesondere darauf, dass es keine Stelle gibt, an der das Raumschiff jedem Meteoriten entkommen kann (sonst wäre das Spiel zu einfach). In unserem Film erstrecken sich die Tweenings von Bild 3 bis 12, 13 bis 22 und so weiter bis zum fünften Tweening, das sich von Bild 43 zu Bild 52 erstreckt.

In Abbildung 19.4 sehen Sie die Zeitleiste des Films mit den Tweenings, die wir in eine eigene Ebene gelegt haben, in Fehler! Verweisquelle konnte nicht gefunden werden. einen Ausschnitt aus der Bühne mit dem Raumschiff und einem der Meteoriten.

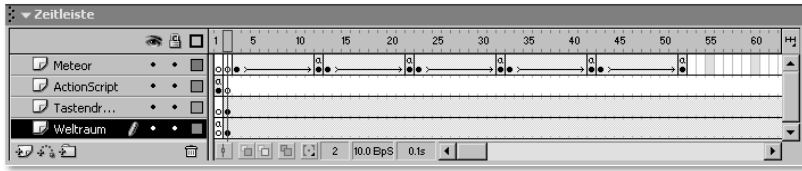


Abbildung 19.4 Die Zeitleiste des Films; Meteoriten in Ebene »Meteor«

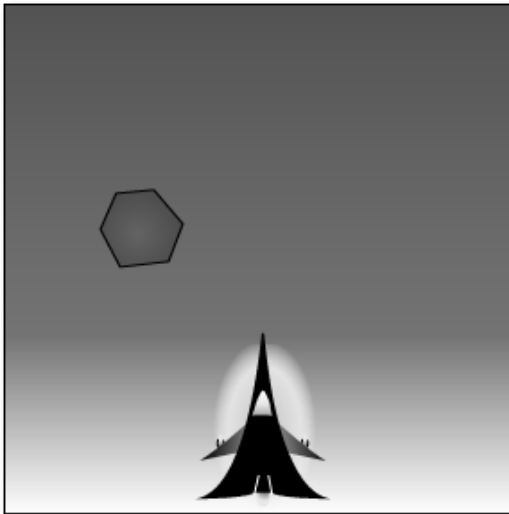


Abbildung 19.5 Das Raumschiff und ein Meteorit

19.2.3 Kollisionstest

Sie müssen nun noch das Raumschiff um etwas ActionScript-Code erweitern, damit Kollisionen mit den Meteoriten erfasst werden. Verknüpfen Sie also die Instanz des Raumgleiters mit folgendem Code:

```
onClipEvent (enterFrame) {
    if (this.hitTest(_root.Meteor1) && !_root.getroffen) {
        _root.blink();
        _root.gotoAndPlay(53);
    }
}
```

Kurz zur Erklärung: Wir definieren im Hauptfilm eine Variable getroffen, die angibt, ob ein Schiff gerade getroffen worden ist. Damit soll verhindert werden, dass ein Meteorit mehr als einen Treffer landen kann. Falls diese Variable den Wert `false` hat, wird zunächst die Funktion `blink()` im Hauptfilm aufgerufen (dazu kommen wir gleich), außerdem wird Bild 53 im Film angesprungen. Dort befinden sich zehn leere Bilder; in dieser Zeit kann sich der Benutzer vom Meteoriteneinschlag erholen und seine Kräfte sammeln.

19.2.4 Automatisches Hageln

Nun kommen wir, wie versprochen, zur Funktion `blink()`. Wie in einigen älteren Computerspielen üblich, blinkt das Raumschiff eine Zeit lang, sobald es getroffen worden ist. Wir realisieren diesen visuellen Effekt, indem wir die Eigenschaft `_visible` des Raumschiffs (beziehungsweise seines Movieclips) nacheinander auf `false` und dann wieder auf `true` setzen. Wie verwenden dazu `setInterval()` und wechseln insgesamt sechs Mal den Zustand:

```
i = 0;
ID = setInterval(function() {
    i++;
    if (i==6) clearInterval(ID);
    Gleiter._visible = !(Gleiter._visible);
}, 200);
```

Außerdem müssen wir dafür sorgen, dass in regelmäßigen Abständen ein neuer Meteorit auf seine Reise geschickt wird. Dies erledigt die Funktion `steinschlag()`.

Das Spiel besitzt mehrere Levels, Level 1 bis Level 5. Jedes Level stellt eine zusätzliche Herausforderung dar: Bei höheren Levels ist der Abstand zwischen zwei Meteoriten geringer. Die Verzögerung (in Millisekunden) wird nach folgender Formel berechnet – `zufall` ist ein zufälliger Wert zwischen 0 und 1-:

```
var verzoegerung = Math.floor((5 - level) * zufall) * 250;
```

In Level 1 kommt der nächste Meteorit also spätestens eine Sekunde, nachdem der vorherige Meteorit »durchgelaufen« ist; in Level 5 direkt danach (dann hat `verzoegerung` den Wert 0). Mit dieser Verzögerung wird dann eine zweite Funktion, `StarteMeteor()`, aufgerufen:

```

var ID2;
function steinschlag() {
    var zufall = Math.random();
    var verzoegerung = Math.floor((5 - level) * zufall) * 250;
    ID2 = setInterval(StarteMeteor, verzoegerung);
}

```

Die zweite Funktion schließlich wählt zufällig einen der fünf Meteoriten (oder eher: die dazugehörige Movieclip-Instanz) aus und startet diese:

```

function StarteMeteor() {
    var zufall = Math.random();
    var bildnr = 3 + 10 * Math.floor(5 * zufall);
    gotoAndPlay(bildnr);
    clearInterval(ID2);
}

```

Damit die Meteoriten auch wirklich nacheinander dem Raumschiff entgegenhageln, müssen Sie nun noch dafür sorgen, dass nach dem Ablauf eines Bewegungs-Tweens der nächste gestartet wird – mittels der Funktion `steinschlag()`. Dazu fügen Sie in das zehnte und letzte Bild jedes Zehnerblocks, in dem ein Tween untergebracht ist, folgenden Code ein:

```

stop();
_root.steinschlag();

```

19.2.5 Zählen

Das Spiel muss ja auch irgendein Ziel haben. Deswegen führen wir ein Punktesystem ein. Am linken Rand des Spielfelds befinden sich ein Punktestand, eine Levelanzeige und auch noch die (anfängs) drei Symbole für die drei Leben (siehe Abbildung 19.6).



Abbildung 19.6 Level, Punkte und Anzahl verbleibender Leben

Beginnen wir zunächst mit dem Zählen der Punkte. In Level 1 gibt es für jeden Meteoriten, dem ausgewichen wurde, zwei Punkte; in Level 2 sind es zwei Punkte mehr. Dies setzen wir so fort bis zum Level 5, dort gibt es zehn Punkte.

Wie Sie ja wissen, wird am Ende jedes Meteoritenfalls die Funktion `steinschlag()` aufgerufen. In dieser Funktion müssen wir die Punktezahl entsprechend anpassen und in dem dafür vorgesehenen dynamischen Textfeld ausgeben:

```
var level = 1;
var punkte = 0;
function steinschlag() {
    punkte += 2 * level;
    Punktefeld = "Punkte: " + punkte;
    // weiter wie vorher
}
```

Wenn ein Spieler ein Leben verliert, wird die Funktion `blink()` aufgerufen – das Raumschiff beginnt zu blinken. In dieser Funktion erledigen wir dann auch noch den dazugehörigen Rest:

- ▶ Der Lebenszähler wird um eins verringert (falls noch möglich).
- ▶ Ein Lebenssymbol wird ausgeblendet.
- ▶ Wenn keine Leben mehr übrig waren, wird zum Ende des Films gesprungen (mit der Statistik, die wir in Abschnitt 6 erstellen werden).

Hier sehen Sie den zugehörigen Code:

```
function blink() {
    if (leben > 0) {
        leben --;
        (eval("Leben" + leben))._visible = false;
    } else {
        gotoAndStop(75); // Anzeige des Endstands
        return;
    }
    // weiter wie vorher
}
```

Schließlich muss auch noch das Level erhöht werden – der Spieler will ja für seine todesmutigen Flugkünste belohnt werden. Dies erledigen wir einfach per Intervall: Alle zehn Sekunden kommt das neue Level. Nach 50

Sekunden, wenn eigentlich Level 6 erreicht worden wäre, wird der Spieler zum Endstand geschickt, er hat damit das Ende des Spiels erreicht.

```
setInterval(function() {  
    if (level<5) {  
        level++;  
        Levelfeld = "Level: " + level;  
    } else  
        gotoAndStop(75);  
}, 10000);
```

19.2.6 Endstand

Im letzten Bild schließlich wird der Endstand ausgegeben. Die erreichten Punkte werden dazu angezeigt. Sollte der Spieler noch Leben übrig haben, also 50 Sekunden lang sich wacker geschlagen haben, wird das zusätzlich belohnt: 25 Punkte pro Leben. In einem entsprechenden dynamischen Textfeld sieht der Spieler dann sein Ergebnis. Je nachdem, ob er mehr oder weniger als 75 Punkte erreicht hat, ist die Meldung mehr oder weniger freundlich:

```
var ergebnis = (punkte < 75) ? "Ziemlich " : "Nicht ";
```

```
Ergebnisfeld = ergebnis + "schlecht!\nErreichte Punkte: " +  
    punkte +  
    "\nÜbrige Leben: " + leben + " x 25 = " +  
    (leben * 25) +  
    "\n\nGesamtpunktzahl: " +  
    (punkte + 25*leben);
```

19.2.7 Musikalische Untermalung

Auf der CD-ROM zum Buch finden sie drei MP3-Dateien:

- ▶ **gut.mp3**, die bei einem guten Endergebnis abgespielt werden soll.
- ▶ **schlecht.mp3**, die bei einer eher niedrigen Punktezahl verwendet werden soll.
- ▶ **treffer.mp3**, die bei einer Kollision zwischen Meteorit und Raumschiff abgespielt werden soll.

Importieren Sie diese drei Dateien in Ihren Film und geben Sie im Dialogfenster **Verknüpfungseigenschaften** (siehe Abbildung 19.1) als Bezeichner

gut, **schlecht** bzw. **treffer** an. Dann müssen Sie nur noch an geeigneter Stelle die Soundeffekte abspielen.

- Den Sound **treffer** bei Kollision spielen Sie am besten zu Beginn der Funktion `blink()` ab:

```
var soundobj = new Sound();
soundobj.attachSound("treffer");
soundobj.start(0, 1);
```

- Die Sounds **gut** und **schlecht** werden im letzten Bild bei der Ausgabe des Ergebnisses verwendet, je nachdem, wie viele Punkte der Benutzer erreicht hat:

```
var sounddatei = (punkte < 75) ? "schlecht" : "gut";
var soundobj = new Sound();
soundobj.attachSound(sounddatei);
soundobj.start(0, 1);
```

19.2.8 Veröffentlichen und spielen

Nachfolgend finden Sie noch einmal den kompletten Code für Bild 1 des Films – diesen Code haben Sie bis dato immer nur häppchenweise und in Auszügen gesehen:

```
var getroffen = false;
var ID, ID2;
var i;
var level = 1;
var leben = 3;
var punkte = 0;

trace("1 :-(");

setInterval(function() {
    if (level<5) {
        level++;
        Levelfeld = "Level: " + level;
    } else
        gotoAndStop(75);
}, 10000);

function blink() {
    var soundobj = new Sound();
```

```

    soundobj.attachSound("treffer");
    soundobj.start(0, 1);
    if (leben > 0) {
        leben --;
        (eval("Leben" + leben))._visible = false;
    } else {
        gotoAndStop(75);
        return;
    }
    i = 0;
    ID = setInterval(function() {
        i++;
        if (i==6) clearInterval(ID);
        Gleiter._visible = !(Gleiter._visible);
    }, 200);
}

function steinschlag() {
    punkte += 2 * level;
    Punktefeld = "Punkte: " + punkte;
    var zufall = Math.random();
    var verzoeigerung = Math.floor((5 - level) * zufall) * 250;
    ID2 = setInterval(StarteMeteor, verzoeigerung);
}

function StarteMeteor() {
    var zufall = Math.random();
    var bildnr = 3 + 10 * Math.floor(5 * zufall);
    gotoAndPlay(bildnr);
    clearInterval(ID2);
}

```

Listing 19.1 Das komplette Bildskript für Bild 1

Ebenfalls etwas länger ist das Bildskript für das letzte Bild:

```

var ergebnis = (punkte < 75) ? "Ziemlich " : "Nicht ";
var sounddatei = (punkte < 75) ? "schlecht" : "gut";

Ergebnisfeld = ergebnis + "schlecht!\nErreichte Punkte: " + punkte +
    "\nÜbrige Leben: " + leben + " x 25 = " + (leben * 25) +
    "\n\nGesamtpunktzahl: " + (punkte + 25*leben);

```

```

var soundobj = new Sound();
soundobj.attachSound(sounddatei);
soundobj.start(0, 1);

stop();

```

Listing 19.2 Das komplette Bildskript für Bild 75

Sie können den Film nur selbst im Webbrowser oder im Flash Player ausprobieren. In Abbildung 19.7 sehen Sie einen Ausschnitt während dem Spielablauf, in Abbildung 19.8 einen möglichen (nicht sonderlich guten) Endstand.

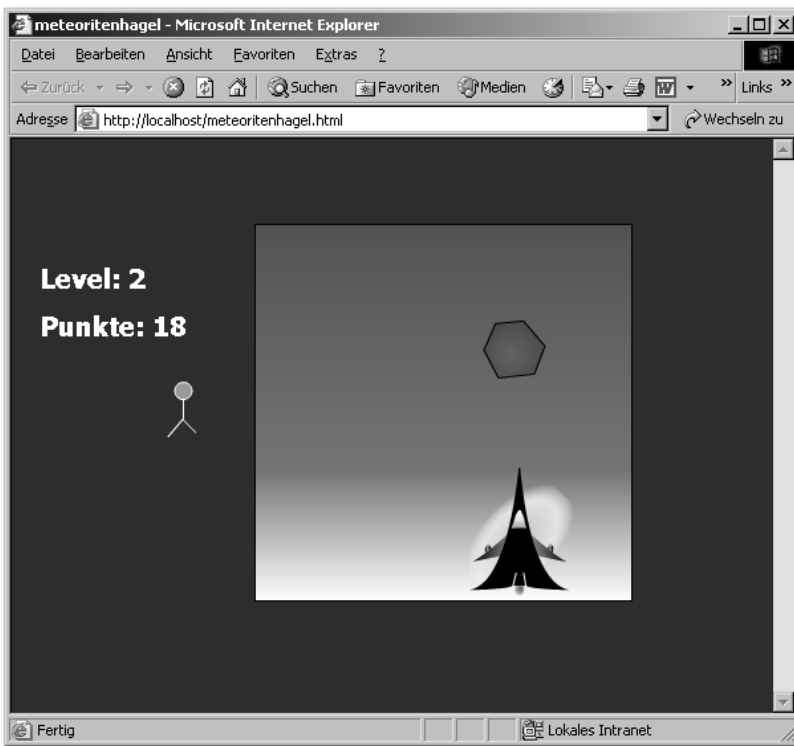


Abbildung 19.7 Das wird knapp ...

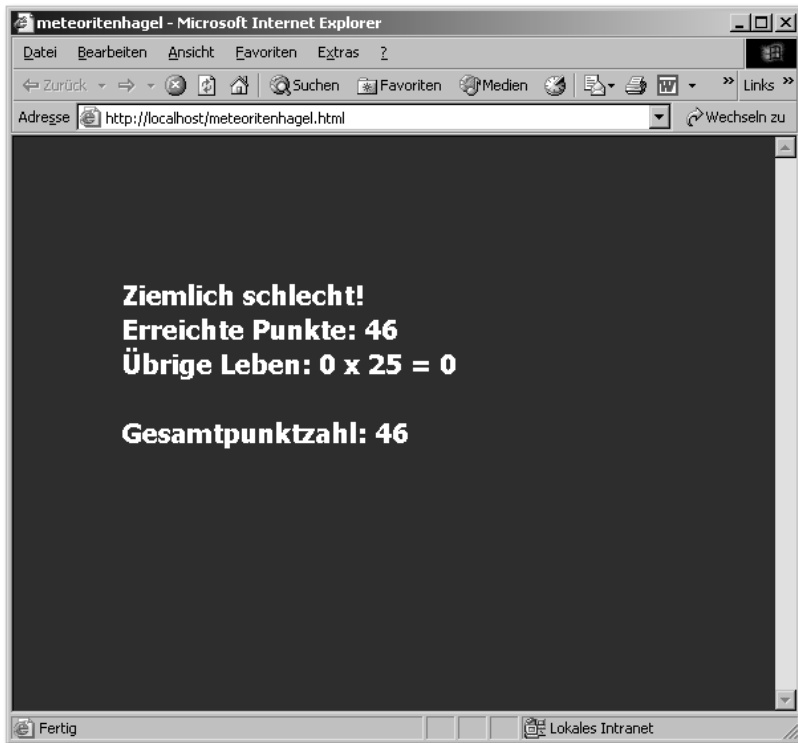


Abbildung 19.8 Zurück in die Flugschule!



Wenn Ihnen das Spiel zu schwer vorkommt, haben Sie mehrere Möglichkeiten, den Schwierigkeitsgrad zu verringern:

- ▶ Setzen Sie die Bildrate herab.
- ▶ Das Raumschiff wird pro Tastendruck weiter nach links oder rechts bewegt als bisher (5 Pixel).
- ▶ Verlängern Sie die Meteoriten-Tweens, so dass ein Meteorit länger unterwegs ist.

19.3 Erweiterungen

Wir haben Ihnen nun an diesem recht übersichtlichen, aber dennoch anspruchsvollen Beispiel gezeigt, wie Sie in Flash ein selbstständiges Spiel erstellen können. Das ist jedoch nur die Spitze des Eisbergs, allein schon für dieses Beispiel gibt es mannigfaltige Erweiterungsmöglichkeiten:

- ▶ Einführung verschiedener Meteoriten,
- ▶ spezielle Bonus-Elemente, die bei Kollision Punkte bringen oder das Schiff eine Zeitlang unverwundbar machen,
- ▶ Erstellung einer Hiscoreliste, die dann serverseitig abgespeichert wird,
- ▶ Schußmöglichkeit: Meteoriten können »beseitigt« werden, fremde Raumschiffe schießen zurück,
- ▶ aufwändigere Grafik :-).

Sie sehen also, mit Flash ist vieles möglich, die begrenzende Variable ist oftmals Ihre eigene Vorstellungskraft.

20 Komponenten von Flash MX

Der Horizont des Berliners ist längst nicht so groß wie seine Stadt.

– Kurt Tucholsky

In der neuen Version von Flash, Flash MX, liefert Macromedia einen Satz Komponenten mit, die so genannten **Flash UI Components** (Sie sehen: in der deutschen Version nicht übersetzt).

Mit diesen Komponenten will Macromedia sein Vorzeigeprodukt Flash endlich vom »Skip-Intro«-Image wegbringen. Stattdessen soll Flash hauptsächlich zum Erstellen grafischer Benutzeroberflächen eingesetzt werden. Der versammelten Fachjournaille wurde dazu in einem Münchner Nobelhotel eine vollkommen in Flash geschriebene Online-Hotelreservierung gezeigt. Alles spielte sich in einem Flash-Film ab.

Was aber war daran neu? Nun, da Flash jahrelang als Animationstool konzipiert und auch vermarktet worden ist, wurden Techniken zur Erstellung von grafischen Oberflächen zunächst nicht mitgeliefert. Die Rede ist von Formularelementen wie etwa Checkboxes oder Radiobuttons. Dafür mussten zuvor erst externe Komponenten entwickelt oder eingekauft werden (oder man fand geeignete Gratis-Komponenten). Im Lieferumfang von Flash waren sie nicht enthalten (einzige Ausnahme: Eingabefeld).

Auch Schaltflächen waren mühsamer zu erstellen, als man das eigentlich erwartet hätte. Sie wissen ja – Sie müssen ein eigenes Symbol erzeugen, um ein Textfeld zu einer Schaltfläche zu machen. Das muss doch einfacher gehen!

In Flash MX ist das jetzt anders. Hier gibt es eine ganze Reihe von UI-Komponenten, und an der einen oder anderen Stelle in diesem Buch haben wir diese bereits eingesetzt. In diesem Kapitel wollen wir alle UI-Komponenten kurz vorstellen und eine kleine Beispielapplikation entwickeln.

20.1 Übersicht

Um auf die Komponenten zuzugreifen, müssen Sie die entsprechende Symbolleiste einblenden (falls noch nicht geschehen). Dazu verwenden

Sie den Menübefehl **Fenster • Komponenten** oder das Tastenkürzel **Strg + F7** (siehe Abbildung 20.1).

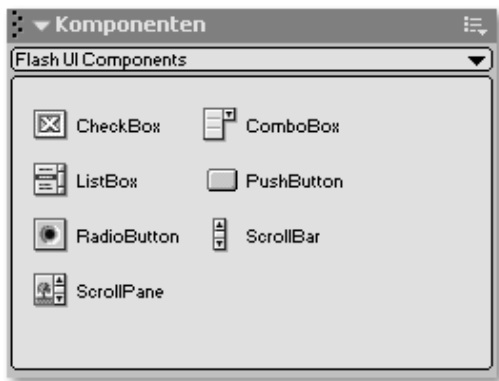


Abbildung 20.1 Die Flash UI Components

Hier finden Sie eine kurze Übersicht der zur Verfügung stehenden sieben Komponenten.

Icon	Name	Beschreibung
	CheckBox	CheckBox (Kontrollkästchen) mit zwei Zuständen: angekreuzt und nicht angekreuzt
	ComboBox	Kann sowohl als Pull-down-Menü dienen als auch als Suchfeld: In einem Textfeld über dem Pull-down-Menü kann Text eingegeben werden; in der Liste werden dann die entsprechenden Felder angezeigt
	ListBox	Auswahlliste
	PushButton	Eine Schaltfläche, die beim Drücken ActionScript-Code ausführen kann
	Radiobutton	Radiobutton (Optionsfeld); von allen Radiobuttons aus einer Gruppe kann immer nur einer ausgewählt werden
	ScrollBar	Scrollleiste, die an Textfelder angehängt werden kann, so dass bei (zu) viel Text gescrollt werden kann
	ScrollPane	Scrollbarer Bereich, in den Movieclips eingebunden werden können

Tabelle 20.1 Die Flash UI Components

Um eine solche Komponente zu verwenden, ziehen Sie sie einfach auf die Bühne. Den Rest erledigen Sie dann über den Eigenschafteninspektor.

20.2 Ansteuerung

Wir wollen Ihnen nun die sieben Komponenten jeweils kurz vorstellen und dabei auf die wichtigsten Eigenschaften eingehen, bevor wir ein komplettes Beispiel erstellen.

20.2.1 CheckBox

Wenn Sie ein `CheckBox`-Element auf die Bühne ziehen, sehen Sie dort nicht nur das Kontrollkästchen, sondern auch den Text **Check Box** neben dem Kästchen. Ein Blick in den Eigenschafteninspektor zeigt Ihnen, wieso: Die Beschriftung ist mit der Checkbox fest verbunden.

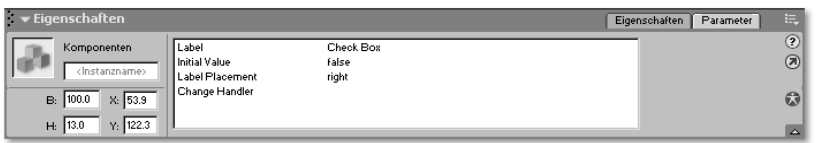


Abbildung 20.2 Der Eigenschafteninspektor für die Checkbox

Sie können im Eigenschafteninspektor unter der Rubrik **Parameter** die folgenden Einstellungen vornehmen.

Parameter	Beschreibung
Label	Die Beschriftung der Checkbox
Initial Value	Gibt an, ob die Checkbox anfangs angekreuzt ist (<code>true</code>) oder nicht (<code>false</code>)
Label Placement	Ausrichtung der Beschriftung an der Checkbox (<code>left</code> = links, <code>right</code> = rechts)
Change Handler	Funktion, die aufgerufen werden soll, wenn die Checkbox angeklickt wird und ihren Zustand ändert

Tabelle 20.2 Parameter für die `CheckBox`-Komponente

Im Hinblick auf `ActionScript` lässt sich diese Komponente auf mehrere Arten und Weisen anpassen; Details finden Sie in der Flash-Hilfe unter dem Stichwort **FCheckBox**. Am wichtigsten ist jedoch die Methode `getValue()`, die zurückgibt, ob die Checkbox angekreuzt ist (`true`) oder nicht (`false`).

20.2.2 ComboBox

Die ComboBox-Komponente ist ein sehr funktionsfähiges Drop-down-Menü. In Abbildung 20.3 sehen Sie den Eigenschafteninspektor für diese Komponente.

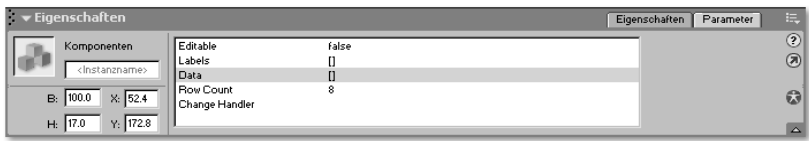


Abbildung 20.3 Der Eigenschafteninspektor für die Combobox

Hier sehen Sie die Bedeutungen der einzelnen Felder.

Parameter	Beschreibung
Editable	Beim Wert false wird ein »herkömmliches« Pull-down-Menü angezeigt, bei true dagegen kann der Benutzer per Tastatur die ersten Zeichen des gewünschten Eintrags eingeben und ihm werden sofort alle passenden Einträge angezeigt.
Labels	Array, das die Beschriftungen der einzelnen Listeneinträge enthält
Data	Array, das die Werte der einzelnen Listeneinträge enthält. Diese Werte sind für den Endbenutzer nicht sichtbar, können aber bei der Formelarauswertung per ActionScript abgefragt werden.
Row Count	Anzahl der gleichzeitig darstellbaren Listeneinträge
Change Handler	Funktion, die aufgerufen wird, sobald ein Listenelement ausgewählt worden ist

Tabelle 20.3 Parameter für die ComboBox-Komponente

Im Hinblick auf ActionScript ist natürlich besonders von Interesse, wie Sie auf das ausgewählte Listenelement zugreifen können. Hier gibt es zwei verschiedene Ansätze.

- Die Methode `getSelectedIndex()` liefert den Index des ausgewählten Elements zurück. Die Methode `getItemAt()` wiederum liefert das zu einer Position gehörende Element zurück. Als Rückgabewert wird ein Objekt mit den Eigenschaften `data` und `label` geliefert:

```
var element = Liste.getItemAt(Liste.getSelectedIndex());
trace(element.label + ": " + element.data);
```

- Etwas kürzer geht es mit der Methode `getSelectedItem()`, die eine Kurzform von `x.getItemAt(x.getSelectedIndex())` darstellt:

```
var element = Liste.getSelectedItem();  
trace(element.label + ": " + element.data);
```

Die Eigenschaft `data` entspricht dem Eintrag im Array `Data`, `label` dem Eintrag im Array `Labels` des gewählten Elements.



Weitere Informationen über die ActionScript-Methoden der ComboBox-Komponente finden Sie in der ActionScript-Referenz unter dem Stichwort **FComboBox**.



20.2.3 ListBox

Die ListBox-Komponente repräsentiert eine Auswahlliste. Im Gegensatz zu einer Drop-down-Liste sind bei einer Auswahlliste mehrere Einträge gleichzeitig sichtbar und müssen nicht per Mausklick erst »aufgeklappt« werden. Deswegen ist es bei einer Auswahlliste auch möglich, mehrere Elemente gleichzeitig auszuwählen.

Dazu muss die `[Strg]`-Taste beim Anklicken gedrückt gehalten werden; zur Markierung von mehreren hintereinander liegenden Einträgen kann  verwendet werden.

Auch wenn Ihnen das bereits bekannt ist, Ihre Anwender wissen es vielleicht noch nicht. Weisen Sie sie also darauf hin!

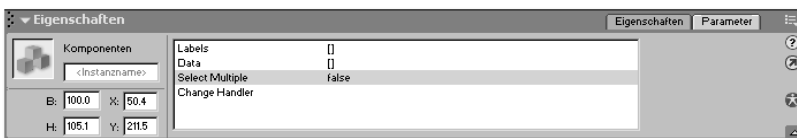


Abbildung 20.4 Der Eigenschafteninspektor für die Auswahlliste

In Abbildung 20.4 sehen Sie den Eigenschafteninspektor für die Komponente; Sie können dort (beziehungsweise im Register **Parameter**) die folgenden Werte einstellen.

Parameter	Beschreibung
Labels	Array, das die Beschriftungen der einzelnen Listenelemente enthält
Data	Array, das die Werte der einzelnen Listenelemente enthält – für den Benutzer unsichtbar, aber für die Formelarauswertung von Bedeutung
Select Multiple	Gibt an, ob mehrere Listenelemente gleichzeitig ausgewählt werden können (<code>true</code>) oder nicht (<code>false</code>)
Change Handler	Funktion, die bei einer Änderung an den gewählten Listenelementen (anderes Element auswählen, neues Element auswählen) aufgerufen wird

Tabelle 20.4 Die Parameter der ListBox-Komponente

Da ein Formular wohl nur dann sinnvoll ist, wenn die Formulardaten auch wieder verwendet werden können, werfen wir auch dieses Mal einen besonderen Blick auf die ActionScript-Funktionalität zur Abfrage der Formulareingaben – in diesem Fall: die Abfrage der gewählten Listenelemente.

Zwar gibt es auch für die Auswahlliste – Eintrag **FListBox** in der ActionScript-Hilfe – die Methoden `getSelectedIndex()` und `getSelectedItem()`, diese ermitteln aber nur jeweils das erste gewählte Element (bzw. dessen Position). Für Mehrfach-Listen sind jedoch alle Angaben von Interesse, weswegen zwei weitere Methoden angeboten werden:

- `getSelectedIndices()` gibt in einem Array die Positionen aller gewählten Elemente zurück; ist kein Element gewählt, liefert die Methode `undefined` zurück.
- `getSelectedItems()` liefert – ebenfalls in einem Array – alle gewählten Elemente als Objekt mit den Eigenschaften `data` und `label` zurück.

Folgender Code gibt mittels `trace()` alle gewählten Elemente einer Liste aus:

```
var el = Liste.getSelectedItems();
for (var i=0; i<el.length; i++)
    trace(el[i].label + ": " + el[i].data);
```


20.2.4 PushButton

Eine Schaltfläche wird mit der PushButton-Komponente realisiert; in Kapitel 17, ActionScript mit JavaScript, haben wir diese Komponente bereits einmal verwendet.

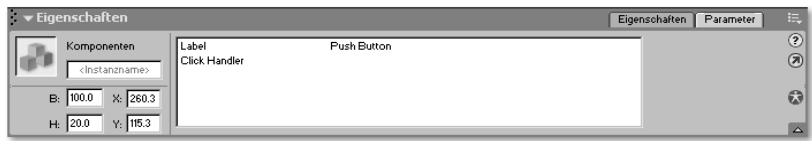


Abbildung 20.5 Der Eigenschafteninspektor für die Schaltfläche

Der Eigenschafteninspektor für diese Komponente (siehe Abbildung 20.5) ist recht spärlich mit Eigenschaften bestückt.

Parameter	Beschreibung
Label	Beschriftung der Schaltfläche
Click Handler	Funktion, die beim Anklicken der Schaltfläche aufgerufen wird

Tabelle 20.5 Die Parameter für die PushButton-Komponente

Der Name der Behandlungsfunktion wird dabei ohne Klammern angegeben. Wenn Sie also den Parameter **Click Handler** auf **Klick** setzen, wird die Funktion `Klick()` aufgerufen, die beispielsweise wie folgt aussehen kann:

```
function Klick() {  
    trace("Schaltfläche angeklickt!");  
}
```

20.2.5 RadioButton

Mit der RadioButton-Komponente wird ein Radiobutton dargestellt. Das allein ist noch nichts Besonderes; sinnvoll wird diese Komponente erst, wenn Sie mehrere Instanzen davon verwenden. Von allen Radiobuttons einer Gruppe kann nämlich immer nur einer ausgewählt werden. Sie können damit dem Benutzer eine Reihe von Alternativen anbieten, von denen nur eine gewählt werden kann (beispielsweise beim Geschlecht).



Abbildung 20.6 Die Eigenschaften des Radiobuttons

Die Eigenschaften der Komponente (Eigenschafteninspektor, Register **Parameter**) können Sie der folgenden Tabelle entnehmen.

Parameter	Beschreibung
Label	Beschriftung des Radiobuttons
Initial State	Ausgangszustand: aktiviert (<code>true</code>) oder nicht (<code>false</code>)
Group Name	Name der Radiobutton-Gruppe; wie gesagt: Von allen Radiobuttons aus einer Gruppe kann immer nur einer aktiviert sein
Data	Wert des Radiobuttons (Bezeichnung)
Label Placement	Ausrichtung der Beschriftung: <code>left</code> = links, <code>right</code> = rechts
Change Handler	Funktion, die aufgerufen wird, wenn der Radiobutton angeklickt (und damit selektiert oder deselektiert) wird

Tabelle 20.6 Die Parameter der RadioButton-Komponente

Um auf den ausgewählten Radiobutton einer Gruppe zuzugreifen, können Sie die Methode `getValue()` der Optionsfeldgruppe verwenden. Diese liefert den `Data`-Parameter des ausgewählten Radiobuttons zurück. Sollte `Data` nicht gesetzt sein, wird die Bezeichnung des Felds ermittelt.



Sollte gar kein Radiobutton der Gruppe ausgewählt worden sein, liefert `getValue()` als Rückgabewert `undefined`.

```
trace(Radiogruppe.getValue());
```



Alternativ dazu können Sie auch jeden Radiobutton einzeln betrachten und mit der Methode `getState()` überprüfen, ob der betreffende Radiobutton ausgewählt wurde (`true`) oder nicht (`false`).

Zugriff auf die Radiobutton-Gruppe erhalten Sie über den angegebenen Gruppennamen. Wie Sie im Debugger sehen können, enthält dann diese Gruppe unter anderem eine Eigenschaft `foobar` mit dem Wert `0` – der Sinn der Variablen bleibt im Dunkeln (`foobar` wird in englischsprachiger Literatur oft als exemplarischer Variablenname verwendet).

20.2.6 ScrollBar

Wie Sie bereits in Kapitel 18, XML, gesehen haben, können Sie die ScrollBar-Komponente von Flash MX dazu verwenden, dynamische Textfelder (und auch Texteingabefelder) mit Scrollbalken zu versehen – bei früheren Flash-Versionen war das sehr aufwändig und nur mit externen Komponenten oder eigener Programmierung möglich.

Sie müssen jedoch ein paar Kleinigkeiten beachten, wenn Sie dieses Element verwenden möchten. Befolgen Sie die folgenden Schritte, um ein Textfeld mit einem Scrollbalken zu bereichern.

1. Erstellen Sie zunächst ein dynamisches Textfeld oder ein Eingabetextfeld.
2. Geben Sie einen Instanznamen für das Textfeld an (ohne geht es nicht!).
3. Ziehen Sie die ScrollBar-Komponente auf die Bühne, über das Textfeld. Sie passt sich dann automatisch der Größe des Textfeldes an.

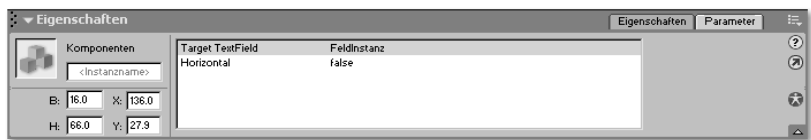


Abbildung 20.7 Die Eigenschaften für den Scrollbalken

Im Eigenschafteninspektor (Abbildung 20.7) können Sie die folgenden Parameter für den Scrollbalken einstellen.

Parameter	Beschreibung
Target TextField	Instanzname des Textfeldes, an den der Scrollbalken angehängt werden soll
Horizontal	Gibt an, ob es sich um einen waagerechten (<code>true</code>) oder senkrechten (<code>false</code> , Standard) Scrollbalken handelt

Tabelle 20.7 Die Parameter der ScrollBar-Komponente

Wenn Sie für ein Textfeld sowohl horizontale als auch vertikale Scrollbalken benötigen, ziehen Sie zweimal die ScrollBar-Komponente auf die Bühne und setzen Sie den Parameter **Horizontal** entsprechend.





Sie können – da Sie Instanznamen verwenden – nicht mehr über den Feldnamen auf den Wert im Textfeld zugreifen; verwenden Sie stattdessen `Instanzname.text`.

20.2.7 ScrollPane

Mit der ScrollPane-Komponente können Sie Movieclips in ein Flash-Dokument einbinden und müssen sich dabei nicht mehr um Scrollbalken et cetera kümmern; die Komponente nimmt Ihnen die Hauptarbeit ab.

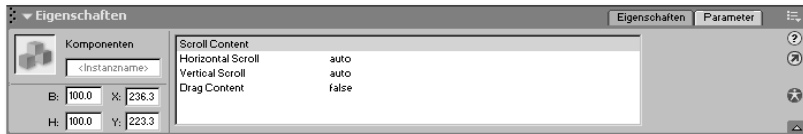


Abbildung 20.8 Die Eigenschaften des Scrollfeldes

In Abbildung 20.8 sehen Sie den Eigenschafteninspektor für die ScrollPane-Komponente; dort stehen die folgenden Parameter zur Verfügung.

Parameter	Beschreibung
Scroll Content	ID des Movieclips, der in dem Feld angezeigt werden soll
Horizontal Scroll	Gibt an, ob waagrechte Scrollbalken angezeigt werden sollen: <code>true</code> = ja, <code>false</code> = nein, <code>auto</code> = bei Bedarf
Vertical Scroll	Gibt an, ob senkrechte Scrollbalken angezeigt werden sollen: <code>true</code> = ja, <code>false</code> = nein, <code>auto</code> = bei Bedarf
Drag Content	Gibt an, ob der Benutzer durch Mausoperationen den Inhalt der Komponente scrollen kann (<code>true</code>) oder lediglich durch Scrollbalken, falls vorhanden (<code>false</code>)

Tabelle 20.8 Die Parameter der ScrollPane-Komponente



Sie müssen den Movieclip nicht unbedingt schon bei der Filmerstellung im Eigenschafteninspektor angeben. Flash bietet die Möglichkeit, den Clip auch dynamisch zur Laufzeit zu laden. Verwenden Sie dazu die Methode `loadScrollContent()` und übergeben Sie als Parameter entweder eine `.swf`-Datei oder eine JPEG-Grafik.

20.3 Beispiel

Nach diesem ersten Überblick wollen wir dieses Kapitel mit einem Beispiel beenden, in dem wir die meisten der zuvor gezeigten Komponenten einsetzen. Ziel ist es, ein Formular zu erstellen und die Formulardaten auszuwerten. Das Formular kann allerdings nur dann abgeschickt werden, wenn alle Felder ausgefüllt worden sind.

20.3.1 Formular erstellen

Zunächst erstellen wir eine Reihe von Formularelementen.

Name	Typ	Beschreibung
FName	Eingabefeld, einzeilig	Name des Benutzers
FMail	Eingabefeld, einzeilig	E-Mail-Adresse des Benutzers
FGeschlecht	Gruppe von RadioButton-Elementen	Geschlecht des Benutzers
FBuecher	Auswahlliste	Bücher von Galileo Press, die der Benutzer besitzt
FKommentar	Eingabefeld, mehrzeilig ohne Umbruch	Weitere Kommentare, die der Benutzer loswerden will
FButton	PushButton-Element	Schaltfläche zum Versenden des Formulars

Tabelle 20.9 Diese Formularelemente werden (unter anderem) verwendet.

Das mehrzeilige Eingabefeld versehen wir zusätzlich noch mit einem horizontalen und einem vertikalen Scrollbalken.

Außerdem setzen wir den **Change Handler** bei denjenigen Formularelementen, die das unterstützen. In unserem Beispiel sind das die beiden Radiobuttons und die Auswahlliste. Als Wert geben wir `aenderung` (ohne Klammern!) an; in der Funktion `aenderung()` (mit Klammern!) werden wir später auf die Änderungen reagieren.

In Abbildung 20.9 sehen Sie das Feedback-Formular im Browser; wir haben es bereits mit einigen Werten belegt.

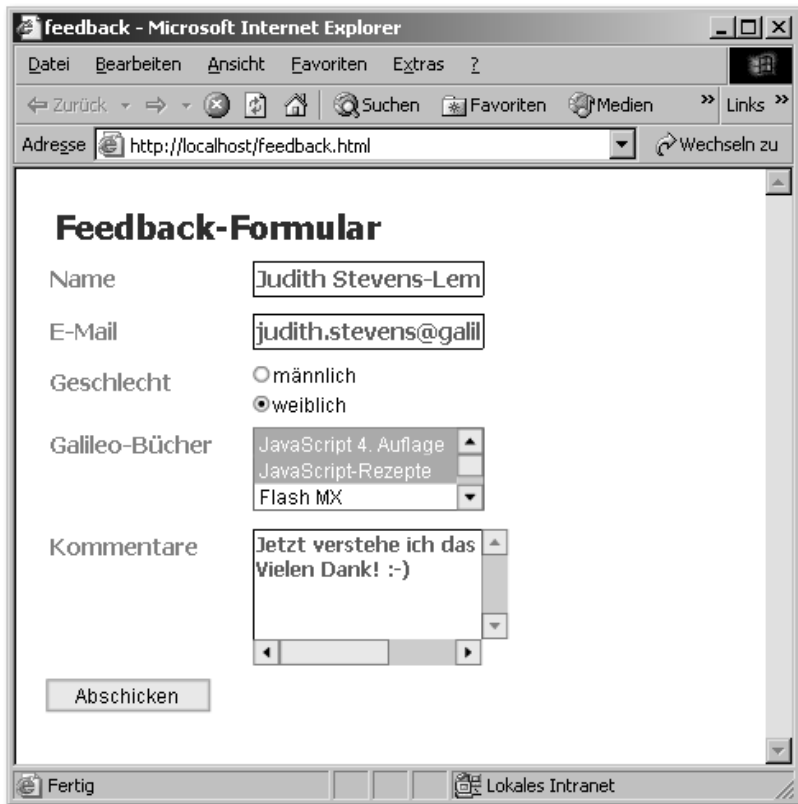


Abbildung 20.9 Das Feedback-Formular, schon ausgefüllt

20.3.2 Formular überprüfen

Beginnen wir nun mit der Programmierung des Bildskripts für den Film. Zunächst einmal wird der Film angehalten.

```
stop();
```

Außerdem wollen wir die Schaltfläche zum Formularversand erst dann freigeben, wenn alle Felder ausgefüllt worden sind. Die Eigenschaft `setEnabled()` der Schaltfläche ermöglicht dies. Der Parameter `false` deaktiviert die Schaltfläche, `true` aktiviert sie (das heißt sie kann angeklickt werden).

```
FButton.setEnabled(false);
```

In der Funktion `aenderung()`, die auch nach jeder Änderung an den Radiobuttons und der Auswahlliste aufgerufen wird, wird der Status der Schaltfläche aktualisiert.

```
function aenderung() {  
    FButton.setEnabled(formularcheck());  
}
```

Als Parameter an `setEnabled()` wird also der Rückgabewert einer Funktion `formularcheck()` übergeben. Liefert `formularcheck()` den Wert `false`, bleibt (oder wird) die Schaltfläche deaktiviert, bei `true` bleibt (oder wird) sie aktiviert.

Wenden wir uns also dieser Funktion direkt zu. Es werden alle Felder überprüft. Wenn eines davon nicht ausgefüllt worden ist, wird `false` zurückgeliefert, sonst `true`.

```
function formularcheck() {  
    if (FName.text == "")  
        return false;  
    if (FMail.text == "")  
        return false;  
    if (FGeschlecht.getValue() == undefined)  
        return false;  
    if (FBuecher.getSelectedItems() == undefined)  
        return false;  
    if (FKommentar.text == "")  
        return false;  
    return true;  
}
```

Ein Problem bei diesem Ansatz ist jedoch, dass Sie zwar bei Radiobuttons und Auswahllisten (und auch einigen der anderen Komponenten) eine Veränderung per Skript feststellen können, Textfelder diese Möglichkeit jedoch nicht bieten. Aus diesem Grund muss die Funktion `aenderung()` periodisch aufgerufen werden, damit auch Änderungen in den Textfeldern berücksichtigt werden.

```
var ID = setInterval(aenderung, 500);
```

20.3.3 Formular verarbeiten

Der letzte Schritt besteht darin, die Formulardaten auf irgendeine Art und Weise weiterzuverarbeiten. Wir begnügen uns zunächst damit, sie einfach auszugeben.

Das Problem besteht nun darin, auf die Werte zuzugreifen. Die Ausgabe erfolgt in einem anderen Bild, und von dort aus können Sie nicht auf Formularelemente im ersten Bild zugreifen. Deswegen empfiehlt sich folgende Vorgehensweise:

- ▶ Lesen Sie im ersten Bild beim Drücken der Schaltfläche die Formulardaten aus.
- ▶ Speichern Sie einen Ausgabetext (Dankesmeldung, Formulardaten) in einer Variablen.
- ▶ Springen Sie jetzt zum nächsten Bild.
- ▶ Geben Sie dort die Formulardaten aus der Variablen in einem dynamischen Textfeld aus.

Wir erstellen zunächst ein neues Schlüsselbild in Bild 2 und dort ein mehrzeiliges, dynamisches Textfeld, an das wir sicherheitshalber noch einen vertikalen Scrollbalken anhängen. Als Instanznamen verwenden wir `FAusgabe`.

Nun zum Code, der beim Anklicken der Schaltfläche ausgeführt werden soll. Wir geben dazu im Eigenschafteninspektor des `PushButton`-Elements unter **Click Handler** den Wert `ausgabe` an. Die Funktion `ausgabe()` wird im Folgenden erstellt.

Zunächst wird die periodische Ausführung der Formularüberprüfung (in Bezug auf den Zustand der Schaltfläche) beendet, das ist nun nicht mehr nötig.

```
clearInterval(ID);
```

Wir beginnen dann mit dem Einlesen der Formulardaten.

```
var ausgabertext = "Danke für Ihre Angaben!\n";
ausgabertext += "\nName: " + FName.text;
ausgabertext += "\nE-Mail: " + FMail.text;
ausgabertext += "\nGeschlecht: " + FGeschlecht.getValue();
ausgabertext += "\nBücher: ";
var el = FBuecher.getSelectedItems();
for (var i=0; i<el.length; i++)
```



```

    ausgabertext += " " + el[i].label;
    ausgabertext += "\nKommentare: " + FKommentar.text;

```

Die Daten sind jetzt in der Variablen `ausgabertext` abgespeichert; nun können wir das Bild wechseln und den Text ausgeben.

```

gotoAndStop(2);
FAusgabe.text = ausgabertext;

```

Nachfolgend finden Sie noch einmal den vollständigen Code für Bild 1 des Films; auf der CD-ROM finden Sie den kompletten Film unter dem Dateinamen **feedback fla**.

```

stop();
FButton.setEnabled(false);

var ID = setInterval(aenderung, 500);

function aenderung() {
    FButton.setEnabled(formularcheck());
}

function formularcheck() {
    if (FName.text == "")
        return false;
    if (FMail.text == "")
        return false;
    if (FGeschlecht.getValue() == undefined)
        return false;
    if (FBuecher.getSelectedItems() == undefined)
        return false;
    if (FKommentar.text == "")
        return false;
    return true;
}

function ausgabe() {
    clearInterval(ID);
    var ausgabertext = "Danke für Ihre Angaben!\n";
    ausgabertext += "\nName: " + FName.text;
    ausgabertext += "\nE-Mail: " + FMail.text;
    ausgabertext += "\nGeschlecht: " + FGeschlecht.getValue();

```

```

    ausgabertext += "\nBücher: ";
    var el = FBuecher.getSelectedItems();
    for (var i=0; i<el.length; i++)
        ausgabertext += " " + el[i].label;
    ausgabertext += "\nKommentare: " + FKommentar.text;

    gotoAndStop(2);
    FAusgabe.text = ausgabertext;
}

```

Listing 20.1 Das Bildskript für Bild 1 des Films



Abbildung 20.10 Die Formulardaten werden ausgegeben.

Die Ausgabe der Formulardaten (in Abbildung 20.11 gezeigt) ist natürlich nur eine Art Notlösung. In der Praxis werden Sie die Daten wohl per E-Mail verschicken. In Kapitel 21, Serverseitige Technologien, finden Sie hierzu weitere Informationen.

20.4 Fazit

Wie Sie an dieser ziemlich gerafften Vorstellung schon gesehen haben, bieten die UI Components von Flash MX oft gewünschte Funktionalitäten und sind zudem nicht schwer zu erlernen. Sie sollten jedoch immer bedenken, dass Sie einen solchen Film nur für das Flash-6-Plug-In veröffentlichen können, frühere Versionen unterstützen die Komponenten nicht. Auch wenn das Flash-Plug-In nicht groß ist, viele Anwender fürchten zusätzliche Installationen und können dann Ihren Film überhaupt nicht betrachten.

Es wird sicherlich noch einige Zeit dauern, bis das Gros der Benutzer den Upgrade vollzogen hat; bis dahin sollten Sie sich genau überlegen, ob Sie nicht doch lieber für ältere Versionen wie Flash 5 oder gar Flash 4 entwickeln, trotz der vielen Vorteile und Erleichterungen von Flash MX.

Wie dem auch sei, Sie sollten für Anwender älterer Plug-Ins (und für alle anderen Anwender ohne Flash) sowieso eine HTML-Version Ihrer Seiten anbieten, da Sie sonst potenzielle Kunden von Ihrer Seite fernhalten. Mit Flash können Sie wunderbare Dinge anstellen, aber kein Webdesigner ist in der Position, einem Anwender diktieren zu können, was er zu installieren hat und was nicht.

21 Serverseitige Technologien

Das sind die wahren Wunder der Technik, dass sie das, wofür sie entschädigt auch wirklich kaputt macht.

– Karl Kraus

Das World Wide Web wird immer mehr zu einem dynamischen Web. Statische HTML-Inhalte gibt es kaum noch, und im professionellen Bereich sind nicht datenbankgestützte Seiten auf dem Rückzug. Die gute Nachricht: Auch mit Flash können Sie die umfangreichen Möglichkeiten serverseitiger Programmierung nutzen und Daten mit Ihrer serverseitigen Anwendung austauschen.

21.1 Voraussetzungen

Die schlechte Nachricht: Die Voraussetzungen für den Einsatz von serverseitigen Technologien sind höher. Sie benötigen am besten einen lokalen Webserver oder zumindest Zugriff auf einen Webserver, um die Skripte zu testen. Die serverseitigen Technologien müssen auch auf dem Server installiert und eingerichtet werden. Dies sind alles mehr oder weniger einfache Aufgaben, nur – in einem Buch über ActionScript und JavaScript kann hierfür nicht viel Platz eingeräumt werden.

Wir gehen also bei den Beispielen in diesem Kapitel von einer bereits bestehenden Installation aus.

Ein weiteres Problem: Es gibt eine ganze Reihe unterschiedlicher Skriptsprachen, wir mussten uns aber (für dieses Buch) für eine entscheiden. Unsere Wahl ist auf PHP gefallen. PHP ist nicht viel besser oder schlechter als andere serverseitige Skriptsprachen und Technologien, hat aber den Vorteil, recht weit verbreitet zu sein und zudem für fast jedes Betriebssystem zur Verfügung zu stehen. Zudem bietet Lycos momentan unter <http://www.tripod.lycos.de/> kostenlosen, werbefinanzierten Webspace inklusive PHP-Unterstützung und MySQL-Datenbank an. Aus diesem Grund werden wir in diesem Kapitel PHP einsetzen und dazu eine MySQL-Datenbank verwenden.

Wenn Sie das selbst ausprobieren möchten, hier die zugehörigen Links:

- Unter <http://www.php.net/> finden Sie die für PHP nötigen Dateien, sowohl als Quellcode als auch als ausführbare Dateien für Windows. Außerdem finden Sie dort eine ausführliche Dokumentation.



- Die Datenbank MySQL erhalten Sie unter <http://www.mysql.com/> für verschiedene Betriebssysteme.
- Unter <http://www.phpmyadmin.net/> gibt es ein bekanntes, webbasiertes PHP-Skript zur einfachen Administration von MySQL – die mitgelieferte Oberfläche von MySQL ist nicht wirklich reich an Funktionen.

Im Folgenden gehen wir davon aus, dass PHP und MySQL installiert und konfiguriert sind. Sie müssen in unseren Skripten dann nur noch die systemspezifischen Angaben (etwa Servername, Benutzername etc.) durch Ihre Angaben ersetzen. Außerdem sollten Sie bereits PHP programmieren können (oder zumindest so neugierig sein, sich dies beizubringen).

Nun ist es allerdings so, dass Flash zurzeit nur unter der Macintosh- und der Windows-Plattform zur Verfügung steht. Während Mac OS eher selten als Webserverplattform zum Einsatz kommt, bietet Windows mit dem Internet Information Server (IIS) oder der kleineren Variante, dem Personal Web Server (PWS) eine ausgereifte Software. Ein regelmäßiger Besuch bei <http://windowsupdate.microsoft.com/> ist zwar sehr zu empfehlen, der Webserver selbst läuft aber äußerst zuverlässig, stabil und performant.

Bei den Microsoft-Webservern ist ASP (das Akronym steht nicht für Application Service Providing/Provider, sondern für Active Server Pages) gleich mit dabei. ASP ist eine Technologie, die die Ausführung von serverseitigen Skripten unterstützt.

Je nachdem, welche Windows-Version Sie einsetzen, ist der Webserver gleich mit dabei oder kann nachinstalliert werden. Details, wo Sie im Falle des Falles die Software herbekommen, entnehmen Sie bitte der folgenden Tabelle:

Betriebssystem	Webserver	Bezugsquelle
Windows 95	PWS	Download unter http://www.microsoft.com/ntserver/nts/downloads/recommended/NT4OptPk/win95.asp
Windows 98/ 98SE	PWS	Befindet sich auf der Betriebssystem-CD-ROM (suchen Sie nach »pws«)
Windows ME	PWS	Eigentlich nicht verfügbar, laut Microsoft ist aber der PWS für Windows 95 (siehe oben) mit Einschränkungen auch unter Windows ME verwendbar

Tabelle 21.1 Die verschiedenen Bezugsquellen für die Webserver

Betriebssystem	Webserver	Bezugsquelle
Windows NT 4.0 Workstation	PWS	Download unter http://www.microsoft.com/ntserver/nts/downloads/recommended/NT4OptPk/ntwx86.asp
Windows NT 4.0 Server	IIS	Download unter http://www.microsoft.com/ntserver/nts/downloads/recommended/NT4OptPk/ntsx86.asp
Windows 2000 Professional/ Server/Advanced Server	IIS	Installation über die Systemsteuerung: Start • Einstellungen • Systemsteuerung • Software • Windows-Komponenten hinzufügen/entfernen
Windows XP Professional	IIS	Installation über die Systemsteuerung

Tabelle 211 Die verschiedenen Bezugsquellen für die Webserver (Forts.)

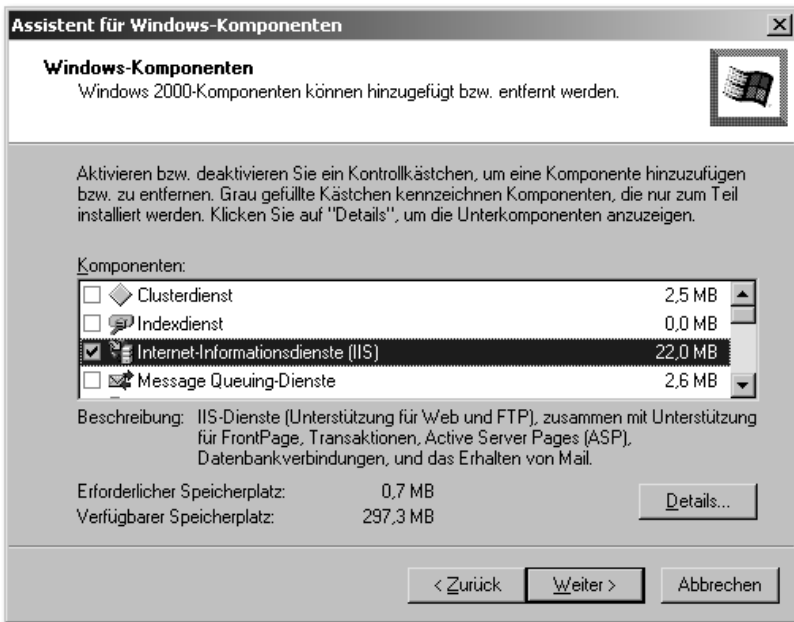


Abbildung 211 Die IIS-Installation unter Windows 2000

Bei allen anderen Windows-Versionen ist kein Webserver enthalten und zumindest von Microsoft keiner verfügbar; eine Notlösung ist hier die Verwendung des Apache-Webservers oder von anderen Produkten. Entgegen anders lautender Gerüchte enthält die Windows XP Home Edition

– als Heimanwender-Betriebssystem vermarktet – keinen Webserver. Das steht sogar im Installationshandbuch.



Unter <http://www.chiliasp.com/> steht ein ASP-Interpreter auch für andere Webserver und Betriebssysteme zur Verfügung. Der ist dann allerdings kostenpflichtig.



Der Anbieter Brinkster, <http://www.brinkster.com/>, bietet kostenlosen Webspace mit ASP- und Datenbankunterstützung an. Die Übertragung der Dateien erfolgt allerdings ziemlich mühsam über die Web-Oberfläche, FTP gibt es nur bei den kostenpflichtigen Paketen.

Auch bei den ASP-Beispielen in diesem Kapitel gehen wir davon aus, dass sowohl ASP als auch der zugehörige Webserver korrekt installiert worden sind und Sie bereits einige Kenntnisse in ASP haben.

Dieses Kapitel besteht aus drei Teilen. Zunächst führen wir Sie in die allgemeine Theorie ein, wie Sie Daten mit serverseitigen Skripten austauschen können. Danach zeigen wir Ihnen einige Beispiele, die wir – jeweils leicht abgeändert, aber aus Gründen der Vergleich- und Nachvollziehbarkeit ähnlich – für PHP und ASP umsetzen.



Auf der CD-ROM zum Buch finden Sie nicht nur alle Quellcodes für diese Beispiele, sondern auch die zum Zeitpunkt der Drucklegung aktuellste Version von PHP und des Apache Webserver.

21.2 Serverseitige Kommunikation

Die Kommunikation zwischen Flash-Film und serverseitigem Programm oder Skript kann entweder in eine oder in beide Richtungen erfolgen. Entweder aus dem Flash-Film wird ein serverseitiges Skript aufgerufen und ausgeführt, ohne dass der Flash-Film am Ergebnis Interesse zeigt, oder das serverseitige Skript gibt eine Rückmeldung an den Flash-Film, die dann wieder ausgewertet werden kann.

21.2.1 Die Funktion `getURL()`

In Kapitel 17, ActionScript mit JavaScript, haben Sie bereits die ActionScript-Funktion `getURL()` in Aktion gesehen. Diese ermöglichte es, neue URLs aufzurufen, entweder im aktuellen Browserfenster oder als Frame oder in einem neuen Fenster. Damit ist es auch möglich, serverseitige Skripte aufzurufen. Das allein ist ja nicht besonders aufregend.

- Auf Rückgabewerte der von `getURL()` aufgerufenen Skripte können Sie mit ActionScript nicht zugreifen.
- Ein einfacher Aufruf eines serverseitigen Skripts ist an sich nichts Besonderes; dies könnte auch durch einen Link geschehen.

Der Clou an `getURL()` ist jedoch, dass Sie an das Skript auch Variablen übergeben können, und zwar alle Variablen der aktuellen Sequenz (des aktuellen Movieclips).

Dazu ein kleiner Exkurs. Es gibt mehrere Möglichkeiten, Daten per http (Hypertext Transfer Protocol) an ein Skript zu übergeben. Die zwei gebräuchlichsten sind diese:

- **GET:** Hier werden die Daten an die eigentliche Web-Adresse angehängt. Das geschieht in den folgenden Schritten:
 - Zunächst wird an die Adresse ein Fragezeichen angehängt.
 - Dann werden die Daten in so genannten Name-Wert-Paaren zusammengefasst, nach dem Muster `name1=wert1, name2=wert2, ...`
 - Nun werden alle Sonderzeichen in das URL-Format umgewandelt. Aus Leerzeichen wird beispielsweise ein `+`, alle anderen Sonderzeichen werden in das Format `%XY` gebracht, wobei XY der Hexadezimalcode des Zeichens ist (eine gültige Repräsentation des Leerzeichens wäre somit auch `%20`, weil hexadezimal 20 dezimal 32 entspricht, dem Zeichencode des Leerzeichens). Dies ist allein schon deswegen notwendig, damit beispielsweise nur ein Fragezeichen in der URL enthalten ist; alle Fragezeichen in den Name-Wert-Paaren werden umgewandelt.
 - Anschließend werden die Name-Wert-Paare durch kaufmännische Und-Zeichen (`&`) aneinander gehängt und an die URL angehängt.
 - Die URL sieht dann also beispielsweise folgendermaßen aus: **`http://servername/skriptname.xyz?name1=wert1&name2=wert2&name3=wert3`**
- **POST:** Hier werden die Daten nicht an die URL angehängt, sondern als Teil der HTTP-Anforderung an den Webserver mitgeschickt. Dazu werden die Daten zunächst auch in das URL-Format gebracht, dann aber in der HTTP-Anforderung untergebracht, sind also somit im Browser nicht sichtbar.

Die URL-Codierung der Daten, also das Ersetzen von Sonderzeichen, funktioniert übrigens auch mit ActionScript und JavaScript; das Kommando hierfür lautet `escape()`. Mit `unescape()` können Sie die Umwandlung rückgängig machen.



Jede der beiden Methoden hat ihre Vor- und Nachteile, also auch eine Daseinsberechtigung. GET ist einfacher zu handhaben und das Testen fällt leichter, weil ja die übergebenen Daten in der URL sichtbar sind. Der GET-Aufruf muss zudem nicht immer per Formular realisiert werden, eine einfache URL genügt (diese kann »von Hand« zusammengesetzt werden, wie oben schrittweise skizziert). Allerdings haben Browser, Webserver und Proxyserver gewisse Längenbeschränkungen für URLs, spätestens bei 2048 Zeichen ist Schluss (bei einigen Systemen sogar schon bei 500 Zeichen). Außerdem können mit GET keine Dateien übertragen werden. Für Passwortabfragen schließlich ist GET völlig unbrauchbar, weil dann die geheimen Daten im Klartext in der URL stünden.

POST dagegen ist nicht ganz so transparent – die Daten sind nicht sichtbar, dafür ist die Datenmenge unbegrenzt und auch Dateien können so übermittelt werden. Aus diesem Grund wird in der Praxis meistens POST eingesetzt.

Aber zurück zur Funktion `getURL()`. Wie Sie bereits wissen, können Sie mit `getURL()` externe URLs und auch JavaScript-Funktionen aufrufen. Dabei haben Sie die beiden folgenden Möglichkeiten kennen gelernt:

- ▶ `getURL(URL)` – ruft die angegebene URL im aktuellen Browserfenster auf,
- ▶ `getURL(URL, Fenster)` – ruft die angegebene URL im angegebenen Fenster oder Frame auf.

Es gibt allerdings noch eine weitere Möglichkeit; Sie können an `getURL()` einen dritten Parameter übergeben:

- ▶ `getURL(URL, Fenster, Methode)` – ruft die angegebene URL im angegebenen Fenster oder Frame auf und verwendet dabei die angegebene Methode (entweder "POST" oder "GET").



Der Flash-Projektor verwendet immer GET, POST ist nur im Browser möglich.

Wie bereits kurz erwähnt wurde: Der Clou an `getURL()` ist, dass alle Variablen der aktuellen Sequenz automatisch an das Skript übergeben werden.

Ein kleiner Film soll dieses Vorgehen erläutern. Dazu werden in einem Film eine Reihe von Variablen deklariert und auch zwei Textfelder eingerichtet, die ihrerseits einen Variablennamen haben. In Abbildung 21.2 können Sie einen Blick auf die Bühne dieses Films werfen (Sie sehen die ASP-

Version; bei der PHP-Version steht im unteren Textfeld analog der Text »PHP«).

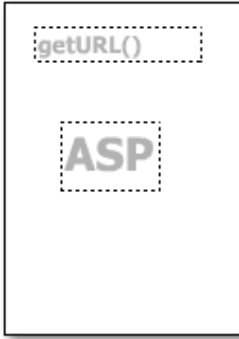


Abbildung 21.2 Die Bühne des Testfilms

Im ersten Bild des Films deklarieren wir zunächst zwei Variablen:

```
var buchtitel = "ActionScript mit JavaScript";  
var autoren = new Array("Christian Wenz", "Tobias Hauser");
```

Dann wird die Funktion `getURL()` aufgerufen:

```
getURL("geturl.php", "_self", "POST");
```

Wenn Sie ASP einsetzen, sieht der Aufruf folgendermaßen aus:

```
getURL("geturl.asp", "_self", "POST");
```

Der Fenstername `"_self"` steht für das aktuelle Fenster. Um die Versandmethode als dritten Parameter angeben zu können, muss der zweite Parameter unbedingt gesetzt werden.

Da sich das Skript **geturl.php** bzw. **geturl.asp** im selben Verzeichnis befindet wie die **.swf**-Datei, können wir den Servernamen weglassen, müssen also nicht **http://servername/geturl.asp** als Parameter angeben.

Auf der CD-ROM zum Buch finden Sie die PHP-Variante dieses Films unter dem Dateinamen **geturl-php fla**, die ASP-Variante unter dem Dateinamen **geturl-asp fla**.

Nun müssen wir nur noch die Skripte erstellen, die alle per POST übergebenen Variablen ausgeben.



Bei PHP ist hierfür das globale Array `$_POST[]` zuständig. Im folgenden Code werden per Schleife alle Werte ausgelesen und in einer HTML-Tabelle ausgegeben:

```
<html>
<head>
<title>getURL() und PHP</title>
</head>
<body>
<table border="0" cellpadding="10">
<th><td>Name</td><td>Wert</td></th>
<?php
foreach ($_POST as $name => $wert) {
    echo("<tr>");
    echo("<td>" . htmlspecialchars($name) . "</td>");
    echo("<td>" . htmlspecialchars($value) . "</td>");
    echo("</tr>\n");
}
?>
</table>
</body>
</html>
```

Listing 211 Das PHP-Skript gibt alle per POST übergebenen Daten aus.



Das Array `$_POST[]` wurde in PHP 4.1.0 eingeführt und ersetzt dort das alte Array `$HTTP_POST_VARS[]`, das jetzt als veraltet gilt. Wenn Sie eine ältere PHP-Version einsetzen, müssen Sie `$HTTP_POST_VARS[]` verwenden.

Analog funktioniert das ASP-Skript:

```
<html>
<head>
<title>getURL() und ASP</title>
</head>
<body>
<table border="0" cellpadding="10">
<tr><th>Name</th><th>Wert</th></tr>
<%
Dim name
For Each name In Request.Form
    Response.Write "<tr><td>"
    Response.Write Server.HtmlEncode(name)
```

```

        Response.Write "</td><td>"
        Response.Write Server.HtmlEncode(Request.Form(name))
        Response.Write "</td></tr>" & vbCrLf
    Next
%>
</table>
</body>
</html>

```

Listing 21.2 Das ASP-Skript gibt alle per POST übergebenen Daten aus.

Sie finden diese Skripte unter den Dateinamen **geturl.php** und **geturl.asp** auf der CD-ROM zum Buch.



Rufen Sie nun die von Flash erzeugte **.swf**-Datei im Webbrowser auf. Sie werden sofort auf das PHP- bzw. ASP-Skript umgeleitet und sollten – eine korrekte Installation des Webserver und der Skripttechnologie vorausgesetzt – in etwa dasselbe Ergebnis wie in Abbildung 21.3 erhalten: Die Variablen des Films werden ausgegeben.



Abbildung 21.3 Die POST-Daten werden im Browser ausgegeben.

Sie sehen also, dass sowohl per ActionScript erstellte Variablen als auch die Textfelder, die einen Variablennamen haben, übergeben werden. Auch das Array wird übergeben, zumindest seine String-Repräsentation (alle Elemente des Arrays nacheinander, durch Kommata voneinander getrennt).



Wenn Sie den Film im Flash-Player aufrufen, werden die Daten per GET übergeben; Sie sehen dann im Browser nicht diese Werte, weil die Skripte ja auf POST-Daten warten.

Per `getURL()` können Sie also prima Variablen versenden und somit serverseitige Skripte mit Daten füllen, den Rückgabewert können Sie allerdings nicht auslesen. Außerdem wird immer ein neues Fenster geöffnet oder die aktuelle Seite verschwindet, was auch nicht immer im Sinne des Erfinders ist. Eine Alternativlösung wäre also günstig.

21.2.2 Die Funktion `loadVariables()`

Der bereits angedeutete Weg, Rückgabewerte von serverseitigen Skripten auszuwerten, geht über die ActionScript-Funktion `loadVariables()`. Ursprünglich diente die Funktion dazu, Daten aus Textdateien einzulesen. Das Format für diese Daten kommt Ihnen sicherlich bekannt vor:

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

Also muss die Datei die Daten in genau dem Format enthalten, in dem sie bei einem Formularversand per POST verschickt worden wären.

Testen wir dies doch an einem kleinen Beispielfilm. Dieser enthält sieben Textfelder, die in einer Überschrift und zwei Spalten zu je drei Textfeldern angeordnet sind. Die Überschrift und die Felder in der linken Spalte sind jeweils statisch, die in der rechten Spalte sind dynamisch, mehrzeilig (mit Umbruch) und tragen die Variablennamen `text1`, `text2` und `text3`:

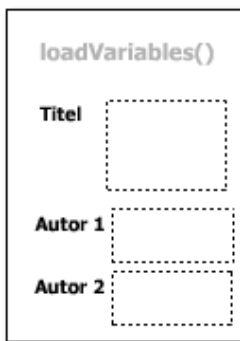


Abbildung 21.4 Die Bühne des Beispielfilms

Im ersten Bild erstellen wir ein Bildskript, das einfach `loadVariables()` aufruft und dabei eine Datei **loadvariables-1.txt** einliest. Der zweite Parameter gibt die Filmsequenz (den Movieclip) an, in die die Variablen gela-

den werden sollen. Eine leere Zeichenkette steht immer für die aktuelle Zeitleiste:

```
loadVariables("loadvariables-1.txt", "");
```

Werfen wir einen Blick auf die Datei **loadvariables-1.txt**. In dieser stehen – in das URL-Format gebracht – drei Variablen:

```
text1=ActionScript+mit+JavaScript&text2=Chris-  
tian+Wenz&text3=Tobias+Hauser
```

Wir umbrechen das einmal ein wenig, damit es etwas übersichtlicher wird:

```
text1=ActionScript+mit+JavaScript  
&text2=Christian+Wenz  
&text3=Tobias+Hauser
```

Nach dem Laden in den Flash-Film sollten also drei Variablen definiert worden sein, `text1`, `text2` und `text3`. Das sind genau die drei Variablen-namen der drei Textfelder in der rechten Spalte.

Wenn Sie diesen Film exportieren und in einem Webbrowser oder im Flash-Player öffnen, sehen Sie das Resultat: Die Variablenwerte aus der Textdatei stehen in den zugehörigen, gleichnamigen Textfeldern (siehe Abbildung 21.5).



Abbildung 21.5 Die Variablen wurden aus der Textdatei eingelesen.



Sie finden diesen Film auf der CD-ROM zum Buch unter dem Dateinamen **loadvariables-1.fla**.

Besonders bemerkenswert: Durch `loadVariables()` wird die URL-Codierung der Daten wieder rückgängig gemacht, aus JavaScript+mit+ActionScript wird also JavaScript mit ActionScript.

Das Laden von Daten aus der Datei dauert einige Zeit. Im oben genannten Film profitieren Sie quasi davon, dass Feldnamen und Variablennamen in der Datei identisch sind. Was aber, wenn Sie den Dateiinhalt in Variablen laden und nicht direkt in Textfelder?

Auf der CD-ROM zum Buch finden Sie einen Beispielfilm mit dem bezeichnenden Dateinamen **loadvariables-gehtnicht.fla**. Dieser Film funktioniert ganz ähnlich wie **loadvariables-1.fla**, jedoch ist die zugrunde liegende Textdatei (**loadvariables-gehtnicht.txt**) eine andere. Hier sehen Sie ihren Inhalt:

```
titel=ActionScript+mit+JavaScript&autor1=Christian+Wenz&autor2=Tobias+Hauser
```

In ein besser lesbares Format gebracht (nur für den Druck):

```
titel=ActionScript+mit+JavaScript
&autor1=Christian+Wenz
&autor2=Tobias+Hauser
```

Nach dem Einlesen müssten also die drei Variablen `titel`, `autor1` und `autor2` existieren. Aus diesem Grund lautet das Bildskript in **loadvariables-gehtnicht.fla** folgendermaßen:

```
loadVariables("loadvariables-gehtnicht.txt", "");

text1 = titel;
text2 = autor1;
text3 = autor2;
```

Wenn Sie diesen Film exportieren und aufrufen, sehen Sie – nichts. Die drei Textfelder bleiben leer (siehe Abbildung 21.6).



Abbildung 21.6 Wo steckt der Fehler?

Den Grund für dieses Verhalten haben wir bereits angedeutet. Um uns selbst zu zitieren: »Das Laden von Daten aus der Datei dauert einige Zeit«. Und das ist auch die Lösung. Nach dem Aufruf von `loadVariables()` fährt der ActionScript-Interpreter mit der Skriptausführung fort, während parallel dazu die Variablen eingelesen werden. Das Ganze geschieht hier also **asynchron**. Deswegen müssen Sie den Film anders konzeptionieren:

- Im ersten Bild rufen Sie `loadVariables()` auf:

```
loadVariables("loadvariables-2.txt", "");
```

- Im zweiten Bild schreiben Sie die Variablenwerte (sofern vorhanden) in die drei Textfelder:

```
text1 = titel;
text2 = autor1;
text3 = autor2;
```

- Im dritten Bild schließlich springen Sie in das zweite Bild zurück, also zum Schreiben der Daten in die Textfelder. Es ist wichtig, dass Sie nicht erneut zum ersten Bild zurückspringen, denn dann würden Sie die Variablen erneut einlesen.

```
gotoAndPlay(2);
```

Und das war es! Wie Sie sich selbst im Flash-Player überzeugen können, werden nun die Textfelder mit Werten gefüllt.



Die zugehörigen Dateien sind **loadvariables-2 fla** und **loadvariables-2.txt**, wobei **loadvariables-2.txt** und **loadvariables-gehtnicht.txt** identisch sind.

Es gibt aber noch eine Möglichkeit, das Ganze etwas performanter zu gestalten. Bei der bisherigen Programmierung wird das Bildskript im zweiten Bild immer wieder ausgeführt, auch wenn die Variablen aus der Datei schon komplett eingelesen wurden. Aus diesem Grund ändern wir das Bildskript für das dritte Bild. Es wird nur noch zurückgesprungen, wenn die letzte der drei Variablen noch nicht eingelesen worden ist; andernfalls wird der Film angehalten:

```
if (autor2)
    stop();
else
    gotoAndPlay(2);
```



Ein häufig angewandter Kniff besteht darin, an das Ende der Textdatei einen Dummy-Parameter zu stellen, der das Dateiende symbolisiert, beispielsweise **&status=ok**. Die Abfrage könnte dann folgendermaßen aussehen:

```
if (status)
    stop();
else
    gotoAndPlay(2);
```

Nun haben Sie gesehen, wie Sie mit `loadVariables()` Daten aus externen Dateien auslesen können. Sie können natürlich anstelle statischer Textdateien auch Skripte verwenden. Wieso das möglich ist, sehen Sie, wenn Sie die **.swf**-Datei von vorn und die zugehörige Textdatei auf einen Webserver überspielen und sie dann in einem Webbrowser aufrufen.

Wie Sie Abbildung 21.7 entnehmen können, werden auch hier die Daten aus der Datei geladen. Da wir den Flash-Film via HTTP angefordert haben, bedeutet das, dass auch die Textdatei per HTTP abgerufen worden ist. Es ist also gleichermaßen möglich, diese Daten per Skript zu erzeugen.

Dazu schreiben wir zwei sehr einfache Skripte, eines in PHP, eines in ASP. Dort setzen wir zunächst den MIME-Typ der Seite auf `application/x-www-form-urlencoded`; dies erleichtert es Flash, die zurückgelieferten Daten korrekt zu interpretieren. Dann wird einfach die Zeichenkette wie im vorhergehenden Beispiel ausgegeben.

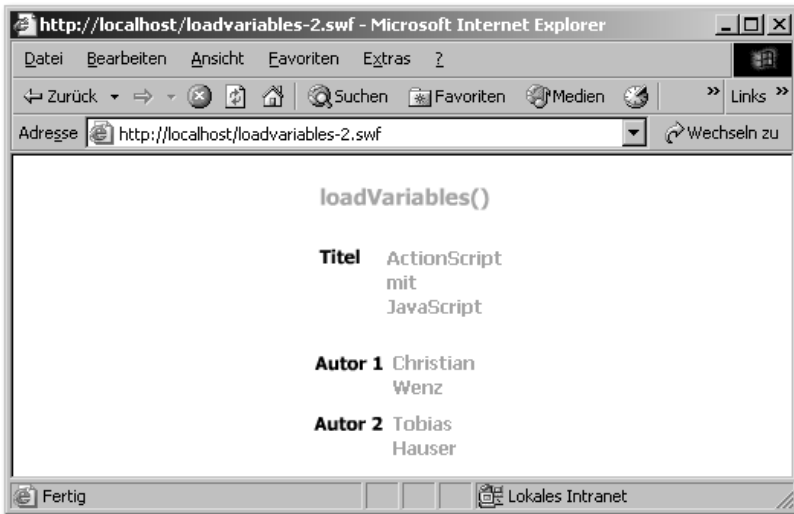


Abbildung 21.7 Das Beispiel funktioniert auch via Webserver.

Hier sehen Sie den zugehörigen PHP-Code:

```
<?php
header("Content-type: application/x-www-form-urlencoded");
echo("titel=ActionScript+mit+JavaScript&autor1=
Christian+Wenz&autor2=Tobias+Hauser&");
?>
```

Listing 21.3 Das PHP-Skript zur Ausgabe der Daten

Und hier sehen Sie das entsprechende ASP-Pendant:

```
<%
Response.ContentType = "application/x-www-form-urlencoded"
Response.Write "titel=ActionScript+mit+JavaScript&autor1=
Christian+Wenz&autor2=Tobias+Hauser&"
%>
```

Listing 21.4 Das ASP-Skript zur Ausgabe der Daten

Wenn Sie Pech haben, wird nach `Hauser` noch eine Leerzeile ausgegeben. Flash interpretiert dann den Zeilenumbruch als Teil der Variablen `autor2`, was natürlich nicht erwünscht ist. Deswegen leiten wir durch das kaufmännische Und (&) am Ende des Strings eine neue, leere Variable ein; der Wert von `autor2` endet damit mit dem `r` in `Hauser`.



Sie müssen nun den letzten Film so modifizieren, dass anstelle der Textdatei das Skript ausgeführt wird:

```
loadVariables("loadvariables.php", "");
```

beziehungsweise

```
loadVariables("loadvariables.asp", "");
```

Die beiden Skripte werden wie gesagt via HTTP aufgerufen. Aus diesem Grund wird nicht der Quelltext dieser Skripte übermittelt, sondern das Resultat der Skripte. Sie können aber selbstverständlich auch den vollständigen Pfad zum Skript angeben, beispielsweise **http://servername/loadvariables.xxx**.



Aus Sicherheitsgründen können Sie von einem Skript aus nur Skripte auf demselben Server (genauer: unterhalb derselben Second-Level-Domain) aufrufen. Ein Skript auf **http://www.galileo-press.de/** kann also keine Datei von **http://www.galileocomputing.de/** abrufen!



Erinnern Sie sich noch an die Methode `getURL()` und die damit verbundene Möglichkeit, alle Variablen einer Sequenz an ein Skript zu schicken, entweder per GET oder per POST? Mit `loadVariables()` ist auch dies möglich, die Syntax ist dieselbe. Im dritten Parameter können Sie die Versandmethode als Zeichenkette angeben ("GET" oder "POST"), wobei GET der Standardwert ist. Allerdings wird die Ergebnisseite nicht im aktuellen oder in einem neuen Browserfenster geöffnet. Wir werden im Abschnitt Beispiele eine Anwendung dafür vorstellen.

21.3 Anwendungen

Nach dem ganzen theoretischen Vorgeplänkel wollen wir nun noch zeigen, wie Sie diese Techniken auch in einer »Real-Life-Anwendung«, das heißt in der Praxis, einsetzen können. Dazu stellen wir zwei Anwendungen vor, bei denen serverseitige Technologien und die Möglichkeiten von Flash zusammenspielen.

21.3.1 Mailversand

Bei dynamischen, aber HTML-basierenden Webseiten sind Formulare das A und O. Sie ermöglichen es den Benutzern, mit den Betreibern der Website in Kontakt zu treten. Durch ein Formular ist es möglich, das erwünschte Feedback in eine strukturierte Form zu bringen. Dies ist ganz nebenbei auch noch ein beliebter Weg, die allgemeine Kontakt-E-Mail-

sächlich aus einem Textfeld (»Versenden«) und einer Hintergrundfarbe. Diese Schaltfläche verwenden wir später, um die Formulardaten versenden zu können.

Abbildung 21.9 können Sie das endgültige Aussehen des Films entnehmen, zumindest das des ersten Bildes. Sie finden das komplette Beispiel – inklusive der kompletten ActionScript-Programmierung – in den Dateien **mailversand-php.fl**a beziehungsweise **mailversand-asp.fl**a.

The image shows a rectangular frame representing a Flash movie clip. Inside the frame, at the top, is the title 'Mailversand' in a bold, sans-serif font. Below the title are three text input fields, each with a label to its left: 'Vorname', 'Nachname', and 'Kommentar'. The input fields are represented by gray rectangles with a dashed border. At the bottom of the frame is a button labeled 'Versenden' in a bold, sans-serif font, enclosed in a rectangular box.

Abbildung 21.9 So sieht der Film in Flash aus.

Nach dem Anklicken der Versenden-Schaltfläche sollen die Formulardaten per E-Mail verschickt werden. Dafür wird `loadVariables()` eingesetzt. Außerdem wird der Film in Gang gesetzt, damit der Abspielkopf zum zweiten Bild weiter geht; dort wird während dem Versand ein Bitte-warten-Text angezeigt:

```
on (release) {  
    loadVariables("mailversand.php", "", "POST");  
    play();  
}
```

beziehungsweise, beim Einsatz von ASP:

```
on (release) {  
    loadVariables("mailversand.asp", "", "POST");  
    play();  
}
```

In den nächsten drei Bildern wird ein Wartetext angezeigt. Wie schon in Kapitel 16, Preloader, wird durch drei Punkte eine Animation simuliert.

Im fünften Bild wird schließlich überprüft, ob der Versand bereits erfolgt ist. In den Skripten zum Mailversand, die wir im Anschluss erstellen werden, geben wir am Ende nach dem Versand eine Meldung aus, ob alles funktioniert hat oder nicht. Bei Erfolg geben wir `status=ok` zurück, ansonsten `status=fehler`. Aus diesem Grund enthält das fünfte Bild folgendes Bildskript:

```
if (status == "ok")  
    gotoAndPlay(7);  
else if (status != "fehler")  
    gotoAndPlay(2);
```

Dieser Code hat folgende Auswirkungen:

- ▶ Wenn vom Skript `status=ok` zurückgegeben worden ist, springt der Film zu Bild 7.
- ▶ Wenn **nicht** `status=fehler` zurückgegeben worden ist, springt der Film zu Bild 2. Dies ist also genau dann der Fall, wenn vom serverseitigen Skript noch keine Rückmeldung erfolgt ist.
- ▶ Im dritten und übrig bleibenden Fall schließlich, beim Rückgabewert `status=fehler`, passiert in diesem Bild nichts. Konsequenz: Der Film läuft weiter zu Bild 6.

In Bild 6 wird also eine Fehlermeldung ausgegeben. Per Schaltfläche kann der Benutzer wieder zu Bild 1 zurückspringen und erhält wieder sein Formular. Bild 7 schließlich enthält eine Dankesmeldung. Bei beiden Bildern wird der Filmablauf mit `stop()` angehalten.

Fehlt nur noch das PHP- bzw. ASP-Skript zum Mailversand. Dazu müssen alle zuvor per POST übertragenen Variablen abgefragt und per Mail verschickt werden. Außerdem muss das Ergebnis des Mailversands ausgegeben werden.

Beginnen wir mit PHP. Als Basis dient das Skript von vorhin, das alle übergebenen Daten per `foreach`-Schleife ausgibt. Nur wird dieses Mal keine

HTML-Tabelle erzeugt, sondern die Daten werden direkt an das Mailprogramm übergeben.

```
<?php
$mailtext = "Daten aus Flash-Formular:\n\n";
foreach ($_POST as $name => $wert) {
    $mailtext .= $name . ": " . $wert . "\n";
}
$ergebnis = @mail("webfeedback@xy.de", "Flash-Feedback",
$mailtext);
if ($ergebnis) {
    echo("status=ok&");
} else {
    echo("status=fehler&");
}
?>
```

Listing 21.5 Mailversand per PHP



Stellen Sie sicher, dass Ihr System für den Mailversand korrekt konfiguriert ist. Werfen Sie dazu in der Konfigurationsdatei **php.ini** einen Blick auf den Abschnitt [mail].

Bei ASP läuft der Vorgang ganz analog ab. Auch hier werden per Schleife alle POST-Daten abgefragt und wird daraus eine E-Mail erzeugt. Je nach Ergebnis des Mailversands erhalten Sie eine positive oder eine negative Rückmeldung:

```
<%
On Error Resume Next

Dim name
Dim mailtext
mailtext = "Daten aus Flash-Formular:" & vbCrLf & vbCrLf
For Each name In Request.Form
    mailtext = mailtext & name & ": "
    mailtext = mailtext & Request.Form(name) & vbCrLf
Next

Set mail = Server.CreateObject("CDONTS.Newmail")
mail.From = "webfeedback@xy.de"
mail.To = "webfeedback@xy.de"
mail.Subject = "Flash-Feedback"
```

```

mail.Body = mailtext
mail.Send
If Err.Number = 0 Then
    Response.Write "status=ok&"
Else
    Response.Write "status=fehler&"
End If
Set mail = Nothing
%>

```

Listing 21.6 Mailversand mit ASP

Sie können nun den Film im Webbrowser testen. Veröffentlichen Sie dazu den Film, um eine **.swf**-Datei und die dazugehörige HTML-Datei zu erhalten. Legen Sie beide Dateien auf Ihrem Webserver in dasselbe Verzeichnis wie das PHP- oder ASP-Skript und rufen Sie die HTML-Datei auf. Wenn alles funktioniert, erhalten Sie nach dem Versand des Skripts und einer kurzen Wartepause eine positive Rückmeldung, wie sie in Abbildung 21.10 zu sehen ist.

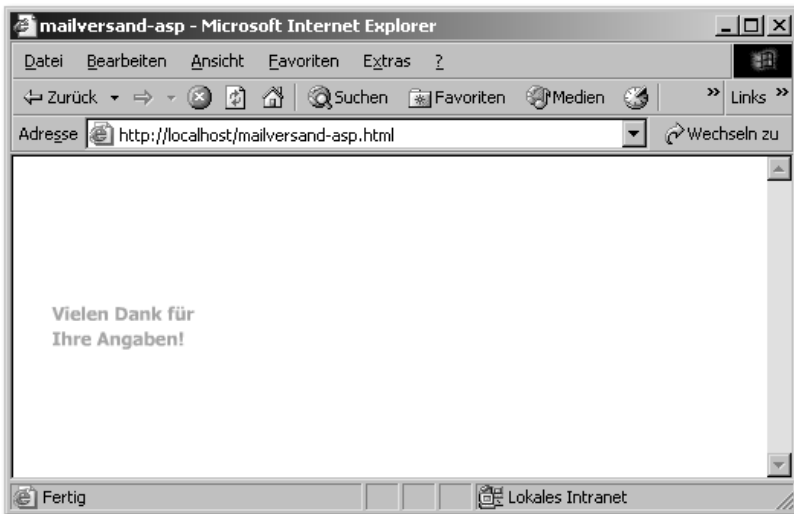


Abbildung 21.10 Der Versand hat funktioniert.

Sollte dagegen etwas schief gegangen sein, erhalten Sie eine Fehlermeldung (siehe Abbildung 21.11).

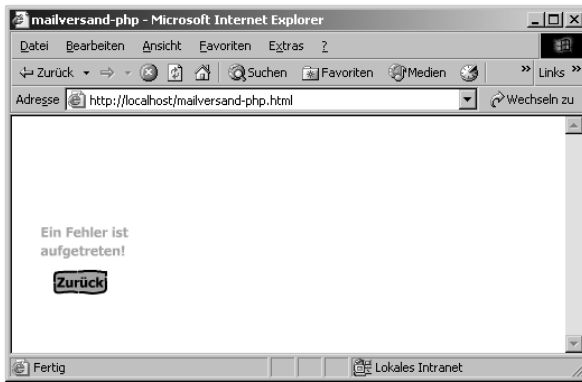


Abbildung 21.11 Der Versand hat nicht funktioniert.



Wenn gar nichts passiert und Sie immer nur »Bitte warten« lesen, könnte ein Syntaxfehler in Ihrem Skript vorliegen. Testen Sie dazu das PHP- oder ASP-Skript direkt im Webbrowser und halten Sie nach Fehlermeldungen Ausschau.

Wenn der E-Mail-Versand funktioniert hat, werden Ihnen die Formular-
daten per E-Mail zugesandt (siehe Abbildung 21.12). Falls nicht, prüfen Sie
die Logdateien Ihres SMTP-Servers, um die Fehlerquelle zu finden und
den Fehler beheben zu können.



Bevor Sie das Beispiel von der CD-ROM testen, ersetzen Sie die E-Mail-
Adressen (bei uns stets **webfeedback@xy.de**) durch Ihre eigene Adresse!



Abbildung 21.12 So kommt die E-Mail bei Ihnen an.

21.3.2 Dynamischer News-Ticker

Im letzten Beispiel haben wir die serverseitigen Sprachen dazu benutzt, Daten aus dem Flash-Film per E-Mail weiterzuleiten. Im Flash-Film selbst war davon gar nicht so viel zu bemerken, lediglich das Ergebnis des Mailversands wurde angezeigt. In diesem Beispiel wollen wir einen anderen Weg gehen.

Auf fast jeder Website befindet sich ein Werbebanner. Anfänglich haben sich Betreiber von gut frequentierten Webseiten beinahe eine goldene Nase damit verdient, heutzutage decken die Werbeeinnahmen im World Wide Web oft kaum noch die Kosten, die die Instandhaltung des Webauftritts verursacht.

Hauptursache hierfür: Die Bereitschaft, Werbebanner anzuklicken, ist in den letzten Jahren kontinuierlich gesunken. Eine Möglichkeit, die Anklickraten wieder etwas zu erhöhen, sind gut gemachte, interessante Banner. Häufig sieht man hier im World Wide Web Banner, die einen tatsächlichen Mehrwert anbieten, beispielsweise aktuelle Nachrichten anzeigen.

In diesem Abschnitt möchten wir ein etwas komplexeres Beispiel entwickeln. In einem Werbebanner werden nacheinander eine Reihe von Nachrichten angezeigt. Per Mausklick kann jeweils eine zu der Nachricht gehörende Webadresse aufgerufen werden. Die Nachrichten werden dabei jeweils aktuell aus einer Datenbank gezogen. Hier kommt dann auch das serverseitige Skript mit ins Spiel: Die PHP- oder ASP-Seite liest Daten aus der Datenbank aus und gibt diese als Name-Wert-Paare aus. Die ActionScript-Funktion `loadVariables()` erledigt dann den Rest.

Wir haben ein ganz ähnliches Banner bereits in Kapitel 18, XML, erstellt. Der Datenzugriff erfolgt jedoch in diesem Beispiel auf eine völlig andere Art und Weise, außerdem bauen wir ein paar kleine Extras ein. Aus diesem Grund beschreiben wir den Banneraufbau erneut von Grund auf.



Flash-Film

Beginnen wir zunächst mit dem Film. Für Werbebanner gibt es im World Wide Web eine Standardgröße, 468 * 60 Pixel. Diese können Sie über die Dokumenteigenschaften einstellen.

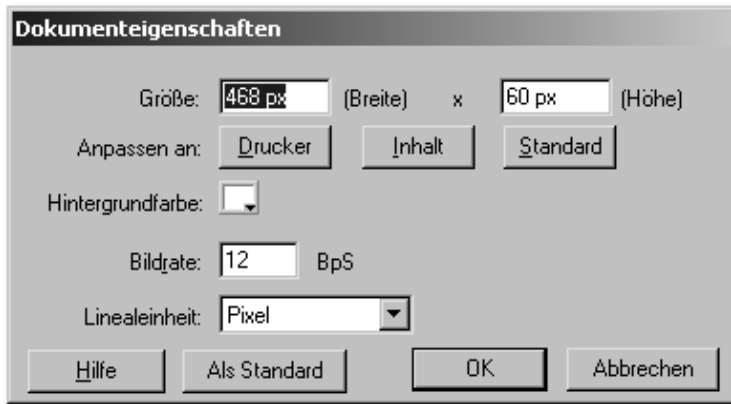


Abbildung 21.13 Die Dokumenteigenschaften für den Film

Wir fahren mit einer Reihe von Textfeldern fort. Das mittlere, größere Textfeld ist dynamisch und wird später die einzelnen Nachrichten enthalten. Aus diesem Grund benötigt es einen Variablennamen (`Newsfeld`). Stellen Sie außerdem sicher, dass der Text im Feld nicht selektiert werden kann (via Eigenschafteninspektor), denn sonst können Sie den Text nicht anklicken. In Abbildung 21.14 sehen Sie, wie der Film mittlerweile in etwa aussieht.

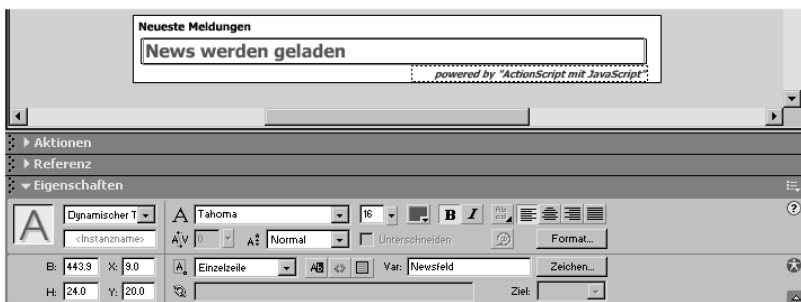


Abbildung 21.14 Der Film und (im Inspektor) die Textfeldeigenschaften

Beginnen wir mit der Programmierung. Am Anfang, in Bild 1, werden zunächst die Variablen aus dem Skript eingelesen:

```
loadVariables("newsbanner.php", "");
```

Das Laden verschlingt einige Zeit, insbesondere wenn eine Datenbank abgefragt werden muss. Also erstellen wir eine kleine Spielerei, mit der die Wartezeit verkürzt werden kann. Es soll, wie schon beim Preloader,

eine kleine Punkte-Animation realisiert werden. Dieses Mal jedoch erzeugen wir sie komplett mit ActionScript. Wir gehen folgendermaßen vor:

- Wir werden abwechselnd keinen, einen, zwei oder drei Punkte nach dem »News werden geladen« anzeigen. Die Anzahl der gerade angezeigten Punkte wird in einer Variablen gespeichert.
- Per `setInterval()` rufen wir periodisch eine Funktion auf, in der ein weiterer Punkt an den Text angefügt wird. Wenn die drei Punkte »voll« sind, wird wieder bei null Punkten begonnen.

Hier sehen Sie den Code, der dies erledigt:

```
var punkteID;  
var punktezahl = 0;  
  
function Punkte() {  
    Newsfeld = "News werden geladen ";  
    for (var i=1; i<=punktezahl; i++)  
        Newsfeld += ".";  
    punktezahl ++;  
    if (punktezahl > 3)  
        punktezahl = 0;  
}
```

```
Punkte();  
punkteID = setInterval(Punkte, 250);
```

Durch `setInterval(Punkte, 250)` wird die Funktion `Punkte()` das erste Mal nach 250 Millisekunden aufgerufen. Da wir nicht so lange warten möchten, rufen wir sie davor einmal direkt auf.



Im dritten Bild fügen wir ein neues Schlüsselbild ein. Nun müssen wir überprüfen, ob die Daten vom serverseitigen Skript bereits komplett geladen worden sind. Wir sehen vor, dass das Skript die Meldungen und zugehörigen Links im folgenden Format zurückliefert (aus Gründen der Übersichtlichkeit umbrochen, gehört natürlich in eine Zeile):

```
text1=Text  
&link1=Link  
&text2=anderer+Text  
&link2=anderer+Link  
&anzanzahl=2
```

Die letzte Variable, `anzahl`, erfüllt zwei Funktionen. Zum einen ermöglicht sie dem ActionScript-Code festzustellen, wie viele Meldungen überhaupt übertragen worden sind – das kann sich ja je nach Datenbestand ändern. Zum zweiten wissen wir aber auch: Wenn die Variable `anzahl` vorliegt, sind alle Meldungen übermittelt worden, die News können also angezeigt werden.

Wir fahren also wie folgt fort:

- ▶ Wenn die Variable `anzahl` gesetzt worden ist,
 - ▶ halten wir den Film an;
 - ▶ beenden wir die Animation mit den Punkten;
 - ▶ zeigen wir die Meldungen an.
- ▶ Ist die Variable `anzahl` dagegen noch nicht gesetzt, gehen wir zu Bild 2, warten also weiterhin auf Antworten vom Skript.

Wir werden später eine Funktion `ZeigeNews()` erstellen, welche die News im Film anzeigt. Der Code für Bild 3 kann dann folgendermaßen aussehen:

```
if (anzahl) {
    stop();
    clearInterval(punkteID);
    ZeigeNews();
    zeigeID = setInterval(ZeigeNews, 3000);
} else
    gotoAndPlay(2);
```

Listing 21.7 Das Bildskript für Bild 3

Nun zur Anzeige der News, `ZeigeNews()`. In der Variablen `pos` wird die Nummer der aktuellen Meldung gespeichert. Wenn `ZeigeNews()` aufgerufen wird, wird diese Zahl um 1 erhöht (und auf 1 zurückgesetzt, falls sie größer als die Gesamtanzahl an Meldungen ist). Dann wird der Wert der Variablen `textX` im Textfeld `Newsfeld` ausgegeben, wobei `X` für die Nummer der aktuellen Meldung steht. Dies funktioniert mit `eval()`: `eval("text" + pos)` liefert den zugehörigen Meldungstext. Damit ist `ZeigeNews()` sehr schnell erstellt:

```
var pos = 0;

function ZeigeNews() {
    pos ++;
```

```

    if (pos > anzahl)
        pos = 1;
    Newsfeld = eval("text" + pos);
}

```

Fehlt eigentlich nur noch eines (zumindest von der Flash-Seite aus) – der Klick auf den Nachrichtentext muss abgefangen werden. Dazu müssen Sie ein neues Symbol erstellen. Innerhalb dieses Symbols erstellen Sie beispielsweise ein Rechteck, das als Umrandung für den Nachrichtentext dient. Platzieren Sie dieses Rechteck an derselben Position wie das dynamische Textfeld. Dann müssen Sie für den Fall, dass dieser Bereich mit der Maus angeklickt wird, eine Aktion definieren:

```

on (release) {
    FolgeLink();
}

```

Die Funktion `FolgeLink()` ruft dann den angegebenen Link auf. Auch hierzu wird wieder die Variable `pos` eingesetzt und mit `eval("link" + pos)` der aktuelle Link ermittelt. Sie müssen lediglich noch den Fall abfangen, dass `pos` den Wert 0 hat, denn dann sind die News noch nicht geladen worden (das serverseitige Skript arbeitet also noch).

```

function FolgeLink() {
    if (pos > 0)
        getURL(eval("link" + pos), "_blank");
}

```

Noch einmal zur Erinnerung: Durch den Parameter `"_blank"` wird dafür gesorgt, dass der Link in einem neuen Fenster geöffnet wird. Wenn Sie dies vermeiden möchten und den Link im aktuellen Browserfenster öffnen möchten, lassen Sie den zweiten Parameter für `getURL()` weg oder verwenden Sie `"_self"`.



Nachfolgend das komplette Bildskript für Bild 1 des Films:

```

loadVariables("newsbanner.php", "");

var punkteID;
var pos = 0;

function ZeigeNews() {
    pos ++;
    if (pos > anzahl)

```

```

        pos = 1;
        Newsfeld = eval("text" + pos);
    }

    function FolgeLink() {
        if (pos > 0)
            getURL(eval("link" + pos), "_blank");
    }

    var punktezahl = 0;
    function Punkte() {
        Newsfeld = "News werden geladen ";
        for (var i=1; i<=punktezahl; i++)
            Newsfeld += ".";
        punktezahl ++;
        if (punktezahl > 3)
            punktezahl = 0;
    }

    Punkte();
    punkteID = setInterval(Punkte, 250);

```

Listing 21.8 Das Bildskript für Bild 1



Für die ASP-Variante müssen Sie den Aufruf von **newsbanner.php** durch den Aufruf von **newsbanner.asp** ersetzen. Auf der CD-ROM finden Sie aber wie gewohnt die Flash-Filme, diesmal unter den Dateinamen **newsbanner-php fla** und **newsbanner-asp fla**.

Serverseitige Skripte

Kommen wir nun zur serverseitigen Programmierung. Wir beginnen mit PHP und der Erstellung der zugehörigen MySQL-Datenbank. Erstellen Sie dazu:

- ▶ eine Datenbank namens `flashnews`,
- ▶ darin eine Tabelle namens `meldungen`.
- ▶ Diese Tabelle enthält drei Spalten:
 - ▶ `id` – INT (`auto_increment`)
 - ▶ `text` – VARCHAR(255)
 - ▶ `link` – VARCHAR(255)

Besonders bequem geht das mit einem grafischen Frontend wie beispielsweise dem bereits erwähnten **phpMyAdmin**. Sie können für das Erstellen der Tabelle aber auch die entsprechenden SQL-Kommandos verwenden:

```
CREATE TABLE "meldungen" (
    "id" INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    "text" VARCHAR(255) NOT NULL,
    "link" VARCHAR(255) NOT NULL
);
```

Erzeugen Sie dann einige Einträge in der Datenbank, damit der Flash-Film auch mit Daten gefüllt werden kann.

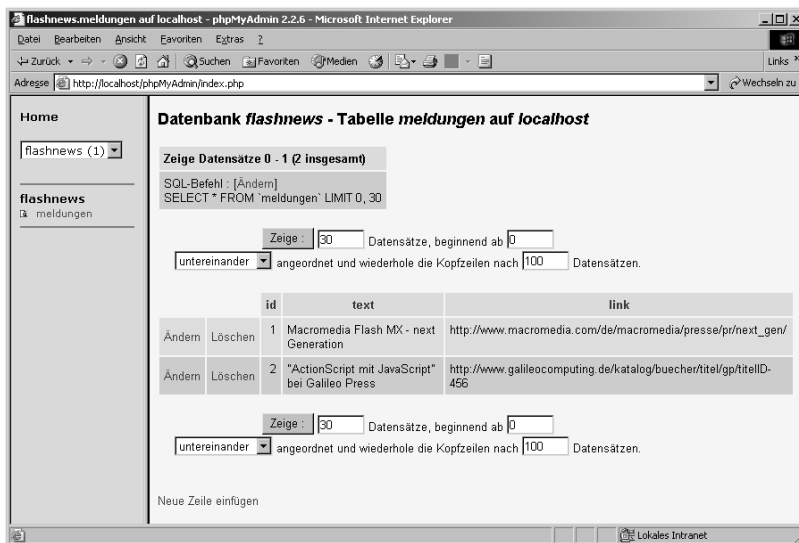


Abbildung 2115 phpMyAdmin in Aktion

Nun müssen Sie nur noch ein Skript schreiben, das folgende Aufgaben erledigt:

- ▶ Auslesen aller Nachrichten aus der Datenbank,
- ▶ Aufbereitung in URL-codierte Name-Wert-Paare,
- ▶ Ausgabe der Daten.

Hier sehen Sie das komplette Skript (Sie finden es unter dem Namen **newsbanner.php** auf der CD-ROM zum Buch):

```
<?php
header("Content-type: application/x-www-form-urlencoded");
```



```

$rueckgabe = "";
$zaehler = 0;

mysql_connect("localhost", "Benutzer", "Passwort");
mysql_select_db("flashnews");

$ergebnis = mysql_query("SELECT * FROM meldungen");

while ($zeile = mysql_fetch_array($ergebnis)) {
    $zaehler ++;
    $rueckgabe .= "&text$zaehler=";
    $rueckgabe .= urlencode($zeile["text"]);
    $rueckgabe .= "&link$zaehler=";
    $rueckgabe .= urlencode($zeile["link"]);
}

mysql_close();

$rueckgabe .= "&anzahl=$zaehler&";
echo($rueckgabe);
?>

```

Listing 21.9 Das PHP-Skript liest die Meldungen aus der Datenbank aus.

Dieses Skript müssen Sie nur noch in dasselbe Verzeichnis kopieren wie die aus dem Flash-Film generierten **.swf**- und HTML-Dateien (natürlich auch auf CD-ROM), dann können Sie das Beispiel testen und einsetzen.



Sie müssen natürlich in `mysql_connect()` Ihren eigenen Servernamen, den Benutzernamen und das Passwort einsetzen, damit das Skript bei Ihnen funktioniert.



Wenn der Flash-Film doch nicht so funktioniert, wie er soll, versuchen Sie doch einmal, das PHP-Skript allein aufzurufen; eventuell stoßen Sie auf Fehlermeldungen, welche dann den Flash-Film durcheinander bringen.

Wenn Sie ASP einsetzen, gehen Sie genauso vor. Zunächst muss eine Datenbank erstellt werden. Wir setzen dafür Access ein. Auch, wenn Sie nicht über Access verfügen, können Sie das Beispiel zumindest auf Ihrem Webserver einsetzen. Die Access-Treiber sind bei Windows enthalten.

Es ist eine gute Idee, unter <http://www.microsoft.com/data/> die neuesten Versionen der Treiber (**Microsoft Data Access Components**) herunterzuladen und zu installieren.



Die Access-Datenbank besteht – analog zum MySQL-Pendant – aus einer Tabelle `meldungen` mit drei Feldern `id`, `text` und `link`. In Abbildung 21.16 sehen Sie die Entwurfsansicht von Access.

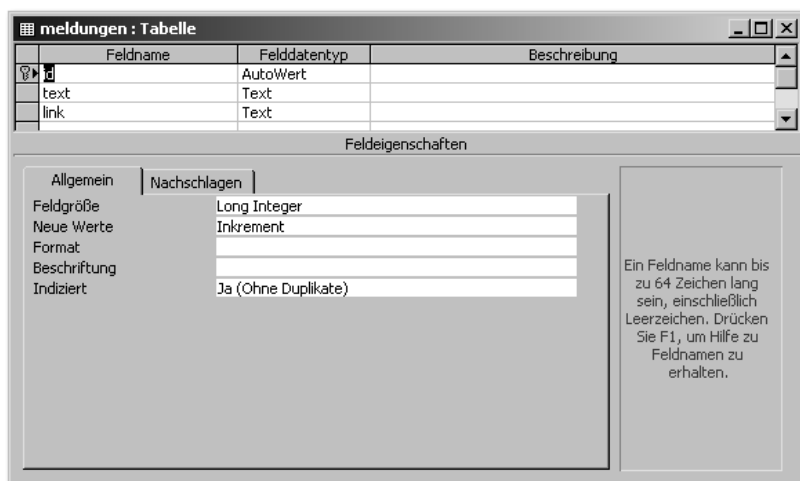


Abbildung 21.16 Die Tabelle in der Entwurfsansicht von Access

Füllen Sie die Datenbank mit ein paar Nachrichten und kopieren Sie die Tabelle in dasselbe Verzeichnis, in der sich die Flash-Dateien (`.swf` und `.html`) und das im Folgenden entwickelte Skript, **newsbanner.asp**, befinden.

Um mit ASP eine Verbindung zu einer Access-Datei aufzubauen, benötigen Sie folgende Anweisung:

```
Set conn = Server.CreateObject("ADODB.Connection")
conn.Open("PROVIDER=Microsoft.Jet.OLEDB.4.0;" & _
          "DATA SOURCE=" & Server.MapPath("flashnews.mdb"))
```

Wenn Sie die Datenbank an einer anderen Stelle ablegen, müssen Sie hinter `DATA SOURCE=` einfach den entsprechenden Dateinamen (mit dem kompletten Pfad) angeben.

Der Rest funktioniert analog zum PHP-Skript:

- ▶ Daten aus der Datenbank einlesen,
- ▶ Name-Wert-Paare erstellen,
- ▶ diese zusammen mit der Anzahl der Nachrichten ausgeben.

Hier sehen Sie das Skript, das Sie auf der CD-ROM unter dem Dateinamen **newsbanner.asp** finden:

```
<%
Response.ContentType = "application/x-www-form-urlencoded"
Dim rueckgabe : rueckgabe = ""
Dim zaehler : zaehler = 0

Dim conn, rs
Set conn = Server.CreateObject("ADODB.Connection")
conn.Open("PROVIDER=Microsoft.Jet.OLEDB.4.0;DATA SOURCE=" & _
          Server.MapPath("flashnews.mdb"))

Set rs = conn.Execute("SELECT * FROM meldungen")

While Not rs.EOF
    zaehler = zaehler + 1
    rueckgabe = rueckgabe & "&text" & zaehler & "="
    rueckgabe = rueckgabe & Server.URLEncode(rs("text"))
    rueckgabe = rueckgabe & "&link" & zaehler & "="
    rueckgabe = rueckgabe & Server.URLEncode(rs("link"))
    rs.MoveNext
Wend

Set rs = Nothing
conn.Close
Set conn = Nothing

rueckgabe = rueckgabe & "&anzahl=" & zaehler & "&"

Response.Write rueckgabe
%>
```

Listing 2110 Das ASP-Skript liest die Meldungen aus der Datenbank aus.

Egal, ob Sie sich für PHP oder ASP entschieden haben, wenn Sie nun den Film im Webbrowser aufrufen, werden nach kurzer Zeit die einzelnen Nachrichten nacheinander eingeblendet.

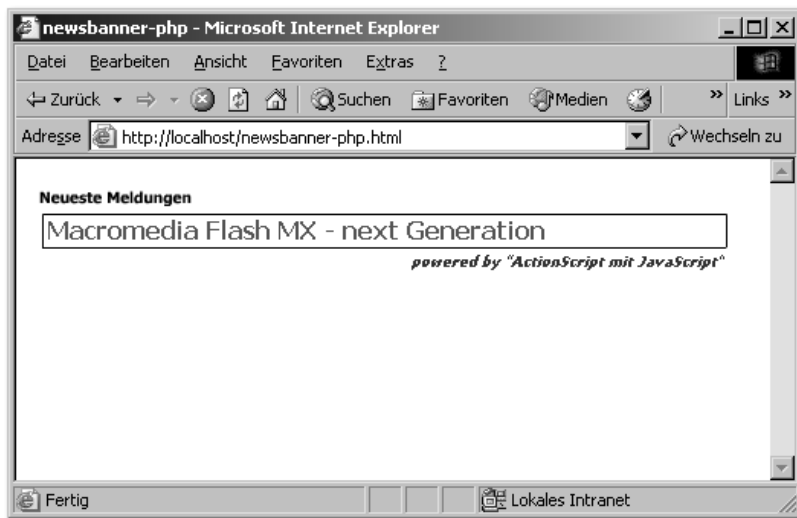


Abbildung 21.17 Das datenbankgestützte Werbebanner

A Unterschiede zwischen ActionScript und ECMAScript

Der Unterschied zwischen Literatur und Journalismus besteht darin, dass der Journalismus unlesbar ist und die Literatur nicht gelesen wird.

– Oscar Wilde

Wenn Sie den ECMAScript-Standard näher betrachten (auf der CD-ROM zum Buch oder direkt unter <http://www.ecma.ch/>), auf dem JavaScript beruht, und ihn mit ActionScript vergleichen, werden Ihnen einige kleine, aber feine Unterschiede auffallen. Wir stellen deswegen an dieser Stelle die wichtigsten Abweichungen vor, um Sie vor Verwechslungen oder auf den ersten Blick unerklärlichen Fehlern zu bewahren.

- ▶ Wenn Sie den Wert `undefined` (beispielsweise den Wert einer zuvor nicht definierten Eigenschaft wie `_root.galileo`) in einen Zahlenwert umwandeln, lautet das Ergebnis nicht etwa `NaN` (wie in JavaScript und ECMAScript), sondern `0`. Der Grund hierfür ist eine erzwungene Abwärtskompatibilität zu Flash 4.
- ▶ Ganz analog: Wenn Sie den Wert `undefined` in einen String umwandeln, lautet das Ergebnis nicht etwa `"undefined"` (wie bei JavaScript/ECMAScript), sondern ist eine leere Zeichenkette. Auch dies ist in der Abwärtskompatibilität zu Flash 4 begründet.
- ▶ In ECMAScript/JavaScript können Sie häufig Rückgabewerte von Funktionen oder Nutzereingaben wie folgt auswerten:

```
if (Zeichenkette)
    // tue irgendetwas.
```

Dies ist eine Kurzform für folgenden Code:

```
if (Zeichenkette.length > 0)
    // tue irgendetwas.
```

Ein String mit der Länge `0` wird also zu `false`, einer mit mindestens einem Zeichen zu `true`. In ActionScript ist das nicht so. Dort wird der String zunächst in eine Zahl umgewandelt. Lautet das Ergebnis `0`, so wird bei `if` daraus `true`, ansonsten `false`. In folgendem Codeschnipsel wird also die `trace`-Anweisung nie ausgeführt:

```
if ("0abc")
    trace("true?!");
```

- In ActionScript können Funktionen direkt im Code definiert werden:

```
setInterval(function() { trace("Funktion!"); }, 1000);
```

In JavaScript/ECMAScript gibt es dafür einen eigenen Function-Konstruktor (`var f = new Function()`), den allerdings Flash nicht unterstützt.

- Einige der Sprachelemente von ECMAScript, insbesondere die zur Fehlerbehandlung (`try`, `catch` – sie werden in diesem Buch nicht behandelt) tauchen in ActionScript überhaupt nicht auf. Das Gleiche gilt für die eingebauten JavaScript-Objekte: `window` und `document` etwa tauchen überhaupt nicht auf, bei `Date` wurden nicht alle Eigenschaften und Methoden implementiert.
- Während in ECMAScript/JavaScript mit `eval()` ganze Funktionsblöcke zur Ausführung kommen können, ermöglicht ActionScript lediglich die dynamische Auswertung von Variablen.

In der Regel werden Sie aber nicht auf die beschriebenen Probleme stoßen – zumindest wenn Sie sauber programmieren! :-)

B Wichtige Tastenkürzel

*Die Kürze ist die Schwester des Talents.
– Anton Tschechow*

Nachfolgend finden Sie die wichtigsten Tastenkürzel bei der Arbeit mit ActionScript in Flash abgedruckt. Unter **Bearbeiten • Tastenkombinationen** können Sie einige dieser Kürzel verändern.

Die Tastenkürzel wurden aus der Windows-Sicht dargestellt; beim Macintosh gelten die bereits bekannten Besonderheiten, Strg etwa entspricht Befehl.



Aktion im Editor	Kürzel
Normalmodus	Strg + ⇧ + N
Expertenmodus	Strg + ⇧ + E
Suchen	Strg + F
Weitersuchen	F3
Ersetzen	Strg + H
Gehe zu Zeile Nummer ...	Strg + G
Code-Hint anzeigen	Strg + ⏏
Syntax prüfen	Strg + T
Automatische Formatierung	Strg + ⇧ + F
Zeilennummern ein-/ausblenden	Strg + ⇧ + L
Importieren aus Datei	Strg + ⇧ + I
Exportieren in Datei	Strg + ⇧ + X
Breakpoint setzen/entfernen	Strg + ⇧ + B
Alle Breakpoints entfernen	Strg + ⇧ + A
Aktion beim Testen des Films	Kürzel
Film im Testmodus starten	Strg + ⇧ + ↵
Film abspielen (im Testmodus)	↵
Debugger ein-/ausblenden	⇧ + F4
Debugging: Weiter	F8

Tabelle B.1 Wichtige Tastenkürzel für ActionScript-Entwickler

Aktion im Editor	Kürzel
Debugging: Überspringen	F9
Debugging: Hineinspringen	F10
Debugging: Verlassen	F11
Debugging: Abbrechen	F7
Variablen auflisten	Strg + Alt + V
Objekte auflisten	Strg + L

Tabelle B.1 Wichtige Tastenkürzel für ActionScript-Entwickler (Forts.)

Index

! (JS) 41
!= 207
!= (JS) 42
!== 209
" 594
#include 183
\$_POST (PHP) 576
\$HTTP_POST_VARS (PHP) 576
% 205
% (JS) 39
%= 205
%=(JS) 40
&& 210
&& (JS) 41
* 205
* (JS) 39
*/ 194
*/ (JS) 29
*= 205
*= (JS) 40
+ 205
+ (JS) 39
++ 205
++ (JS) 39
+= 205
+=(JS) 40
/ 205
/ (JS) 39
/* 194
/* (JS) 29
// 194
// (JS) 29
//--> (HTML) 26
/= 205
/=(JS) 40
< 206
< (JS) 42
<!-- (HTML) 26
<= 207
<= (JS) 42

 (HTML) 45
<div> (HTML) 133, 477
<embed> (HTML) 437, 448, 470
<form> (HTML) 33, 110, 112
<hr> (HTML) 28
 (HTML) 96
<input type= 33
<layer> (HTML) 133
<object> (HTML) 448, 470
<option> (HTML) 114
<script> (HTML) 23, 31
 language 25
<scriptingNews> 524
<select> (HTML) 114
-= 205
= 204
-= (JS) 40
== 206
==(JS) 42
=== 209
> 206
> (JS) 42
>= 206
>= (JS) 42
\\ 199
\' 199
\\ 199
\\b 199
\\f 199
__proto__ 296
_alpha 345, 371
_blank 595
_blank (HTML) 151
_currentframe 371
_droptarget 372
_focusrect 372
_framesloaded 372, 434
_global 242
_height 372
_highquality 303
_name 372
_parent 344
_quality 275, 301
 in der Praxis 301
_root 344
_rotation 351, 372
_self 575, 595
_soundbuftime 304, 403
_target 372
_totalframes 372
_url 372
_visible 367, 372, 541
_width 372
_x 372

_xmouse 372
_xscale 372
_y 373
_ymouse 373
_yscale 373
|| 210
|| (JS) 41

A

abs 333
Abspielkopf
 Steuerung 375
acos 334
ActionScript
 #include 183
 __proto__ 296
 _global 242
 arguments 246
 Array 275
 Bandbreitenoptimierung 422
 Date 216
 UTC-Zeit 328
 Datentypen 195
 drucken 269
 Eigenschaften 288
 _alpha 371, 372
 _currentframe 371
 _droptarget 372
 _focusrect 372
 _framesloaded 372
 _height 372
 _name 372
 _quality 301
 _rotation 372
 _soundbuftime 304
 _target 372
 _totalframes 372
 _url 372
 _visible 372
 _width 372
 _xmouse 372
 _xscale 372
 _y 373
 _ymouse 373
 _yscale 373
 align 397
 attributes 506
 autoSize 392
 background 392
 backgroundColor 392

blockIndent 397
bold 396
border 392
borderColor 392
bottomScroll 392
bullet 397
childNodes 507
color 396
contentType 507
docTypeDecl 507
duration 409
embedFonts 392
firstChild 507
font 396
hasAudio 409
hasAudioEncoder 409
hasMP3 409
hscroll 392
html 392
htmlText 392
ignoreWhite 507
indent 397
italic 396
lastChild 508
leading 397
leftMargin 397
length 310, 392
loaded 508
maxChars 392
maxhscroll 393
maxscroll 393
multiline 393
nextSibling 508
nodeName 508
nodeType 508
nodeValue 509
parentNode 509
password 393
position 409
previousSibling 509
restrict 393
rightMargin 397
screenResolutionX 410
screenResolutionY 410
scroll 393
selectable 393
size 396
status 509
tabIndex 393
tabStops 397

- target 397
- text 393
- textColor 393
- Textfelder 390
- textHeight 393
- textWidth 393
- type 393
- underline 396
- url 396
- variable 394
- wordWrap 394
- xmlDecl 509
- Einsatzgebiete 155
- Ereignisprozeduren 369
 - onData 370
 - onDragOut 370
 - onDragOver 370
 - onEnterFrame 370
 - onKeyDown 370
 - onKeyUp 370
 - onKillFocus 370
 - onLoad 370
 - onMouseDown 371
 - onMouseMove 371
 - onMouseUp 371
 - onPress 371
 - onRelease 371
 - onReleaseOutside 371
 - onRollOut 371
 - onRollOver 371
 - onSetFocus 371
 - onUnload 371
- Ereignisse
 - data 369
 - enterFrame 324, 369
 - keyDown 369
 - keypress 533
 - keyUp 369
 - load 369
 - mouseDow 369
 - mouseMove 369
 - mouseUp 369
 - onChanged 394
 - onKillFocus 394
 - onLoad 409
 - onScroller 394
 - onSetFocus 394
 - onSoundComplete 409
 - unload 369
- externer Code 183
- Fallunterscheidung 211
 - else 213
 - else-if 214
 - if 211
 - switch case 215
- Funktionen 237, 252
 - clearInterval 274
 - duplicateMovieClip 263
 - escape 254
 - eval 254
 - fscommand 260
 - getProperty 254
 - getTimer 254
 - getURL 260
 - getVersion 255
 - gotoAndPlay 256
 - gotoAndStop 256
 - isFinite 201, 255
 - isNaN 255
 - loadMovie 261
 - loadMovieNum 261
 - loadVariables 262
 - loadVariablesNum 262
 - nextFrame 256
 - nextScene 256
 - parseFloat 255
 - parseInt 255
 - play 256
 - prevFrame 256
 - prevScene 256
 - print 269
 - printAsBitmap 269
 - printAsBitmapNum 269
 - printNum 269
 - removeMovieclip 263
 - set 267
 - setInterval 273, 322
 - setProperty 264
 - startDrag 264
 - stop 257
 - stopAllSounds 257
 - stopDrag 264
 - targetPath 255
 - trace 273
 - unescape 254
 - unloadMovie 261
 - unloadMovieNum 261
 - updateAfterEvent 264
- in Flash 171
- Infinity 201

- Klassen
 - eigene 287
- Kommentare 194
- Konstanten
 - E 329
 - LN10 329
 - LN2 329
 - LOG10E 329
 - LOG2E 329
 - PI 329
 - SQRT1_2 329
 - SQRT2 329
- Kontrollstrukturen 210
- Methoden 290
 - abs 333
 - acos 334
 - addListener 394
 - appendChild 504
 - asin 334
 - atan 334
 - atan2 334
 - attachMovie 374
 - attachMovieClip 351, 360
 - attachSound 408
 - ceil 333
 - charAt 313, 314
 - charCodeAt 314
 - cloneNode 504
 - concat 315, 316
 - cos 334
 - createElement 504
 - createEmptyMovieClip 351, 356
 - createEmptyMovieclip 374
 - createTextField 374, 389
 - createTextNode 505
 - delete 316
 - duplicateMovie 374
 - duplicateMovieClip 351
 - exp 333
 - floor 333
 - fromCharCode 315
 - getBounds 374
 - getBytesLoaded 374, 408
 - getBytesTotal 374, 408
 - getDate 326
 - getDay 326
 - getDepth 374, 394
 - getFontList 395
 - getFullYear 327
 - getHours 327
 - getMilliseconds 327
 - getMinutes 327
 - getMonth 327
 - getNewTextFormat 395
 - getPan 408
 - getSeconds 327
 - getTextFormat 395
 - getTime 327
 - getTransform 408
 - getURL 374
 - getVolume 408
 - getYear 327
 - globalToLocal 374
 - gotoAndPlay 375
 - gotoAndStop 375
 - hasChildNodes 505
 - hitTest 375, 536
 - indexOf 312, 315
 - insertBefore 505
 - join 317, 319
 - lastIndexOf 312, 315
 - load 505
 - loadMovie 375
 - loadSound 408
 - loadVariables 375
 - localToGlobal 375
 - log 334
 - Mathematisch 329, 330
 - max 334
 - min 334
 - nextFrame 375
 - parseXML 505
 - play 375
 - pop 317
 - pow 334
 - prevFrame 375
 - push 317
 - random 330, 334
 - removeListener 395
 - removeMovieClip 364, 375
 - removeNode 505
 - removeTextField 395
 - replaceSel 395
 - reverse 317
 - round 330, 334
 - send 506
 - sendAndLoad 506
 - setDate 326
 - setFullYear 327
 - setHours 327

- setMask 375
- setMilliseconds 327
- setMinutes 327
- setMonth 327
- setNewTextFormat 395
- setPan 408
- setSeconds 327
- setTextFormat 395
- setTime 327
- setTransform 408
- setVolume 409
- setYear 327
- shift 317
- sin 334
- slice 315, 317
- sort 318
- sortOn 318
- splICE 318
- split 308, 315
- sqrt 334
- start 409
- startDrag 376
- stop 376, 409
- stopDrag 376
- substr 315
- substring 312, 316
- swapDepths 376
- tan 334
- toLowerCase 316
- toString 318, 319, 327, 506
- toUpperCase 316
- unloadMovie 364, 376
- unshift 318
- valueOf 319, 327
- Objekte
 - Array 316
 - String 306
 - Wahrheitswert 319
- Objektorientierung 286
- Operatoren 204
 - 205
 - 205
 - ! 210
 - != 207
 - !== 209
 - % 205
 - %= 205
 - && 210
 - * 205
 - *= 205
 - + 205
 - ++ 205
 - += 205
 - / 205
 - /= 205
 - < 206
 - <= 207
 - 205
 - = 204
 - == 206
 - === 209
 - > 206
 - >= 206
 - || 210
- Player-Steuerung 262
- Polymorphie 299
- Referenz 187
- return 245
- Schaltflächenereignisse
 - dragOut 163
 - dragOver 163
 - keyPress 163
 - press 163
 - release 163
 - releaseOutside 163
 - rollOut 163
 - rollOver 163
- Schleifen 219
 - do-while 222
 - for 222
 - for in 223
 - while 219
- Schlüsselwörter 193
 - _parent 344
 - _root 344
 - prototype 294
 - this 288, 344
- Skript drucken 183
- Sonderzeichen 198
- Sprachelemente 178
- Suchen und Ersetzen 176
- super 298
- Textfelder 389
- trace 170
- Unterschiede zu ECMAScript 603
- Variablen 191
 - Datentypen 195
 - undefined 192
- Vererbung 294
- with 366

- XML 504
 - Zeichnungsmethoden 356
 - Zugriff auf JavaScript 438
- Active Server Pages 447
- Active Server Pages -> s. ASP
- ActiveX 437
- ActiveX-Control
 - Methoden 472
 - Zusätzliche Methoden 474
- add 204
- addListener 394
- Aktion 172, 178, 253
 - Bedienfeld 172
 - veraltet 253
- Aktionen-Bedienfeld 172
 - Aktuelles Skript immer vorn 175
 - Drucken 183
 - Ersetzen 176
 - Expertenmodus 172
 - Gehe zu Zeilennummer 176
 - Normalmodus 172
 - Objekte 305
 - Position 175
 - Suchen und Ersetzen 176
 - Voreinstellungen 184
 - Werkzeugleiste 174
 - Zeilennummer anzeigen 177
- alert (JS) 76, 79
- align 397
- allowscale 439
- Animation
 - DHTML 141
 - mit JavaScript 105
- Antialiasing 275
- appendChild 504
- arguments 246
- arguments (JS) 58
- Arithmetische Operatoren 204
- Array 196, 275
 - assoziativ 281
 - auslesen 223, 276
 - Datentypen 276
 - durchsuchen 280
 - for 277
 - for-in 278
 - Index 276
 - instanziiieren 276
 - JavaScript 55
 - leer 279
 - length 279
 - Objekt 316
 - Schleife 277
 - Werte 275
 - Werte ändern 280
 - while 278
- asin 334
- ASP 447, 570
- Assoziatives Array 281
- atan 334
- atan2 334
- attachMovie 374
- attachMovieClip 351, 360
- attachSound 407, 408, 535
- attributes 506
- Ausgabe-Fenster 170
- Ausrichten 226
- Auto-Format 415
- Automatische Typkonvertierung 201
- autoSize 392

B

- Back 474
- background 392
- backgroundColor 392
- Bandbreitenoptimierung 422
- Bandbreiten-Profiler 169, 422
- Bedienfeldsätze 187
- Bewegungstween 258
 - drehen 265
 - steuern 259
- Bibliothek 165, 336
 - Sounds 401
- Bild 256
 - Bildmarkierung 272
- Bilder 95
- Bildrate 412
- blockIndent 397
- Bogenmaß 332
 - Umwandlung 332
- bold 396
- Boolean 195, 202
- border 392
- borderColor 392
- bottomScroll 392
- break 216
- break (JS) 48
- Breakpoints 420
 - entfernen 420
- Browser 20
 - Erkennung 71

- Fähigkeiten 409
- Fenster 79
 - anpassen 84
 - Bestätigung 80
 - Größe 85, 86
 - in Fenster schreiben 82
 - mit Rückgabewert 81
 - modal 79
 - öffnen 82
 - Parameter 85
 - verlinken 83
 - Warn 79
- Kommandos 444
- Kompatibilität 22
- Test 470
- Unterscheidung
 - DHTML 137
- Unterschiede 20
- Versionsüberprüfung 27
- Vollbild 450
- Weiterleitung 76
- XML 501
- Ziel 460
- Browser -> s. Webbrowser
- Bühne 156
- bullet 397

C

- capabilities 409
- case 215
- case (JS) 53
- ceil 333
- charAt 313, 314
- charAt (JS) 43
- charCodeAt 314
- childNodes 507, 514
- clearInterval 274, 430, 564
- Client
 - testen 409
- cloneNode 504
- close (JS) 134, 479
- Code
 - auslagern 183
 - Breakpoints 420
 - diktengleich 185
 - einbinden 184
 - importieren 183
 - in importieren Movieclips 184
 - strukturieren 237, 247
 - Syntaxfarbcodierung 185

- Codehinweise 185
- color 396
- Components 551
- concat 315, 316
- contentType 507
- continue (JS) 48
- Cookies 464
- cos 334
- createElement 504
- createEmptyMovieClip 351, 356, 374
- createTextField 374, 389
 - Parameter 389
- createTextNode 505
- CurrentFrame 474

D

- data 369
- Date 320, 604
- Date (JS) 63
- Datenbank
 - Access 598
 - Anbindung 596, 598
 - ASP 598
 - MySQL 596
 - PHP 598
- Datentypen 195
 - Array 196, 276
 - Boolean 195
 - Movieclip 196, 337
 - null 195
 - Object 196
 - String 195
 - typeof 196
 - Typkonvertierung 201
 - undefined 195
 - Vergleich 208
 - Zahl 195
- Datum 216, 320
 - in der Praxis 321
- JavaScript 63
- Umrechnung 328
- Datum -> s. Date
- Debugger 419
 - Breakpoints 420
 - Debug-Modus 420
- default 215
- default (JS) 53
- Deklaration 191
- Dekrement 205
- delete 316

- dependent (JS) 85
- DHTML 131
 - Animation 141
 - browserunabhängig 137
 - Internet Explorer 135
 - Netscape 133
 - W3C 136
 - Werbebanner 148
- directories (JS) 85
- do while (JS) 46
- DOCTYPE 507
- docTypeDecl 507
- document (JS) 471, 604
- Document Type Definition 500
- Document Type Definition -> s. DTD
- document.all (JS) 135, 479
- document.body.clientWidth (JS) 149
- document.body.scrollLeft (JS) 150
- document.body.scrollTop (JS) 150
- document.close (JS) 83
- document.cookie (JS) 491
- document.forms (JS) 112
- document.layers (JS) 133, 479
- document.open (JS) 83
- document.write (JS) 23, 28, 79, 83
- Dokumentenebene 261, 379
- DOM 20, 132
- do-while 222, 278
- Drag-and-Drop 264
- dragOut 163
- dragOver 163
- Drucken 269
 - Bildbeschriftung 270
- DTD 500
- duplicateMovieClip 263, 351, 374
 - in der Praxis 385
- duration 409
- Dynamische Textfelder 169, 196, 527
 - Einstellungen 228
 - HTML 199
 - mehrzeilig 196
- E**
- E 329
- Ebene
 - Dokument 261
- Ebenen
 - Dokument 379
 - Stapel 380
- ECMAScript 20, 155, 603
- Eigenschaften 284
 - ändern 290
 - auslesen 293
 - definieren 288
 - JavaScript 61
 - Movieclip 371
 - Textformat 396
 - vorgefertigte 300
- Eigenschafteninspektor 165, 169
 - Sound 404
- Eingabefelder 169
- else 213
 - in der Praxis 232
 - Kurzform 213
- else (JS) 49
- else-if 214
- embedFonts 392
- Endlosschleifen 221
- enterFrame 369, 537
- Ereignisorientierung 171
- Ereignisprozeduren 162, 164
- Ereignisse 161
 - Ereignisprozeduren 162, 164
 - JavaScript 32
 - Movieclip 162, 367
 - Schaltfläche 162, 163
 - Schlüsselbilder 162
 - Tastatur 533
 - Taste 163
 - Textfelder 394
- Ereignissound 402
- Ereignisprozeduren 369
- Ersetzen 176
- escape 254, 573
- eval 254, 604
- eval (JS) 54, 74, 604
- Event-Handler 162
 - JavaScript 32
- exec 439
- exp 333
- Expertenmodus 172, 180
 - Auto-Format 181, 415
 - Codehinweise 182
 - Syntax überprüfen 181, 416
 - Warnungen 185
 - Zielpfad einfügen 181
- Explizite Typkonvertierung 202
- Externe Filme 378
- Externer Code 183

F

- Fallunterscheidung 211
- false 195, 206
- false (JS) 41
- Film
 - abspielen 256
 - anhalten 257
 - Bildrate 412
 - Dateinamen 449
 - extern 378
 - in der Praxis 380
 - Hierarchie 379
 - laden 261, 378
 - löschen 386
 - Name 470
 - Qualität 275, 301
 - Qualitätsstufen 170
 - Steuerung 169
 - testen 169, 415
 - vorladen 432
- Film-Explorer 188
- Filmsequenz 167, 335
- Filmsteuerung 256, 259, 482
- firstChild 507, 514
- fla 169
- Flash
 - Ansichten 187
 - Datenbankanbindung 596, 598
 - Ebenenkonzept 353
 - eigenständiger Player 262
 - Film-Explorer 188
 - Formular 561
 - Hilfe 188
 - Koordinatensystem 159
 - Player-Version 255
 - serverseitig 569
 - Sound 401
 - UI Components 551
 - XML-Unterstützung 497
 - Zeichensatz 199
- Flash 4 156
 - Ebenenkonzept 353
 - Endlosschleifen 222
 - Film laden 380
 - Funktionen 274
 - String-Operatoren 208
 - Tell Target 366
 - Zugriff auf Movieclips 345
- Flash-Player
 - Kommandos 438
 - Kontextmenü 439
- Flash-Plug-In
 - Methoden 472
- FlashVersion 474
- Float 199
- floor 333
- floor (JS) 70
- Fokus 372
- font 396
- for 222, 277
- for (JS) 44
- for in 223
- for in (JS) 48
- foreach (PHP) 587
- for-in 278
- Formular 561
 - prüfen 562
 - verarbeiten 564
- Formulare
 - Auswahllisten (HTML) 114
 - Checkboxen (HTML) 114
 - Fehlermeldung (JS) 118
 - Formularfelder (HTML) 112
 - JavaScript 109
 - Nutzerinteraktion (JS) 120
 - Radiobuttons (HTML) 113
 - Texteingabefelder (HTML) 125
 - Vollständigkeitsüberprüfung automatisch (JS) 124
- Forward 474
- FrameLoaded 474
- Frames 90
 - Frameset 91
 - in der Praxis 91
 - Mehrere Ändern 93
- frames (JS) 90
- fromCharCode 315, 368
- fscommand 260, 438
 - Befehle 262
 - Code strukturieren 457
- FTP 31
- fullscreen 439
- function 287
- function (JS) 57
- Funktionen 237
 - eigene 237
 - Fehlerquellen 251
 - Filmsteuerung 256
 - global 252
 - Gültigkeitsbereich 239

- in Zeitabständen aufrufen 273
- JavaScript 57
- Literal 239
- Movieclip-Steuerung 263
- Namenskonventionen 238
- Parameter 242
- rekursiv 249, 519
- return 245
- veraltet 274
- verschachteln 250
- vorgefertigte 252

G

- Ganze Zahl 199
- Geometrie 331
- GET 573
- getAscii 368
- getBounds 374, 376
- getBytesLoaded 374, 408, 434
- getBytesTotal 374, 408, 434
- getCode 368
- getDate 326
- getDate (JS) 64
- getDay 326
- getDepth 374, 394
- getElementById (JS) 136
- getFontList 395
- getFullYear 327
- getFullYear (JS) 66
- getHours 322, 327
- getHours (JS) 64
- getMilliseconds 327
- getMilliseconds (JS) 64
- getMinutes 323, 327
- getMinutes (JS) 64
- getMonth 327
- getMonth (JS) 64
- getNewTextFormat 395
- getPan 408
- getProperty 254, 373
- getSeconds 323, 327
- getSeconds (JS) 64
- getTextFormat 395
- getTime 327
- getTimer 254
- getTransform 408
- getURL 260, 374, 460, 529, 572
- getUTCHours 328
- GetVariable 472
- getVersion 255

- getVolume 408
- getYear 327
- getYear (JS) 64
- GIF 105
- Gleitkommazahl
 - Float 199
- Globale Variablen 241
- globalToLocal 374
- GMT 328
- gotoAndPlay 256, 375
- gotoAndStop 256, 375, 414
- GotoFrame 472, 483
- Grafiken 95
- Grafiksymbole 167
- Greenwich Mean Time 328
- Greenwich Mean Time -> s. GMT
- Gültigkeitsbereich 194
 - Funktion 240
 - Variable 240

H

- hasAudio 409
- hasAudioEncoder 409
- hasChildNodes 505
- hasMP3 409
- height (HTML) 96
- height (JS) 85
- hitTest 375, 377, 536
- HMTL
 - Kommentar 26
- href (HTML) 33
- hscroll 392
- HTML 19
 - in Textfeldern 199
 - JavaScript in HTML 23
- html 392
- htmlText 392
- HTTP 31, 463, 573
- Hyperlink 260, 460
 - deaktivieren (JS) 100

I

- id-Attribut (HTML) 135
- IE 135
- if 211
 - in der Praxis 231
 - Kurzform 211
 - verschachteln 212
- if (JS) 49
 - Kurzform 51

- ifFrameLoaded 275, 426
- ignoreWhite 507, 513
- Importieren
 - Sound 402
- indent 397
- Index
 - Array 276
- indexOf 312, 315
- indexOf (JS) 46, 48
- Infinity 201
- Inkrement 205
- innerHeight (JS) 85
- innerHTML (JS) 137
- innerWidth (JS) 85
- insertBefore 505
- Instanz 165
 - Eigenschaften 165
 - Instanzname 337
 - Movieclip 337
 - umwandeln 337
- Instanziiieren 287
- Integer 199
- Integer -> s. Ganze Zahl
- Internet Explorer 437
 - DHTML 135
 - XML 501
- Internet Explorer -> s. IE
- Internet Explorer -> s. Microsoft
 - Internet Explorer
- Internet Explorer -> s. MSIE
- Intervall 273
- isFinite 201, 255
- isNaN 255, 485
- isPlaying 472
- italic 396

J

- JavaScript 19
 - Array 55
 - Index 55
 - auf ActionScript zugreifen 469
 - Browser-Erkennung 71
 - Browsertest 27
 - Datenspeicherung 53
 - Datentypen 36
 - umwandeln 43
 - Datum 63
 - rechnen 67
 - umwandeln 69
 - Dekrement 39
 - Editoren 21
 - Einbau in HTML 23
 - Ereignisse 32
 - Event-Handler 32, 77
 - Externe Datei 30
 - Fallunterscheidungen 49
 - Fenster 79
 - Parameter 85
 - Formulare 109
 - Vollständigkeitsüberprüfung 109
 - Frames 90
 - Zugriff 90
 - Funktionen 57
 - Parameter 57
 - Grafiken 96
 - austauschen 97
 - Browserüberprüfung 101
 - vorladen 98
 - Image-Objekt 95
 - Inkrement 39
 - Kommentare 29
 - Konsole 22
 - Kontrollstrukturen 44
 - location 76
 - Math 69
 - Namenskonventionen 35
 - Navigator 72
 - Objekte 63, 79
 - Objektorientierung 60
 - Operatoren 38
 - arithmetisch 38
 - Konkatenation 44
 - logisch 41
 - String 42
 - Vergleich 42
 - Rollover 99
 - Schleifen 44
 - Sonderzeichen 37
 - String 36
 - Syntax 23
 - Testen 22
 - Variablen 35
 - deklarieren 37
 - Gültigkeit 60
 - Versionen 20, 26
 - Wahrheitswerte 37
 - Weiterleitung 76
 - Y2K 65
 - Zeitversetzt 89

javascript
 (HTML) 32
 (JS) 460
join 317, 319
JPEG 105
JScript 20

K

keyDown 368, 369
keyPress 163, 368
keypress 533
keyUp 369
Klasse 284
Klassen
 definieren 290
 eigene 287
 Parameter 291
Kollisionstest 375, 536
Kommandos
 Flash-Player 438
 Webbrowser 444
Kommentare 194
 JavaScript 29
Komponenten 170, 522, 551
 Ansteuerung 553
 Bedienfeld 170, 552
 CheckBox 553
 ComboBox 554
 ListBox 555
 PushButton 440, 557
 RadioButton 557
 ScrollBar 559
 ScrollPane 560
 vorgefertigt 170
 vorgefertigte 552
Komponenten -> s. Components
Komponenten -> s. UI Komponenten
Konkatenation 204
Konkatenation -> s. String
 verbinden
Konstanten 329
Konstruktor 276, 287, 288
Kontrollstrukturen 210
Koordinatensystem 159
 Movieclip 161
 Registrierungspunkt 166
 Ursprung 159
 x-Achse 159
 y-Achse 159

L

Ladeanzeige 425
language (HTML) 25
lastChild 508
lastIndexOf 312, 315
Latin-1 199
leading 397
left (JS) 85
leftMargin 397
length 279, 310, 392
length (JS) 43, 56
Lineale 159
Literal 239
LiveConnect 437
LiveScript 19
LN10 329
LN2 329
load 369, 505, 511
loaded 508
LoadMovie 472
loadMovie 261, 375
 in der Praxis 384
loadMovieNum 261, 380
loadSound 408
loadVariables 262, 375, 578
loadVariablesNum 262
localToGlobal 375
location (JS) 76
location.href (JS) 93, 139
location.replace (JS) 139
location.search (JS) 93
log 334
LOG10E 329
LOG2E 329
Lokale Variablen 240
Loop 402

M

mailto 31
Mailversand 584
Maske 375
max 334
maxChars 392
maxhscroll 393
maxscroll 393
menubar (JS) 85
Methoden 284
 definieren 290

- Flash-Plug-In und
 - ActiveX-Control 472
 - undokumentiert 473
- JavaScript 61
- Mathematisch 329, 330
- Movieclip 373
- Textfelder 394
- Microsoft Internet Explorer 135
- MIME 507, 582
- min 334
- Modulo 205
- mouseDown 369
- mouseMove 369
- mouseUp 369
- Movieclip 167, 335
 - Datentyp 196, 337
 - duplizieren 351
 - Eigenschaften 371
 - Ereignisse 162, 324, 367
 - Haupt 343
 - Hierarchie 335
 - in der Praxis 338
 - Instanz 337
 - anfügen 351, 360
 - erstellen 350
 - leere erstellen 351, 356
 - löschen 364
 - Instanzname 372
 - Koordinate 372
 - Koordinatensystem 161
 - löschen 364
 - Maße 372
 - Methoden 373
 - mit Code importieren 184
 - Pfad 181, 348, 372
 - skalieren 372
 - Steuerung 263
 - Symbol 336
 - Exportieren 361
 - Tiefe 352
 - unsichtbar 336
 - Verknüpfung 361
 - Zeichnungsmethoden 356
 - Zeitleiste 167, 335
 - Zielpfad 181, 348
 - Zugriff 343
- Movieclip -> s. Filmsequenz
- Mozilla 20
- MSIE 135
- multiline 393

- MySQL 570
 - phpMyAdmin 597
 - mysql_connect (PHP) 598

N

- name-Attribut (HTML) 112
- Namenskonventionen 192
 - JavaScript 35
- NaN 603
- navigator (JS) 72
- navigator.appName (JS) 72
- navigator.userAgent (JS) 72
- Netscape 19, 437, 524
- Netscape Navigator
 - DHTML 133
- new 276, 287, 288
- new (JS) 55
- news 31
- News-Ticker 591
- nextFrame 256, 375
- nextScene 256
- nextSibling 508, 513
- nodeName 508, 518
- nodeType 508, 518
- nodeValue 509, 519
- Normalmodus 172, 178
 - Aktion hinzufügen 178
 - Ausdruck 180
 - Funktionen 246
 - Kommentare 194
- null 195
- null (JS) 58
- Number 202

O

- Object 196
- Objekt 283
 - Datentyp 196
 - instanziiieren 288
- Objektaktionen 162
- Objekte
 - auflisten 416
 - Datum 320
 - JavaScript 61
 - Key 533
 - Sound 403
 - System 409
 - Textfeld 389
 - Textformat 395
 - vorgefertigte 300

- Wahrheitswert 319
- with 366
- XML 497, 502
- Objektorientierung 283
 - Begriffe 283
 - Eigenschaften 284
 - JavaScript 60
 - Klasse 284
 - Methoden 284
 - Polymorphismus 286
 - Vererbung 285
- onblur (JS) 77
- onchange (JS) 77
- onChanged 394
- onclick (JS) 32, 77
- onClipEvent 324, 367
- onData 370, 512
- onDragOut 370
- onDragOver 370
- onEnterFrame 370
- onfocus (JS) 77
- onKeyDown 370
- onKeyUp 370
- onKillFocus 370, 394
- onLoad 370, 409, 511
- onload (JS) 77, 84
- onMouseDown 371
- onMouseMove 371
- onmouseout (JS) 78, 99
- onmouseover (JS) 77, 99
- onMouseUp 371
- onPress 371
- onRelease 371
- onReleaseOutside 371
- onRollOut 371
- onRollOver 371
- onScroller 394
- onSetFocus 371, 394
- onSoundComplete 409
- onsubmit (JS) 110
- onUnload 371
- onunload (JS) 78
- open (JS) 479
- Operatoren 204
 - arithmetisch 204
 - bitweise 210
 - Dekrement 205
 - in der Praxis 230
 - Inkrement 205
 - JavaScript 38

- Logisch 209
- Vergleich 206
 - Datentypen 208
 - Strings 207
 - Zuweisung 204
- outerHeight (JS) 85
- outerWidth (JS) 85

P

- Pan 473
- Parameter 242
 - auslesen 246
 - Klassen 291
- parentNode 509
- parseFloat 255
- parseInt 255, 485
- parseInt (JS) 74
- parseXML 505
- password 393
- PercentLoaded 473
- PHP 463, 569
 - php.ini 464
 - Session-Management 464
- phpMyAdmin 597
- PI 329
- Play 473
- play 256, 375, 414
- Polymorphismus 286
- pop 317
- Position 352
- position 409
- POST 573
- Potenz 334
- pow 334
- Preloader 425, 476
 - gesamter Film 432
- Preloader -> s. Vorlader
- press 163
- prevFrame 256, 375
- previousSibling 509
- prevScene 256
- print 269
- printAsBitmap 269
- printAsBitmapNum 269
- printNum 269
- Projektor 442
- prototype 294
- Prozeduren 57
- push 317
- push_exec 440

push_fullscreen 440
push_quit 440

Q

quit 439

R

random 330, 334
random (JS) 69
ReadyState 474
Referenz 187
Rekursive Funktionen 249
release 163
releaseOutside 163, 267
removeListener 395
removeMovieClip 263, 364, 375, 386
removeNode 505
removeTextField 395
replaceSel 395
restrict 393
return 245
return (JS) 57
reverse 317
Rewind 473
rightMargin 397
rollOut 163
Rollover
 JavaScript 99
rollOver 163
round 330, 334
RSS 524
 RSS-DEV 524
Runden 330

S

Schaltfläche
 Ereignisse 162, 163
 Symbol 167
 Zustand 163, 167
Schaltflächen
 in der Praxis 229
Schaltflächenereignisse 163
Schaltflächenereignisse -> s. Objektak-
 tionen
Schaltflächensymbol 167
Schleifen 219
 Endlos 221
Schlüsselbilder
 Ereignisse 162
Schlüsselwörter 193

screenResolutionX 410
screenResolutionY 410
screenX (JS) 85
screenY (JS) 85
scroll 393
Scrollbar 522, 559
scrollbars (JS) 85
selectable 393
selected (HTML) 114
selectedIndex (JS) 115
send 506
sendAndLoad 506
Serverseitig 569
 Mailversand 584
Session 464
 Cookies 464
 ID 464
session_id (PHP) 465
session_name (PHP) 465
session_start (PHP) 465
set 267
set variable 268
setDate 326
setEnabled 563
setFullYear 327
setHours 327
setInterval 273, 427, 541, 593
 in der Praxis 322
setInterval (JS) 89
setMask 375
setMilliseconds 327
setMinutes 327
setMonth 327
setNewTextFormat 395
setPan 408
setProperty 264, 373
setSeconds 327
setTextFormat 395
setTime 327
setTimeout (JS) 89
setTransform 408
SetVariable 473, 490
setVolume 409
setYear 327
SetZoomRect 473
shift 317
showmenu 439
siehe Funktionen -> s. Prozeduren
sin 334
size 396

- slice 315, 317
- Sonderzeichen 198
- sort 318
- sortOn 318
- Sound 401
 - abspielen 535
 - anhalten 257
 - Balance 408
 - Eigenschafteninspektor 404
 - Ereignis 402
 - exportieren 406
 - global 403
 - importieren 402
 - in der Praxis 544
 - Loop 402
 - Objekt 403
 - Streaming 402
 - transformieren 408
 - Vorladezeit 304
- Spiele 533
- splice 318
- split 308, 315
- SQL 597
- sqrt 334
- SQRT1_2 329
- SQRT2 329
- src (JS) 96
- start 407, 409
- startDrag 264, 376
- Statische Textfelder 169
- status 509
- status (JS) 85
- stop 257, 376, 409, 414, 433, 587
- stopAllSounds 257, 403
- stopDrag 264, 376
- StopPlay 473
- Streaming
 - Sound 402
- Strikte Gleichheit 209
- Strikte Ungleichheit 209
- String 195, 196, 202
 - JavaScript 36
 - Konkatenation 204
 - length 310
 - Objekt 306
 - Sonderzeichen 198
 - verbinden 204
 - vergleichen 207
- String -> s. Zeichenkette
- substr 315
- substring 312, 316
- substring (JS) 43
- Suchen und Ersetzen 176
- super 298
- swapDepths 376
- swf 169
- switch 215
- switch (JS) 52
- switch case 215
 - in der Praxis 232
- swLiveConnect 448, 490
- Symbol
 - Eigenschaften 360
 - Exportieren 361
- Symbole 165
 - Art ändern 166
 - Grafik 167
 - In Symbol konvertieren 166
 - Movieclip 167, 336
 - Neues Symbol 166
 - Registrierungspunkt 166
 - Schaltfläche 167
 - suchen 188
- Szenen 168, 256
 - Bedienfeld 168
 - testen 416

T

- tabIndex 393
- tabStops 397
- Tags 498
- tan 334
- target 397, 460
- target (HTML) 151
- targetPath 255
- Taschenrechner 224
- Tastatur 533
- Tastatursteuerung 163
- Tastenkombinationen 187
- TCallFrame 475
- TCallLabel 475
- TCurrentFrame 475
- TCurrentLabel 475
- Tell Target 366
 - Methoden 474
 - TCallFrame 475
 - TCallLabel 475
 - TCurrentFrame 475
 - TCurrentLabel 475
 - TGetProperty 475

- TGetPropertyAsNumber 475
- TGotoFrame 475
- TGotoLabel 476
- TPlay 476
- TSetProperty 476
- TStopPlay 476
- tellTarget 275
- Testen 169, 415
 - Debugger 419
 - Debug-Modus 420
 - Menü Fehlersuche 431
 - Objekte auflisten 416
 - Szene 416
 - Übertragungsrate 431
 - Variablen auflisten 416
- text 393
- textColor 393
- Textfelder 168
 - dynamisch 169, 196
 - Eigenschaften 390
 - Eingabe 169, 585
 - Ereignisse 394
 - formatieren 395
 - in der Praxis 398
 - mehrzeilig 196
 - Methoden 394
 - mit ActionScript 389
 - statisch 169
 - Textformat 395
 - Variable 197
 - Zeilenumbruch 198
- Textformat 395
 - Eigenschaften 396
- textHeight 393
- Text-Werkzeug 169
- textWidth 393
- TGetProperty 475
- TGetPropertyAsNumber 475
- TGotoFrame 475
- TGotoLabel 476
- this 288, 344
- Tiefe 352
- Tiefe -> s. Position
- toggleHighQuality 275, 303
- toLocaleString (JS) 69
- toLowerCase 316
- toLowerCase (JS) 73
- toolbar (JS) 85
- top (JS) 85
- toString 203, 318, 319, 327, 506

- TotalFrames 473
- totelframes 434
- toUpperCase 316
- TPlay 476
- trace 170, 273, 417
 - für Veröffentlichen deaktivieren 419
- trapallkeys 439
- true 195, 206
- true (JS) 41
- TSetProperty 476
- TStopPlay 476
- type 393
- type (HTML) 29
- typeof 196
- Typkonvertierung
 - automatisch 201
 - explizit 202

U

- u 199
- UI Komponenten 170, 522, 551
- undefined 192, 195, 603
- underline 396
- Unendlichkeit 201
- unescape 254, 573
- Unicode 199
- unload 369
- unload/Movie 261, 364, 376, 386
- unload/MovieNum 261
- unshift 318
- updateAfterEvent 264
- URL 260
- url 396
- UserLand 524
- UTC-Zeit 328

V

- value-Attribut (HTML) 113
- valueOf 319, 327
- Variablenundefined 192
- Variable
 - setzen 267
- variable 394
- Variablen 191
 - auflisten 416
 - Datentypen 195
 - Deklaration 191
 - Funktion 239
 - global 241
 - Gültigkeitsbereich 194

- JavaScript 35
 - lokal 240
 - Namenskonventionen 192
- VBScript 447
- Vererbung 285
 - Mehrfachvererbung 296
- Vergleichsoperatoren 206
- Veröffentlichen
 - Einstellungen 419
 - HTML 445
 - in der Praxis 545
 - mit FSCCommand 444
 - Projektor 442
- Video 410
 - abspielen 414
 - anhalten 414
 - exportieren 412
 - importieren 410
 - zurückspulen 414
- visibility (JS) 491
- Vollbild 450
- Voreinstellungen 184
- Vorladen
 - JavaScript 98, 102
- Vorlader 425

W

- W3C 132, 497
 - DHTML 136
- W3C -> s. World Wide Web Consortium
- Wahrheitswerte 195
 - JavaScript 37
- Warnmeldungen 186
- Webbrowser 444
- Weiterleitung 76
- Werbebanner
 - DHTML 148
- while 219, 278
- while (JS) 47
- width (HTML) 96
- width (JS) 85
- window (JS) 471, 604
- window.alert (JS) 31, 79
- window.close (JS) 453
- window.confirm (JS) 80
- window.innerWidth (JS) 149
- window.open (JS) 82, 463
- window.pageXOffset (JS) 150
- window.pageYOffset (JS) 150

- window.prompt (JS) 81, 120
- with 275, 366
- wordWrap 394
- World Wide Web Consortium 132
- write (JS) 479
- Wurzel 330, 334

X

- XHTML 455
- XML 497
 - Anwendungen 510
 - Daten laden 510
 - formatieren 516
 - Hierarchie 503
 - Regeln 498
 - Root-Element 500
 - testen 501
 - wohlgeformt 500
 - XMLNode 504
- xmlDecl 509
- XSLT 501

Y

- Y2K
 - JavaScript 65

Z

- Zahl 195, 199
 - Infinity 201
 - JavaScript 36
 - Number 202
- Zeichenkette 195
- Zeichnungsmethoden 356
 - Matrix 358
 - Quadrat 358
 - Verlauf 357
- Zeilenumbruch 198
- Zeitleiste 156
 - Haupt 163
- Zieleditor 348
- Zielpfad
 - Darstellung 350
 - einfügen 181, 348
 - Modus 349
- Zielpfad -> s. Zieleditor
- Zoom 473, 483
- Zufallszahl 330
- Zustände
 - Aktiv 271